Escola d'Enginyeria de Telecomunicació i Aeroespacial de Castelldefels

UNIVERSITAT POLITÈCNICA DE CATALUNYA

FINAL DEGREE PROJECT

TFG TITLE: MRO 4.0: Redesign and enhancement of an aircraft maintenance management application

BACHELOR: Degree in Airport Engineering

AUTHOR: Óscar Martínez Sánchez

DIRECTOR: José Antonio Castán Ponz

DATE: 30 de septiembre del 2021

Overview

This Final Degree Project shows the improvements and changes made to the AirDocs application created and developed by Laura González Huguet in her TFG project and whose reading date was 22/12/2020. These changes have been introduced to improve the user experience when using it.
AirDocs 2.0 is characterised by the entry of data about repairs and inspections of aircrafts with relevant documentation, so several adjustments have been made to reduce the amount of data the user has to enter with keyboard and reduce the time spent on filling in forms or documentation.
Its code has been completely redone with Android Studio; a programme specialised in the creation of Android applications. Previously, DropBox was used for cloud storage of the app data which required the administrator being online for the application to work. To overcome this, the cloud storage method has been changed and replaced by Google's Firebase with a dedicated server for AirDocs 2.0 which is always online.
New features have been introduced, for example, a registration screen where the user fills in a form so that the administrator knows which users have access to the application or the possibility to upload photos of the damage that the technicians carry out.
Each type of user (administrator, engineer, or cabin crew) has a separate home screen, with details of their account, and this has been done in such a way that the user can navigate intuitively and visually comfortable. All menus have been redesigned or condensed in such a way that a user is in control of their actions and not induce a sense of disorientation when searching for certain sections.
The PC version of AirDocs 2.0 has disappear and has been replaced with a method which the user connects their devices and displays AirDocs 2.0 application on the computer screen.
Overall, AirDocs 2.0 has been reinvented with improvements and new features to provide users with the best experience.

TÍTULO DEL TFG: MRO 4.0: Rediseño y mejora de una aplicación para la gestión en el mantenimiento de aeronaves

TITULACIÓN: Grado en Ingeniería de Aeropuertos

AUTOR: Óscar Martínez Sánchez

DIRECTOR: José Antonio Castán Ponz

DATA: 8 de septiembre del 2021

Resumen

Este Trabajo Fin de Grado muestra las mejoras y cambios realizados sobre la aplicación AirDocs creada y desarrollada por Laura González Huguet en su proyecto TFG y cuya lectura fue con fecha 22/12/2020. Estos cambios se han introducido para mejorar la experiencia del usuario al utilizarla.
AirDocs 2.0 se caracteriza por la introducción de datos sobre reparaciones e inspecciones de las aeronaves con la documentación pertinente, por lo que se han realizado varios ajustes para reducir la cantidad de datos que el usuario tiene que introducir con el teclado y reducir el tiempo dedicado a rellenar formularios o documentación.
Su código se ha rehecho por completo con Android Studio; un programa especializado en la creación de aplicaciones para Android. Anteriormente, se utilizaba DropBox para el almacenamiento en la nube de los datos de la aplicación, lo que requería que el administrador estuviera conectado para que la aplicación funcionara. Para superar esto, se ha cambiado el método de almacenamiento en la nube y se ha sustituido por Firebase de Google con un servidor dedicado para AirDocs 2.0 que siempre está en línea.
Se han introducido nuevas características, por ejemplo, una pantalla de registro en la que el usuario rellena un formulario para que el administrador sepa qué usuarios tienen acceso a la aplicación o la posibilidad que subir fotos de los daños que los técnicos realicen.
Cada tipo de usuario (administrador, ingeniero o tripulante de cabina) tiene una pantalla de inicio independiente, con los detalles de su cuenta, y se ha hecho de forma que el usuario pueda navegar de forma intuitiva y visualmente cómoda. Todos los menús han sido rediseñados o condensados de forma que el usuario tenga el control de sus acciones y no le provoque una sensación de desorientación al buscar determinadas secciones.
La versión para PC de AirDocs 2.0 ha desaparecido y ha sido sustituida por un método en el que el usuario conecta sus dispositivos y muestra la aplicación AirDocs 2.0 en la pantalla del ordenador.
En general, AirDocs 2.0 se ha reinventado con mejoras y nuevas funciones para ofrecer a los usuarios la mejor experiencia.

TÍTOL DEL TFG: MRO 4.0: Redisseny i millora d'una aplicació per a la gestió en el manteniment d'aeronaus

TITULACIÓ: Grau en Enginyeria d'Aeroports

AUTOR: Óscar Martínez Sánchez

DIRECTOR: José Antonio Castán Ponz

DATA: 8 de setembre de 2021

Resum

Aquest treball de fi de grau mostra les millores i canvis realitzats en l'aplicació AirDocs 2.0 creada i desenvolupada per Laura González Huguet en el seu projecte de TFG i la data de lectura va ser el 2020/12/22. Aquests canvis s'han introduït per a millorar l'experiència de l'usuari en utilitzar-la.
AirDocs 2.0 es caracteritza per la introducció de dades sobre reparacions i inspeccions de les aeronaus amb la documentació pertinent, per la qual cosa s'han realitzat diversos ajustos per a reduir la quantitat de dades que l'usuari ha d'introduir amb el teclat i reduir el temps dedicat a emplenar formularis o documentació.
El seu codi s'ha refet per complet amb Android Studio; un programa especialitzat en la creació d'aplicacions per a Android. Anteriorment, s'utilitzava DropBox per a l'emmagatzematge en el núvol de les dades de l'aplicació, la qual cosa requeria que l'administrador estigués connectat perquè l'aplicació funcionés. Per a superar això, s'ha canviat el mètode d'emmagatzematge en el núvol i s'ha substituït per Firebase de Google amb un servidor dedicat per a AirDocs 2.0 que sempre està en línia.
S'han introduït noves característiques, per exemple, una pantalla de registre en la qual l'usuari emplena un formulari perquè l'administrador sàpiga quins usuaris tenen accés a l'aplicació o la possibilitat que pujar fotos dels danys que els tècnics realitzin.
Cada tipus d'usuari (administrador, enginyer o tripulant de cabina) té una pantalla d'inici independent, amb els detalls del seu compte, i s'ha fet de manera que l'usuari pugui navegar de manera intuïtiva i visualment còmoda. Tots els menús han estat redissenyats o condensats de manera que l'usuari tingui el control de les seves accions i no li provoqui una sensació de desorientació en buscar determinades seccions.
La versió per a PC de AirDocs 2.0 ha desaparegut i ha estat substituïda per un mètode en el qual l'usuari connecta els seus dispositius i mostra l'aplicació AirDocs 2.0 en la pantalla de l'ordinador.
En general, AirDocs 2.0 s'ha reinventat amb millores i noves funcions per a oferir als usuaris la millor experiència.

INDEX

FIGURE INDEX

<u>**CHAPTER 3. AIRDOCS 2.0: HOW IT WORKS**</u>

TABLE INDEX

GLOSSARY

AMT: Airplane maintenance technician

API: Application Programming Interface

APK: Android Application Package

ATA: Aircraft

BaaS: Backend as a Service

CAMO: Continuing Airworthiness Maintenance Organization

CRS: Certificate of Release to Service

FC: Flight Cycle

FH: Flight Hour

IDE: Integrated Development Environment

JVM: Java Virtual Machine

MRO: Maintenance, Repair and Overhaul.

NDK: Native Development Kit

SB: Service Bulletin

SDP: Software Development Platform

SDT RB: Structural Damage Task, Report Back

SQL: Structured Query Language

UI: User Interface

INTRODUCTION

AirDocs was created with the purpose of removing paper documentation and replace it with a digital record in a cloud storage. The data that employees enter the application corresponds to the maintenance and/or other processes of a fleet of aircraft belonging to a company. This process facilitates access to certain documents at any place and at any time, thus facilitating the speed with which employees can perform the tasks they have to carry out.

This project is the continuation of Laura González Huguet's TFG to improve AirDocs first version, both in the user interface and the application code. The objective of these changes is to improve the user experience both visually and functionally. The application has been remade from scratch with Android Studio, although it must be said that in essence the main objective of AirDocs that Laura González Huguet developed has not changed, but improvements have been implemented to make the new AirDocs 2.0, an application that the user will find easy and quick to use and will have the desire to work with it again.

This project consists of three main chapters. The first part consists of the method and working environment with which this project has been remade. It describes which software has been used for the development of the new AirDocs 2.0 code and why this was chosen. It also proceeds to explain the change of AirDocs first version cloud storage, which has been replaced with Firebase. Firebase is owned by Google and one of its main features is real-time access to it. On the other hand, the language used to recode AirDocs 2.0 is and in this chapter is explain what disadvantages and advantages it has over other programming languages.

The second chapter is the thickness of the project. Here we proceed to the detailed description of each new features and all the new functionalities implemented. It also specifies how the data related to aircraft maintenance is entered, as the interface has been changed and the way the forms are filled in is now different from before. In addition, significant improvements have been implemented in user registration and data handling. All of this is specified in detail in its section.

The third and last chapter describes the new way of using AirDocs 2.0 on a PC. Now it dispenses of a conventional programme running on Windows and the users’ device is connected directly to their PCs on which the application is installed to display it on the PC screen. It is specified which programme is used to carry out this action and the procedure for connecting to the PC.

Finally, in the annexes there is the main code that makes it possible for AirDocs 2.0 to work.

CHAPTER 1. WORK ENVIRONMENT

This chapter explains how AirDocs 2.0 was remade and the tools to accomplish it. It also describes the main features of Android Studio, the working environment used for the building of the new app and the pros and cons of using this instead of other programming environments. In addition, it is exposed which is the new cloud database and how it was implemented.

1.1. Android Studio

Android Studio is an integrated development environment (IDE) that provides tools and methods for developing Android applications. It is based on IntelliJ IDEA, which replaces Eclipse, the previous IDE used by Android Studio. The objective of Android Studio is to set up a working environment dedicated to programming of applications for Android devices and adapting these to the evolution of the system through the time. In this work environment the user can find the necessary tools to create their desire apps with the maximum help it can provide so that even an inexperienced user can get started in the world of programming an Android app. The app development includes from the coding of the app to the user interface and when the app is finally done it can be published in the official Google store called Play Store.

Figure 1.1 Android Studio logo

In addition to IntelliJ's powerful code editor, methods, and developer tools, it offers even more characteristics that improves the user experience and productivity when creating apps, such as:

- Flexible Gradle-based build system
- Fast and feature-rich emulator
- Unified environment to developing apps for all Android devices
- Apply changes to push code and resource changes to run apps without restarting.
- Code templates and GitHub integration to help building common app features and import sample code

- Extensive testing tools and frameworks
- C++ and NDK support
- Built-in support for Google Cloud Platform

Android Studio is constantly evolving, so from time-to-time developers release updates to improve and add new features. The following sections describe the main differences that make this application developer so widely used in the programming world.

1.1.1. Main features

The main features that make Android Studio a perfect tool for app development are the next ones:

- **Layout editor:** The layout of the app can be easily and quickly built by adding a wide variety of attributes. These can be introduced either by hand coding or, the most used, dragging UI elements into the visual design editor. The design editor also gives you a preview of the layout that the user is creating so it shows how it will display on the different screen sizes and different layouts, both landscape and portrait mode.

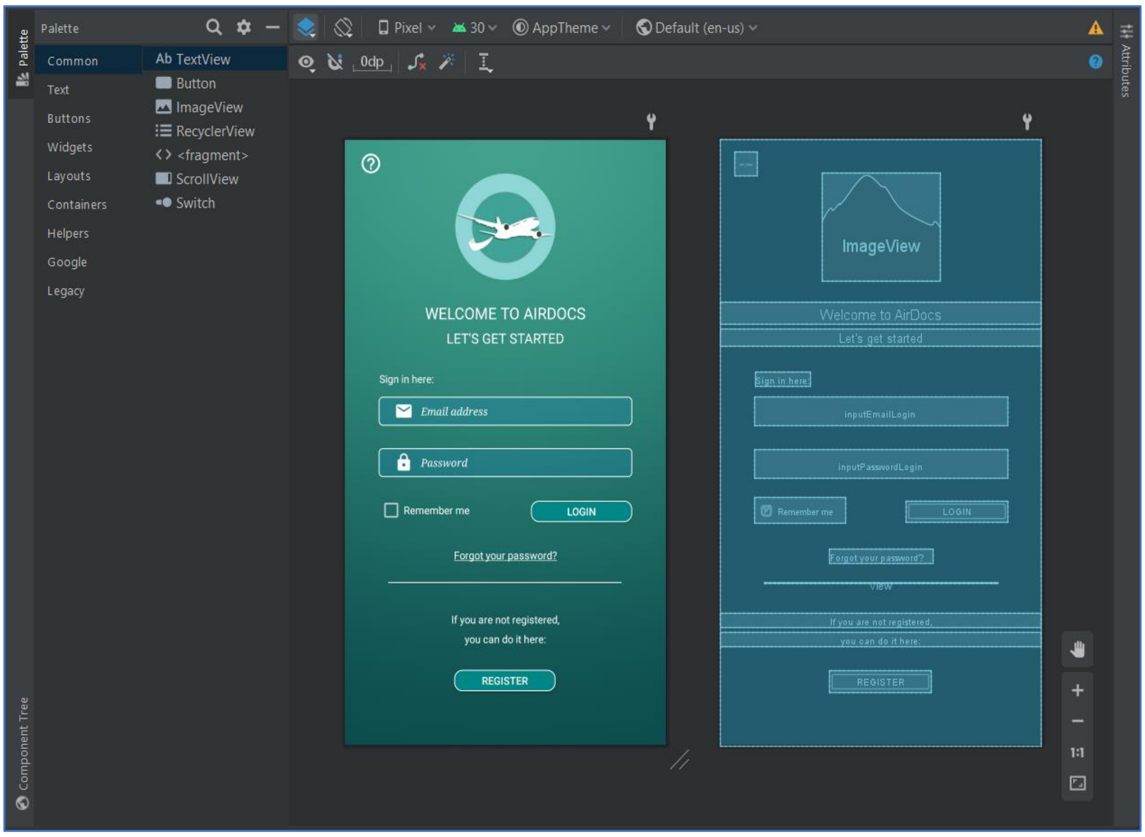


Figure 1.2 Android Studio editor layout

- **Templates:** One important characteristic is that Android Studio gives the most common templates that mostly apps are using. For example, if the user desires to implement a navigation drawer, it is not necessary to code all of it from the beginning, instead Android Studio gives a template that makes the task much easier and quicker.

- **Kotlin support:** Kotlin is a programming language for Android. In this project, Kotlin is used for building AirDocs 2.0. This is widely explained on section 1.2.

- **Integrated emulator:** Android Emulator emulates an Android device on the computer so that the user can test the application on various devices and Android API levels without having to own a specific Android device. The emulator gives the user all the functions of a real device. Testing the application on this emulator is faster and easier than testing it on a physical device.

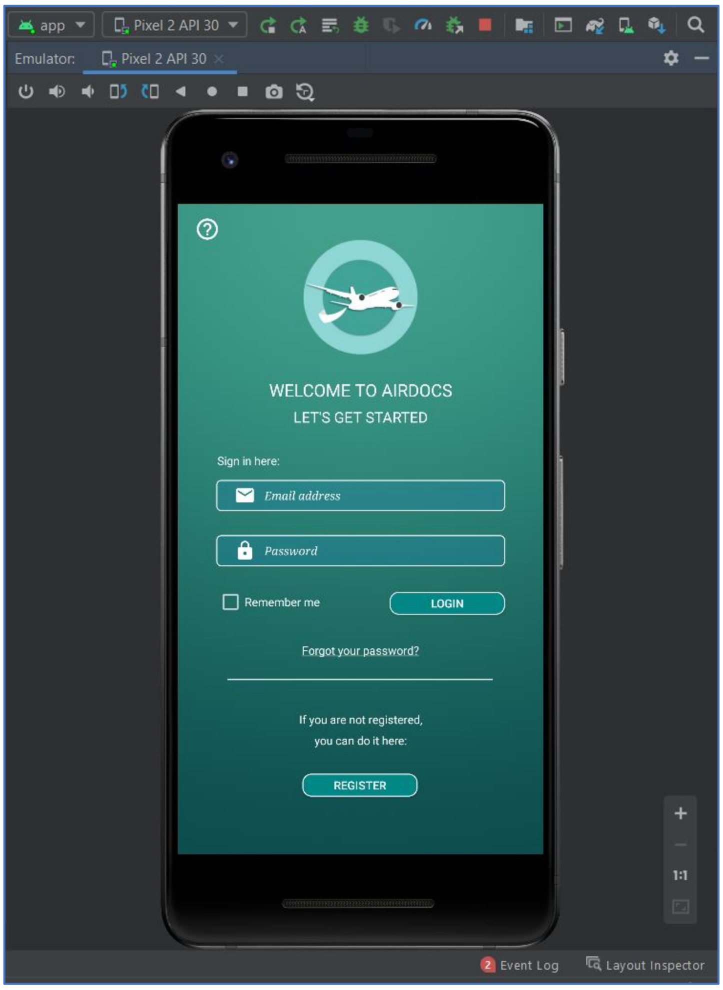


Figure 1.3 Android Studio integrated emulator

- **Fast running**: Running the app is easier thanks to the technology within it that understands what changes have been done and deliver it instantly without the need for rebuilding the APK.

- **Code editor:** Android Studio provides the user with a smart and fast code editor. This helps to write a precisely code with no errors in it. It gives recommendations to complete the code and analyse the best options for the app before running it.

- **Different screen devices**: Android Studio can build apps for various android devices such as tablets, watches or even TVs.

- **Connecting with Firebase:** Android Studio helps provide a real-time experience project development through dynamic upgrades in the app. Connecting with Firebase helps to create direct updates and provides database connections. This and more are explained in section 1.3.

1.2. Kotlin programming language

Figure 1.4 Kotlin logo

Kotlin is a free, open-source programming language initially designed for the Java Virtual Machine (JVM). It is mainly focused on interoperability, safety, clarity, and tooling support. Kotlin's future is predicted to be limited not only to Android users but also to iOS (Apple's operating system). Kotlin was created and later updated by JetBrains, the company behind IntelliJ IDEA and has been open source since 2012. Although it can be used where Java can be seen, it mainly objective is for Android app development, which has an official support by Google. Some important companies are already using it, such as Google, Pinterest, Kickstarter Uber and more. This programming language is going to be very present in all our devices in the imminent future.

So, why use Kotlin for the development of AirDocs 2.0 instead of Java or other programming languages? The next points clarify the reason for choosing it.

- **Reliability:** One characteristic of using Kotlin is that it gives you no margin for errors, supplying the user with a simple but reliable codebase. Kotlin allows the user writing less code, so it implies fewer bugs. When the user let the framework taking care of little coding aspects, the user can focus more on the gross of the code. When the compilation is running, it can determine if there is any potential error. In comparison, Kotlin is a safer language than Java is.

- **Learning:** Kotlin expects to enhance the highlights of Java not simply change them. All projects that involve Kotlin can also be subject to all the skills that developers learn and acquire when using Java.

- **Maintainability:** Many IDEs are supported by Kotlin which one of them is Android Studio and others SDK tools. The developer productivity is also raised and maximized because of the continuous management of the toolkits they are frequently using.

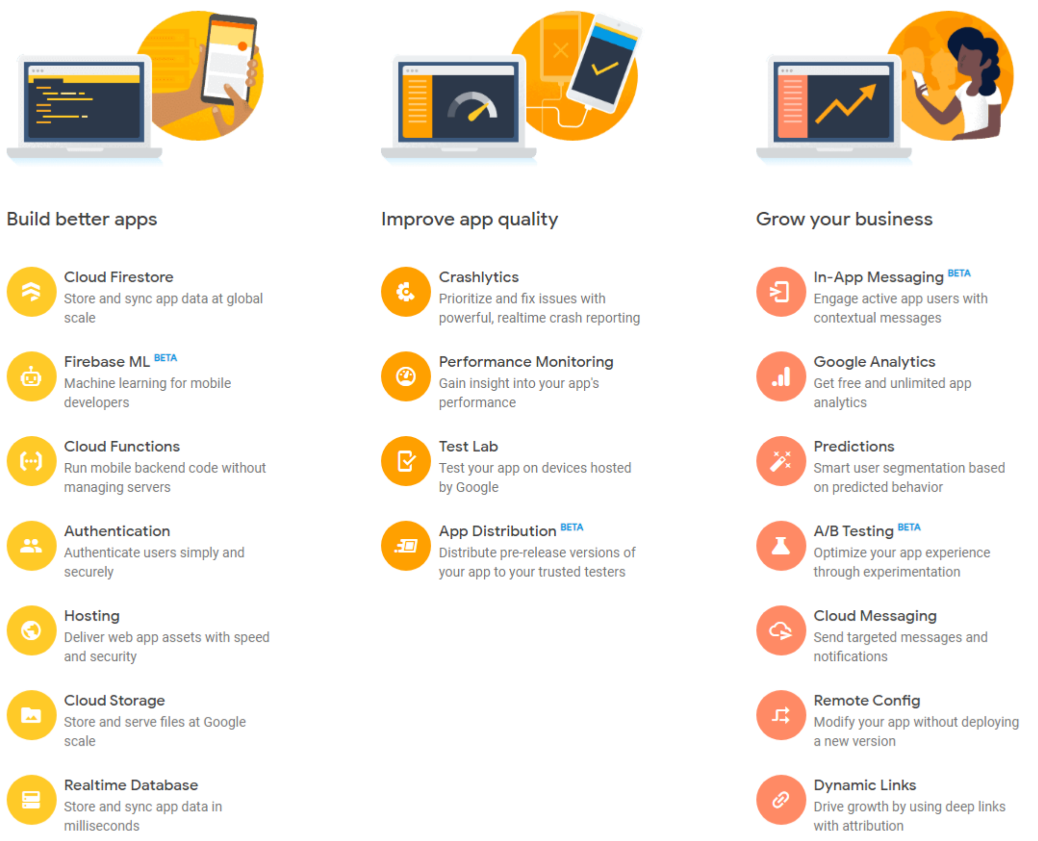
- **Existing Java code:** Kotlin can use Java code parts and vice versa, so this makes possible the direct translation to Kotlin and the two can be use at the same time.

- **Productivity increased:** Being compact, clear, and efficient code, it has a concise and intuitive syntax. This makes an increase of the user productivity making him to write less code and running it. So, overall, contributes to better code maintainability.

1.3. Firebase

Figure 1.5 Firebase logo. Source: Firebase.

Firebase is a software development platform (SDP) launched by Firebase company in 2011 and was acquired later by Google in 2014. It started as a real-time database, but nowadays it has more than 18 services and dedicated APIs. Firebase gives the user multiple tools to build and improve their application. It also supplies with services that the developers would normally have to build themselves. Some of these services are analytics, account authentication, databases, file storage and more (see Figure 1.6). All of them are in the cloud which is easy to access and manipulate in real time and whenever the user needs to.



Build better apps

Cloud Firestore
Store and sync app data at global scale

Firebase ML BETA
Machine learning for mobile developers

Cloud Functions
Run mobile backend code without managing servers

Authentication
Authenticate users simply and securely

Hosting
Deliver web app assets with speed and security

Cloud Storage
Store and serve files at Google scale

Realtime Database
Store and sync app data in milliseconds

Improve app quality

Crashlytics
Prioritize and fix issues with powerful, realtime crash reporting

Performance Monitoring
Gain insight into your app's performance

Test Lab
Test your app on devices hosted by Google

App Distribution BETA
Distribute pre-release versions of your app to your trusted testers

Grow your business

In-App Messaging BETA
Engage active app users with contextual messages

Google Analytics
Get free and unlimited app analytics

Predictions
Smart user segmentation based on predicted behavior

A/B Testing BETA
Optimize your app experience through experimentation

Cloud Messaging
Send targeted messages and notifications

Remote Config
Modify your app without deploying a new version

Dynamic Links
Drive growth by using deep links with attribution

Figure 1.6 Firebase services. Source: TechBriefers

The main characteristic of Firebase is that it does not need to make HTTP calls to sync the data. Conventional database supplies the user with the data needed when it is requested, instead Firebase platform sends the information stored in the database as soon as it is updated in the application and automatically send it to all users. It is possible thanks that Firebase platform is a Backend as a Service (BaaS). It eliminates the need to supervise back-end databases and gets the corresponding tools, so Google, after all, oversees maintaining and operating the system. In conclusion, the client SDK supplied by Firebase interacts directly with these back-end services, without the need to build any software between the user app and the service, so developers only need to query the database in the app (see Figure 1.7).

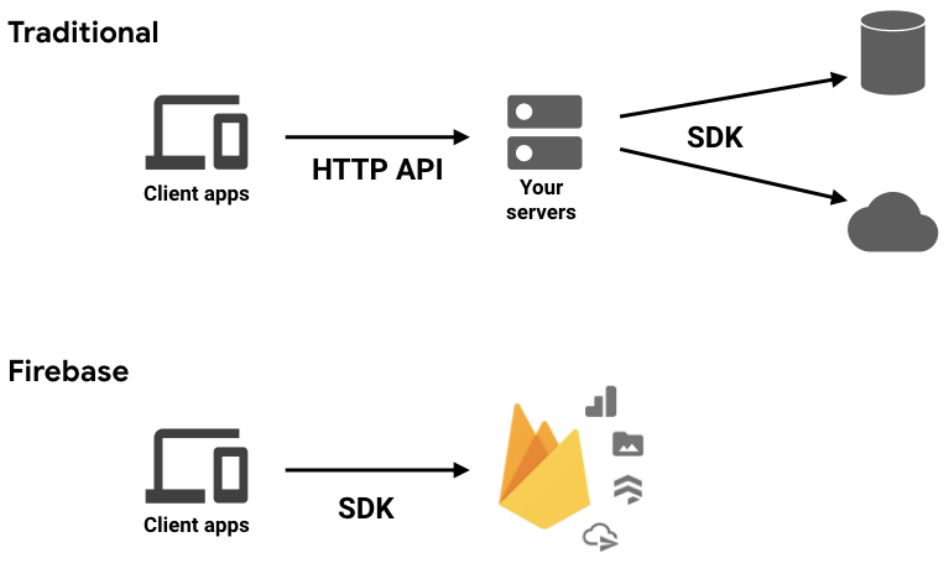


Figure 1.7 Connection scheme with Firebase. Source: GeeksforGeeks.

1.3.1. Firebase in AirDocs 2.0

Firebase provides cloud real-time synchronization with AirDocs 2.0. There are two main databases: Cloud Firestore and Realtime Database.

- **Realtime Database:** It is the original database of Firebase. It stores and synchronizes data with NoSQL database hosted in the cloud. The data is synchronized in real-time to all users connected and even it is available when the app is offline. It uses JSON documents to store the information in a large tree scheme and can use queries to look for specific data, but it has some limitations filtering and sorting.

- **Cloud Firestore:** This database use collections of documents to store the app data. The documents in a collection can contain subcollections or data fields. This kind of structure allows the user to store the information as tables and rows, extending the database's compatibility and can also be written directly. The information stored can be written as boolean, objects, arrays, numbers, strings, and null values. The user can also retrieve geographic points, time stamps, and shallow references to documents making it easier to manage data integrity and find type-based errors. When building the documents, references can be used to keep away data denormalization by minimizing the number of duplicated copies required. In addition, the user can query through the collections or subcollections to look for local copies of the information stored. In the same way that Realtime Database works, it can also work in real-time and when the app is offline.

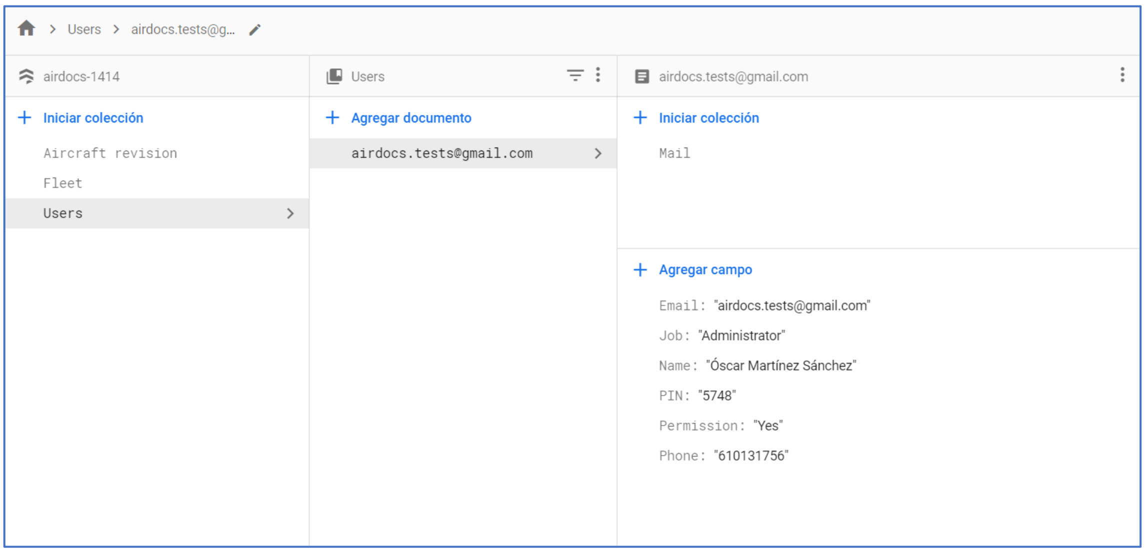
In conclusion, Cloud Firestore offers extra functionality, execution, and versatility on a framework intended to help more powerful features in the future. The best solution that fits AirDocs 2.0 is Cloud Firestore, so it has been integrated into the app.

1.3.1.1. User information’s structure

The user data is stored as we can see in Figure 1.8. First, the main collection named “Users” is created and holds all the data corresponding to registered users in the app. Inside the collection there are the documents named after the user email. The reason for naming it that way is because the app may contain two or more persons with the same name, but it cannot be two with the same email. The user information that is stored in AirDocs 2.0 is explained at Table 1.1.

Table 1.1 User registration information in Firebase server

Email	User’s email.
Job position	Describes the user’s job: administrator, CAMO engineer, MRO engineer, total access engineer, AMT and cabin crew.
Name	User’s full name.
PIN	PIN created by the user to confirm actions inside the app.
Permission	Shows if a user has permission to enter the app. Can be changed by the administrator to manage and control who uses the app.
Phone	Contact phone number.



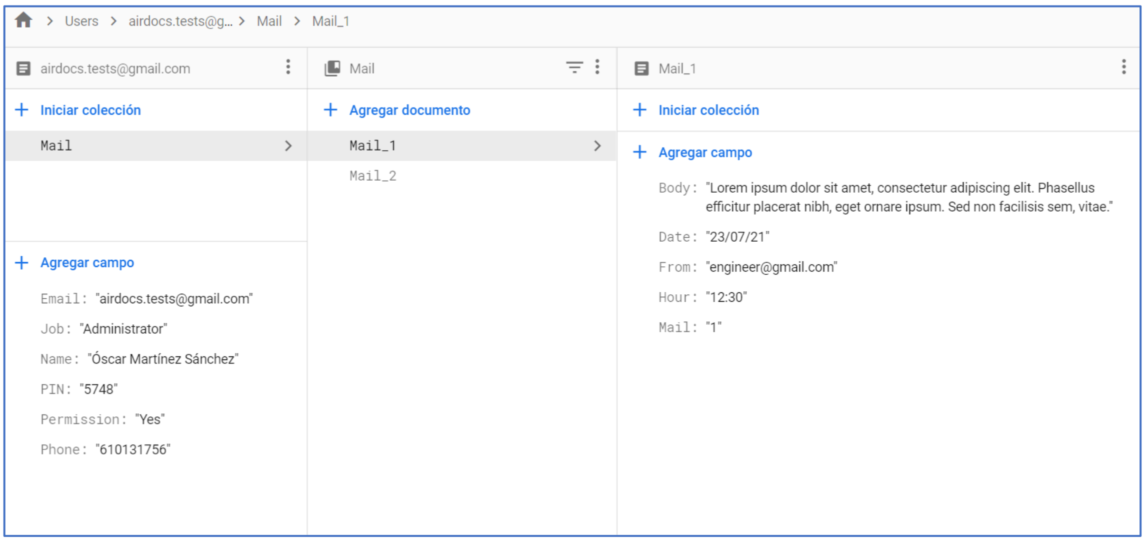
```
Email: "airdocs.tests@gmail.com"
Job: "Administrator"
Name: "Óscar Martínez Sánchez"
PIN: "5748"
Permission: "Yes"
Phone: "610131756"
```

Figure 1.8 User information structure

Whitin the user information there is a subcollection named “Mail”. Here there are all mails sent by other users. Inside this subcollection there are one document per mail sent which is explained at Table 1.2 and the structure is shown at Figure 1.9.

Table 1.2 Mail information in Firebase server

Body	Mail content
Date	Date which the mail was sent
From	User who sent the mail
Hour	Hour which the mail was sent
Mail	Identification to track the mail

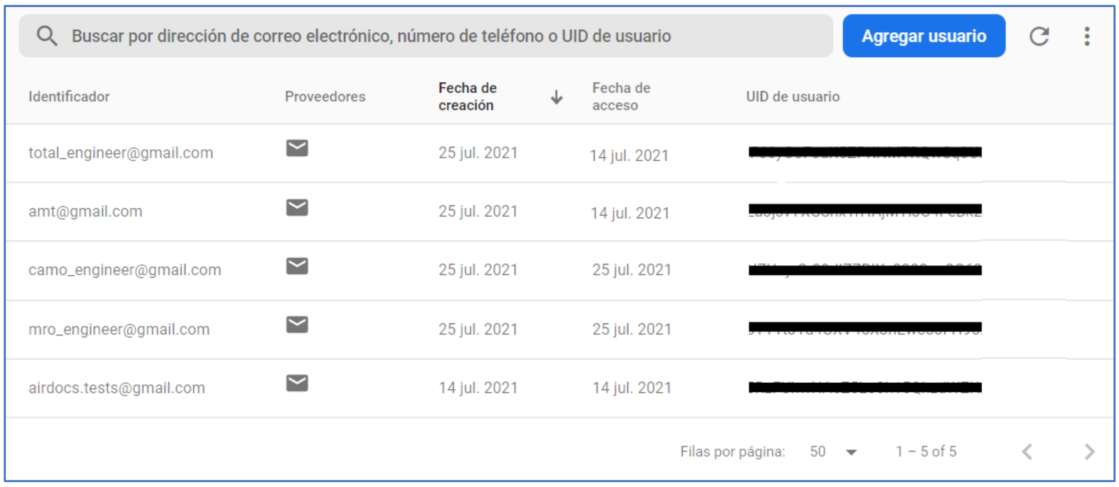


```
Email: "airdocs.tests@gmail.com"
Job: "Administrator"
Name: "Óscar Martínez Sánchez"
PIN: "5748"
Permission: "Yes"
Phone: "610131756"
```

```
Body: "Lorem ipsum dolor sit amet, consectetur adipiscing elit. Phasellus
      efficitur placerat nibh, eget ornare ipsum. Sed non facilisis sem, vitae."
Date: "23/07/21"
From: "engineer@gmail.com"
Hour: "12:30"
Mail: "1"
```

Figure 1.9 Mail information structure

Cloud Firestore uses the Authentication with email and password service to authenticate users to AirDocs 2.0. Firebase Authentication SDK provides methods to create and manage users that use their email addresses and passwords to sign in and stores it creating a unique UID for each one (see Figure 1.9). Firebase Authentication also handles sending password, reset emails and validation account.



Buscar por dirección de correo electrónico, número de teléfono o UID de usuario

Agregar usuario

Identificador	Proveedores	Fecha de creación	Fecha de acceso	UID de usuario
total_engineer@gmail.com	✉	25 jul. 2021	14 jul. 2021	[redacted]
amt@gmail.com	✉	25 jul. 2021	14 jul. 2021	[redacted]
camo_engineer@gmail.com	✉	25 jul. 2021	25 jul. 2021	[redacted]
mro_engineer@gmail.com	✉	25 jul. 2021	25 jul. 2021	[redacted]
airdocs.tests@gmail.com	✉	14 jul. 2021	14 jul. 2021	[redacted]

Filas por página: 50 1 – 5 of 5

Figure 1.10 User authentication service

In AirDocs 2.0 there is a verification system provided by Firebase. This system sends an email to the user when it is registered. When the link is clicked, it will automatically verify the user into the app.

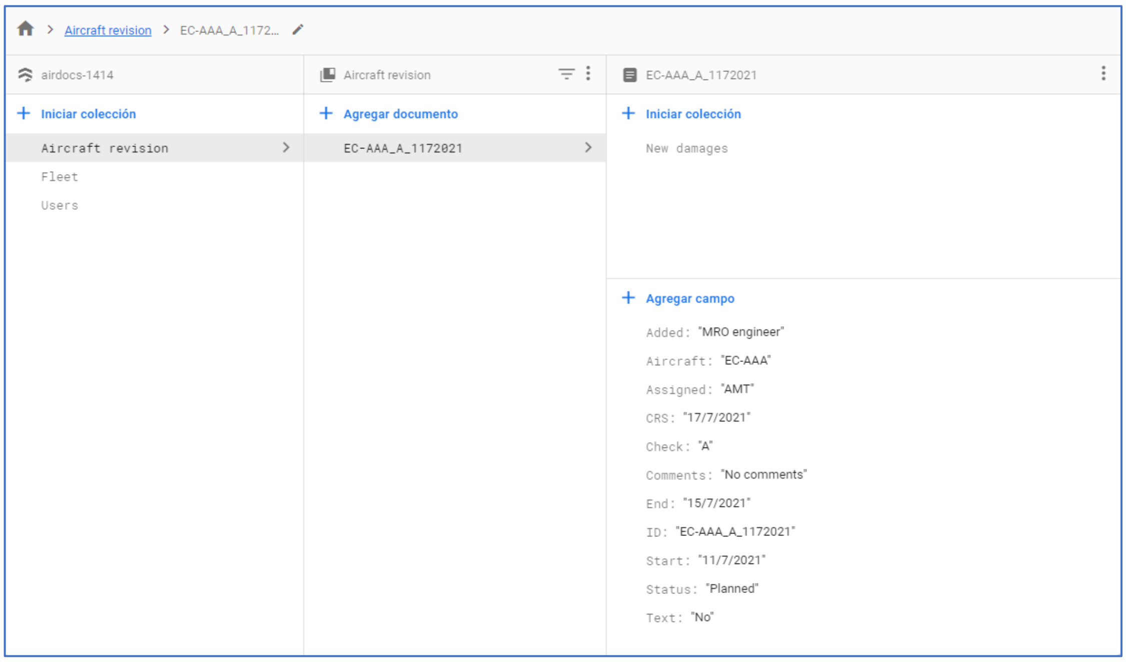
1.3.1.2. Aircraft check's structure

When an aircraft check is programmed, the information is stored in "Aircraft revision" collection and holds all the documents. The documents that contain all the data is named as follows: aircraft registration plus the type of check plus the date when it begins (ex. "ECC-AA_A_1172021"). This is done to avoid creating duplicate or rewriting the same document. For example, if an engineer creates a new check the app will send a warn if there is an existing document in the database. Table 1.2 shows the data fields inside the document.

Table 1.3 Aircraft check information in Firebase server

Added	The engineer's name that added the check.
Aircraft	Aircraft registration.
Assigned	AMT to which it has been assigned.
CRS	Date of entry into service.
Check	Type of check.
Comments	Comments added by the user.
End	Ending date.
ID	This identification is the same as the document's name. It is used for fetching the data.

Start	Starting date.
Status	Check status.
Text	Indicates whether there are comments or not.



> Aircraft revision > EC-AAA_A_1172...

airdocs-1414 | Aircraft revision | EC-AAA_A_1172021

+ Iniciar colección | + Agregar documento | + Iniciar colección

Aircraft revision | EC-AAA_A_1172021 | New damages

Fleet

Users

+ Agregar campo

```
Added: "MRO engineer"
Aircraft: "EC-AAA"
Assigned: "AMT"
CRS: "17/7/2021"
Check: "A"
Comments: "No comments"
End: "15/7/2021"
ID: "EC-AAA_A_1172021"
Start: "11/7/2021"
Status: "Planned"
Text: "No"
```

Figure 1.11 Aircraft check list structure

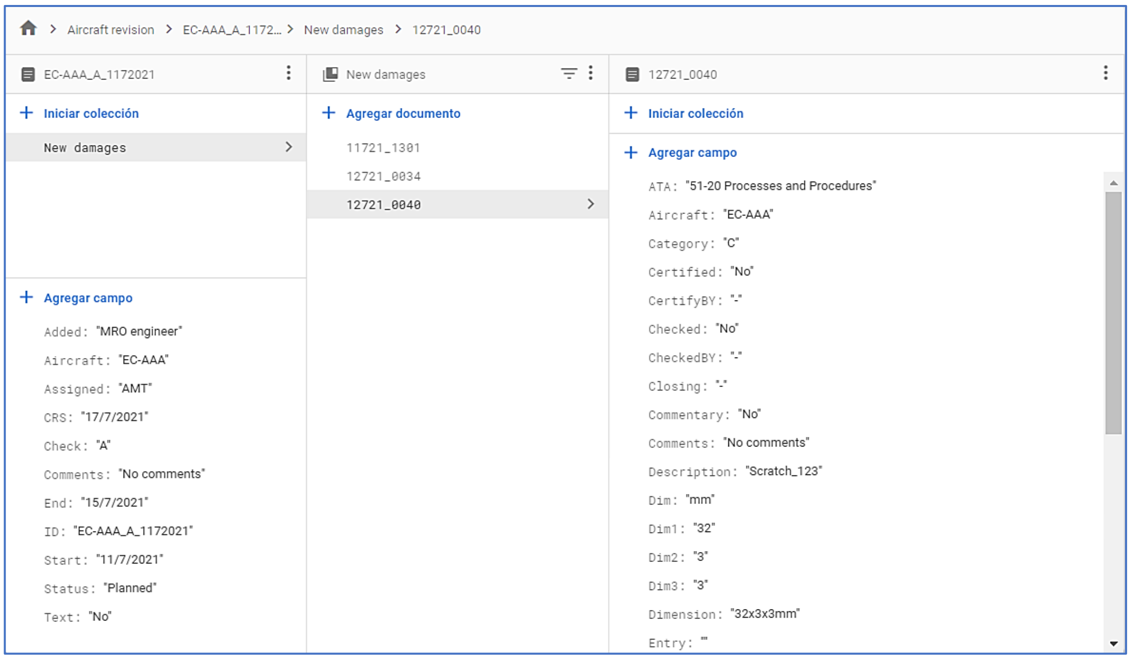
1.3.1.3. New damage's structure

Inside the aircraft check structure (see Figure 1.12) there is a subcollection named "New damages". In this subcollection there are all the damages that the AMTs adds. The document that contains all the data is named as follows: the date and hour when it is added (ex: 12721_0040). This allows the system not to have two or more duplicated or rewrite damages.

Table 1.4 Aircraft damage information in Firebase server

ATA	ATA chapter description.
Aircraft	Aircraft registration.
Category	Damage category (A, B or C).
Certified	Shows if the damaged is certified or not.
CertifyBY	The engineer's name that certified the damage. Only for administrators, total access, and MRO engineers.

Checked	Shows if the damage is checked or not.
CheckedBY	The engineer's name that checked the damage. Only for administrators, total access, and CAMO engineers.
Closing	Closing date of the damage's inspection.
Commentary	Shows if there is a comment or not.
Comments	Shows the comments written by the users.
Description	Describes type and location of the damage.
Dim	Shows the type of dimension chosen (m, mm, cm…)
Dim1	First dimension.
Dim2	Second dimension.
Dim3	Third dimension.
Dimension	Combination of "Dim1, "Dim2", "Dim3" and "Dim" (ex: 123x30x50mm).
Entry	Entry date of the damage when found.
ID	This identification is the same as the document's name. It is used for fetching the data.
Inspection FC	Inspection flight cycle. Only for B and C categories.
Inspection FH	Inspection flight hours. Only for B and C categories.
Inspection days	Inspection days. Only for B and C categories.
Location	Damage's location.
Map	Shows if it is included in the aircraft damage's map or not.
PN	Piece number
RB	Shows if the damage has been repaired or inspected (SDT).
References	Shows the references about documentation used.
Repair FC	Repair flight cycle. Only for A category.
Repair FH	Repair flight hours. Only for A category.
Repair days	Repair days. Only for A category.
Signed	The AMT's name who signed the damaged.
SN	Piece's serial number.
Type	Type of damage (scratch, dent…).


Aircraft revision > EC-AAA_A_1172... > New damages > 12721_0040

EC-AAA_A_1172021

+ Iniciar colección

New damages

+ Agregar campo

Added: "MRO engineer"
Aircraft: "EC-AAA"
Assigned: "AMT"
CRS: "17/7/2021"
Check: "A"
Comments: "No comments"
End: "15/7/2021"
ID: "EC-AAA_A_1172021"
Start: "11/7/2021"
Status: "Planned"
Text: "No"

New damages

+ Agregar documento

11721_1301
12721_0034
12721_0040

12721_0040

+ Iniciar colección

+ Agregar campo

ATA: "51-20 Processes and Procedures"
Aircraft: "EC-AAA"
Category: "C"
Certified: "No"
CertifyBY: "-"
Checked: "No"
CheckedBY: "-"
Closing: "-"
Commentary: "No"
Comments: "No comments"
Description: "Scratch_123"
Dim: "mm"
Dim1: "32"
Dim2: "3"
Dim3: "3"
Dimension: "32x3x3mm"
Entry: ""

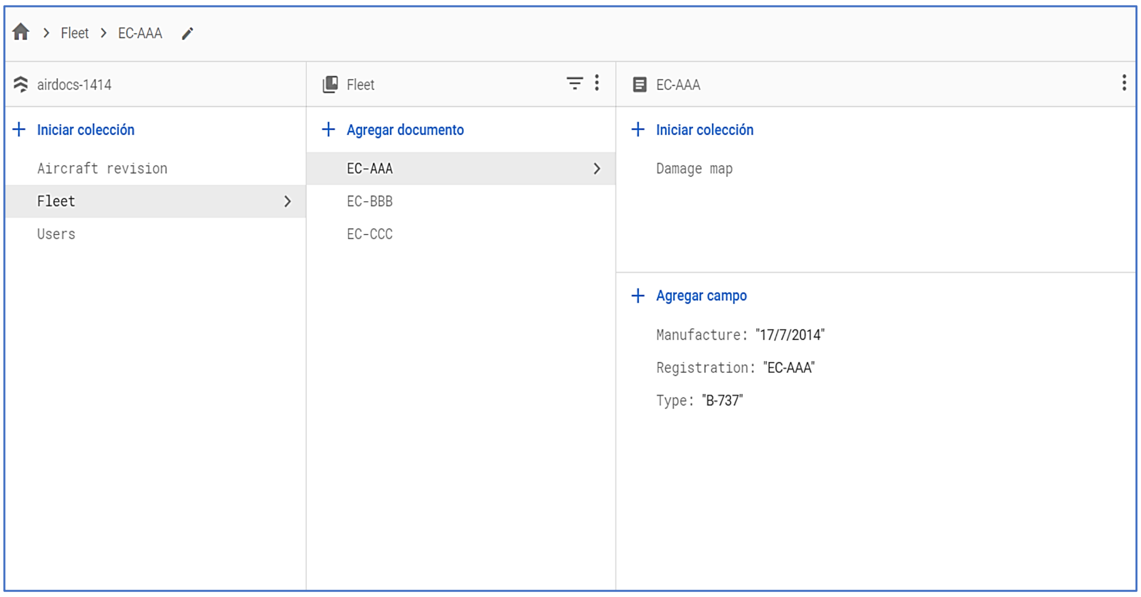
Figure 1.12 Damage list structure in Firebase

1.3.1.4. Fleet information

When a new aircraft is added to the fleet, the information is stored in “Fleet” collection and the document that contains the data is named after the aircraft registration.

Table 1.5 Aircraft information in Firebase server

Added	The engineer’s name that added the check.
Aircraft	Aircraft registration.
Assigned	AMT to which it has been assigned.
CRS	Date of entry into service.
Check	Type of check.
Comments	Comments added by the user.
End	Ending date.
ID	This identification is the same as the document’s name. It is used for fetching the data.
Start	Starting date.
Status	Check status (planned, delayed…)
Text	Indicates whether there are comments or not.



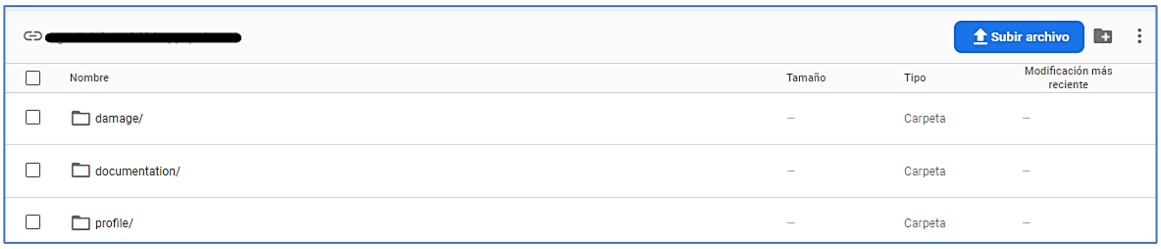
```
> Fleet > EC-AAA

airdocs-1414              Fleet                      EC-AAA
+ Iniciar colección       + Agregar documento        + Iniciar colección
Aircraft revision         EC-AAA  >                  Damage map
Fleet  >                  EC-BBB
Users                     EC-CCC
                                                     + Agregar campo
                                                     Manufacture: "17/7/2014"
                                                     Registration: "EC-AAA"
                                                     Type: "B-737"
```

Figure 1.13 Fleet list structure in Firebase

Inside the aircraft collection there is a subcollection containing the damage map which includes all the damages that the engineers decided to include. These damages have the same fields as Table 1.2.

1.3.1.5. File storage

Firebase has a section where documents and other files can be stored. For this app it is needed three folders: damages, documentation, and profile. In the “damages” folder there are pictures taken by AMTs about the damages. The second one named “documentation” stores all the documents that the users upload in the cloud and other users can check and download them. Finally, in the “profile” folder there are the user’s profile pictures to identify them.



Nombre	Tamaño	Tipo	Modificación más reciente
damage/	–	Carpeta	–
documentation/	–	Carpeta	–
profile/	–	Carpeta	–

Figure 1.14 Storage structure in Firebase

CHAPTER 2. AIRDOCS 2.0 SPECIFICATIONS

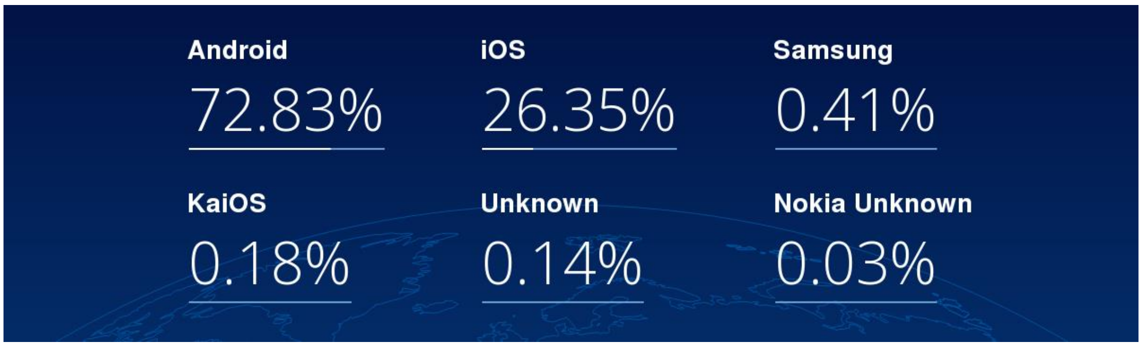
In this chapter it is explained the AirDocs 2.0 specifications and which devices can be run. Also, it is described the types of screens where AirDocs 2.0 can work and the different layouts that exists to offer the user the best experience.

2.1 Main target devices

Previously AirDocs first version was intended for both PC, Android and iOs devices. This new version is designed for Android devices only, although through a PC program the user can connect the device to their computer so they can work both on their device and on the computer. The decision not to launch AirDocs 2.0 on iOs is widely explain in the next sections.

2.1.1. Market analysis

The market share of Android and iOS operating systems occupy almost 99% of the total market. Of that 99%, 72.83% belongs to Android and 26.35% to iOS (see Figure 2.1).



Android	iOS	Samsung
72.83%	26.35%	0.41%
KaiOS	Unknown	Nokia Unknown
0.18%	0.14%	0.03%

Figure 2.1 Mobile operating system market share. Source: Statcounter.

The global market share distribution can be seen in Figure 2.2 and Figure 2.3. In North America and Asia, iOS dominates, but the percentage difference is very small, so Android is still very present in these countries. The Android and iOS market share in the United Kingdom is a 50% percent for each one while Japan has the most iOS usage operating system.

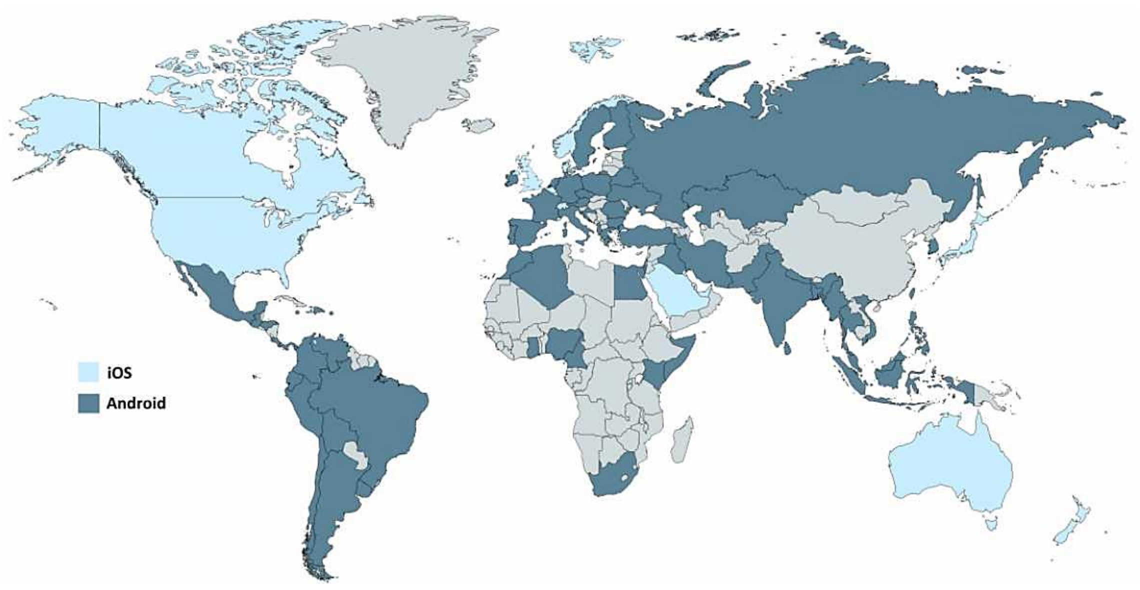


Figure 2.2 Mobile operating system market share. Source: Statcounter

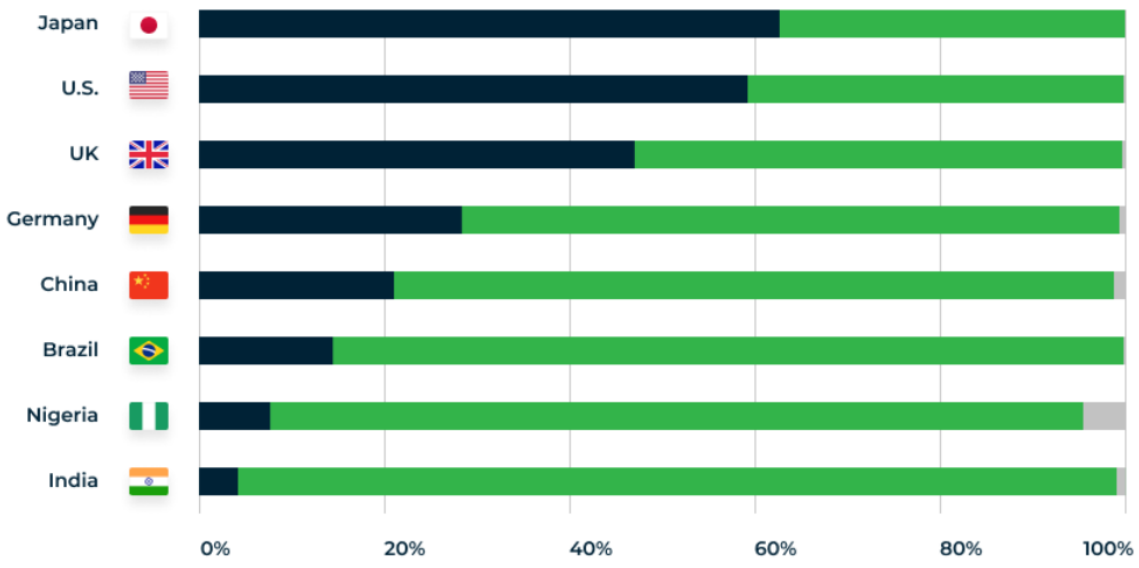


Figure 2.3 Mobile global operating system market share. Source: Statcounter

Studying the statistics (Figure 2.2 and Figure 2.3), it is implied that developing apps with Android is more profitable and the tendency is growing towards more and more use of Android in more countries where iOS is leader.

2.1.2. Open source

The open source in general refers to the source code that is programmed in such a way that it can be modified by users or developers by free and other people can even post updates so it can improve the performance and fix errors. The advantages of using Android as an open source are:

- Customizing applications in ways that other operating systems cannot allow the developer to use. For example, to implement pushing notifications can take about half a day to build in Android but in iOS can take around three days.

- Cheaper and handier to operate due to the environments that offers.

- Testing and creation of apps in the developers' devices for free using a USB cable while others operating systems cannot do it because of its closed source.

2.1.3. App distribution

App distribution is another great advantage. There are two options for the user app distribution: by publishing on Google Play or privately with other methods. AirDocs 2.0 is meant to be used in a company so there is no point in make it public in Google Play. This can be achieved by building the APK on the development environment and installing it directly into the device. Other operating systems are more restrictive on installing apps outside their respective official store. For Android is possible to publish a link to access a server in the cloud where the app is stored and give this link to the workers within a company, i.e., Google Drive. Automatically the device will detect than an app is trying to be installed and the user only must approve it. The only thing that is required is that permission must be given to the device to install app from an external source that is not Google Play. The last option is to distribute the application through the testing systems of Google Play. This kind of service is oriented for testing, but it can be used anyways to facilitate installation of the app.

2.2. Android device requirements

AirDocs 2.0 has been built for API 30 (Android 11). This is the latest version of Android launched on September 8, 2020. But Android Studio allows the app to be compatible to more versions than the last one, in this case the app can be used in devices with API level from 26 to 30. API level 26 corresponds to Android 8.0 which was launched in 2017. This decision is taken because, starting in August 2021, new apps must target API level 30 (Android 11) according to Android official website. Also, starting in November 2021, app updates will be required to target API level 30 or above and adjust for behavioural changes in Android 11 as it is mentioned in Android official Developers' website. Targeting to newer API levels bring the app new security and performance improvements and enhance user experience. Also, by doing this allows to take advantage of the platform’s latest features to use in the user’s application.

2.3. Support multiple screens

AirDocs 2.0 has been built to support a great variety of screen devices, from the biggest tablet screens to the smallest phone screens. Android Studio allows the developer to build the same layout for multiple kinds of device screens:

- 320dp: phone screen.
- 480dp: large phone screen 5".
- 600dp: 7” tablet.
- 720dp: 10” tablet.

With this feature, AirDocs 2.0 assures that it will display its layouts correctly without visual errors on all devices.

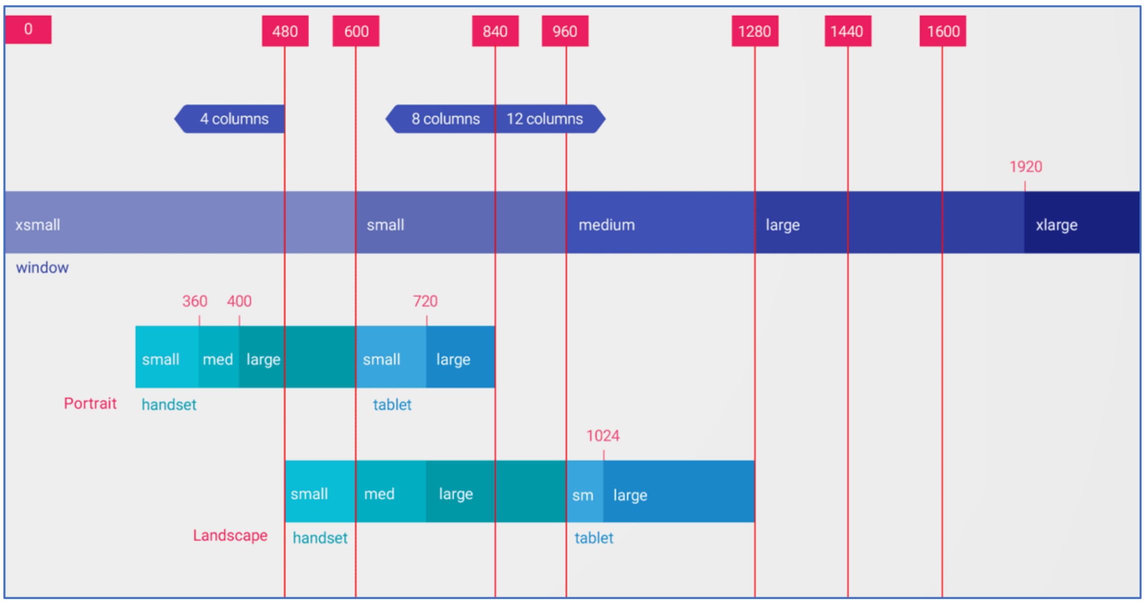


Figure 2.4 Different screen sizes. Source: Android Developers.

2.4. Layout orientation

Another feature of AirDocs 2.0 is to allow portrait and landscape orientation. It allows the users to choose the best display for them. Figure 2.5 and Figure 2.6 shows how the users sees the same layout while being in portrait and landscape.

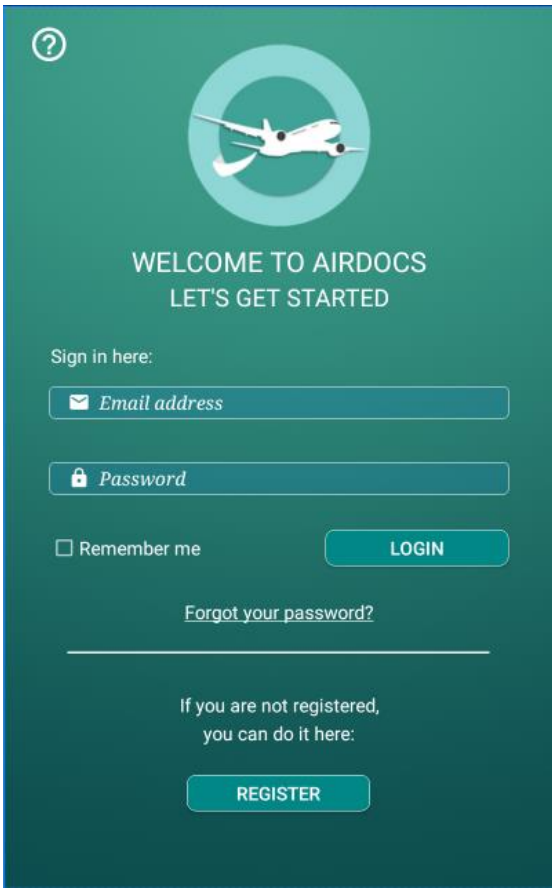


Figure 2.5 Portrait mode

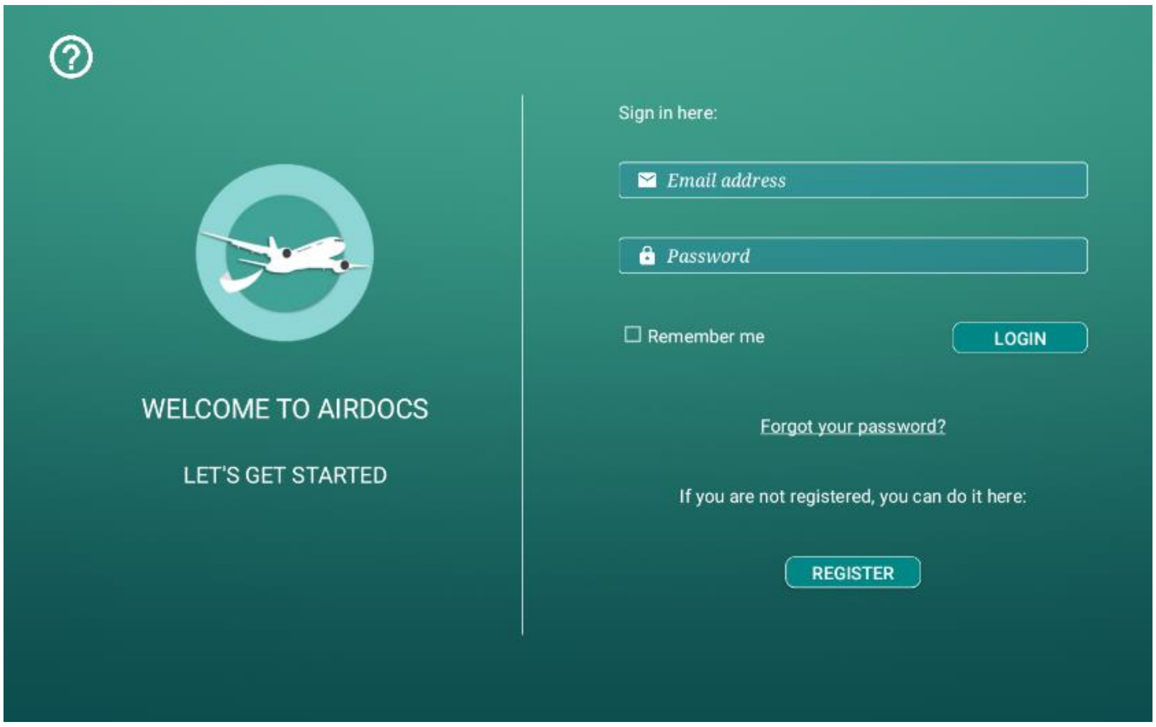


Figure 2.6 Landscape mode

CHAPTER 3. AIRDOCS 2.0: HOW IT WORKS

In this section, it explains all the new features and improvements introduced and a detailed description of the app’s layouts. In terms of typing the information, because of the changes applied, there are new forms to introduce the damages, checks and other data so there is also an explanation on how to do it.

3.1. Splash screen

When launched, the first screen the users see is the splash screen. It shows the logo of AirDocs 2.0, and the database used. The main objective of this is to give the app time to load all its functions so it will load faster later.

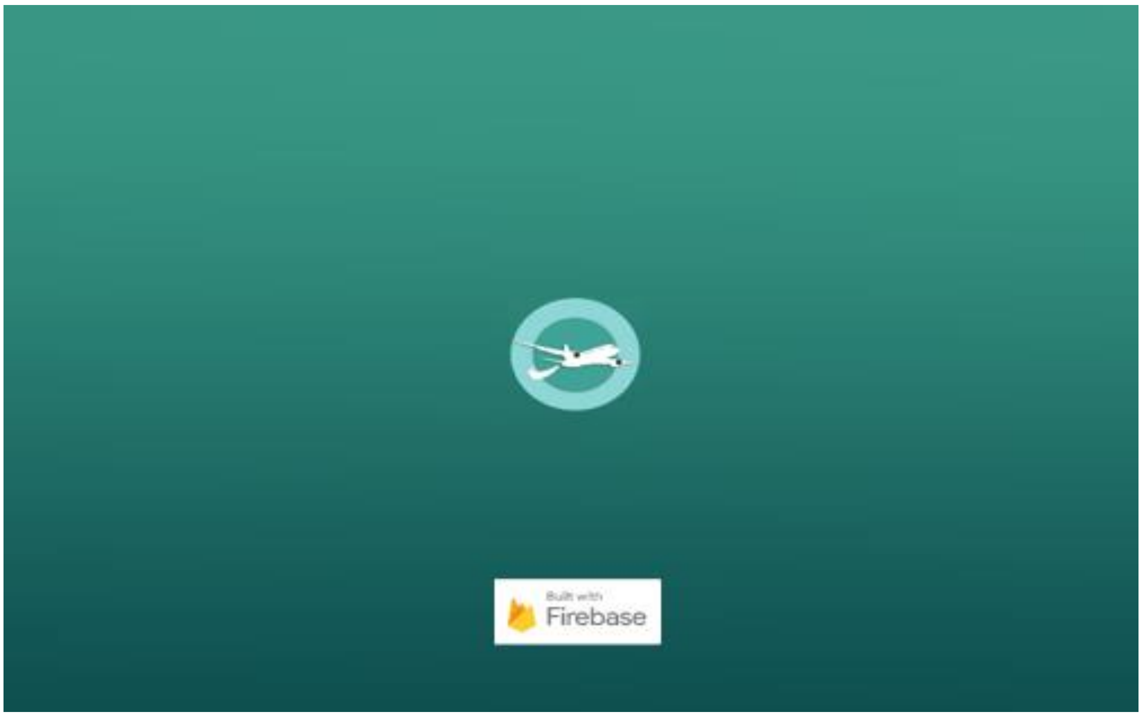


Figure 3.1 Splash screen

3.2. Welcome screen

Right after the splash screen has finished the welcome screen is shown. Here the user must type their email address and password to log into AirDocs 2.0. There are multiple warning errors when the users want to enter when data is incorrect; if the email and password are wrong, the email does not exist, the administrator has not given access to the user, or if the email has not been verified, a message will trigger in all four cases. A “Remember me” check button is now available so when exiting the app and returning to it the credentials are stored and the user won’t have to type them again, making it faster to log in.

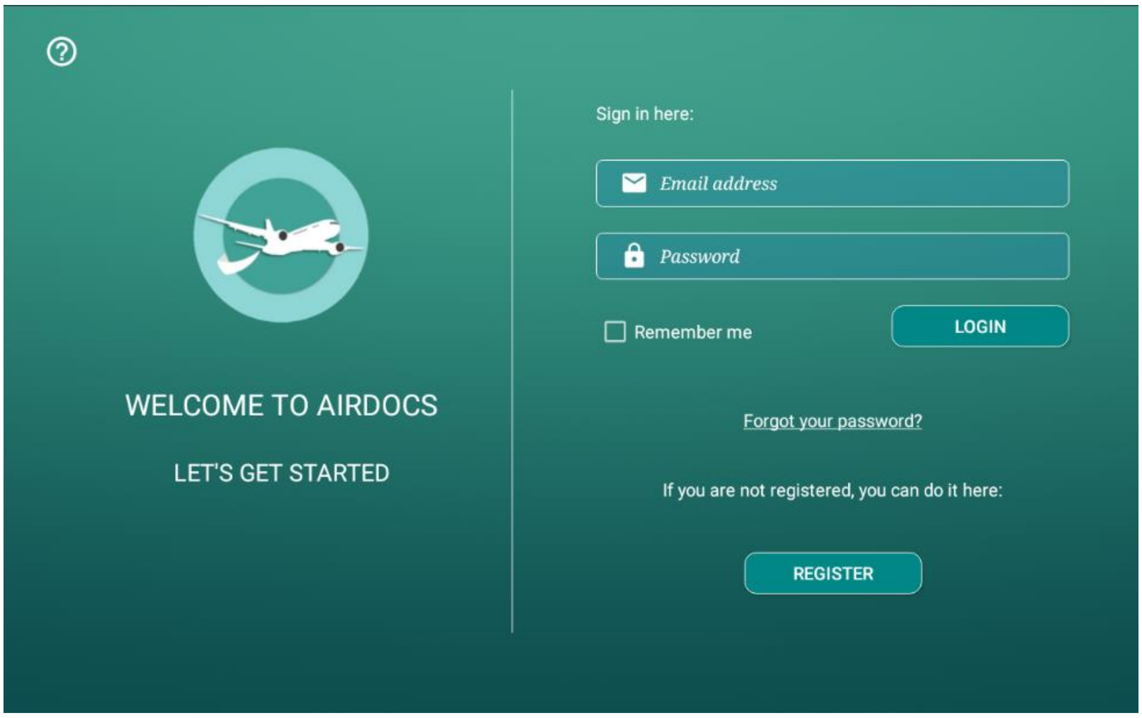


Figure 3.2. Welcome screen

If the user forgets their password, it is possible to recover it by clicking “Forgot your password?” and it will send to user’s email a link to reset the password. At the bottom, new users can click on “Register” and fill a form to use AirDocs 2.0.

At the left upper corner of the screen there is an information button. When clicking on it a message will display showing an email address to contact the administrator in case of doubts or other questions the user wants to ask.

```
Hello,

Follow this link to reset your AirDocs password
for your airdocs.tests@gmail.com account.

https://airdocs-1414.firebaseapp.com/...

If you didn’t ask to reset your password, you can ignore this email.

Thanks,

Your AirDocs team
```

Figure 3.3 Reset password email template

Firebase has a high security method when refers to passwords so when entering the new password, it must be at least 6 or more characters and a mix of letters and numbers. This condition is also included in the app registration.

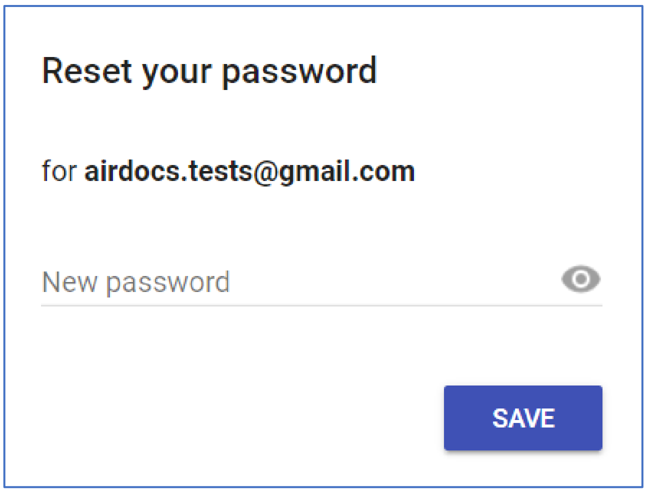


Fig 3.4. Changing the password when resetting password

3.3. Register screen

If new users want to access the app, they must register typing their personal data. All the necessary data to introduce is shown at Figure 3.5 and even they can upload a profile photo helping to identify themselves. The data users must fill is explained at Table 3.1.

Table 3.1 User registration data

Email	User's email.
Job position	Describes the user's job: administrator, CAMO engineer, MRO engineer, total access engineer, AMT, and cabin crew.
Name	User's full name.
PIN	PIN created by the user to confirm actions inside the app.
Phone	Contact phone number.
Password	User's password to access the app
Confirm password	Field to check if password was written correctly.

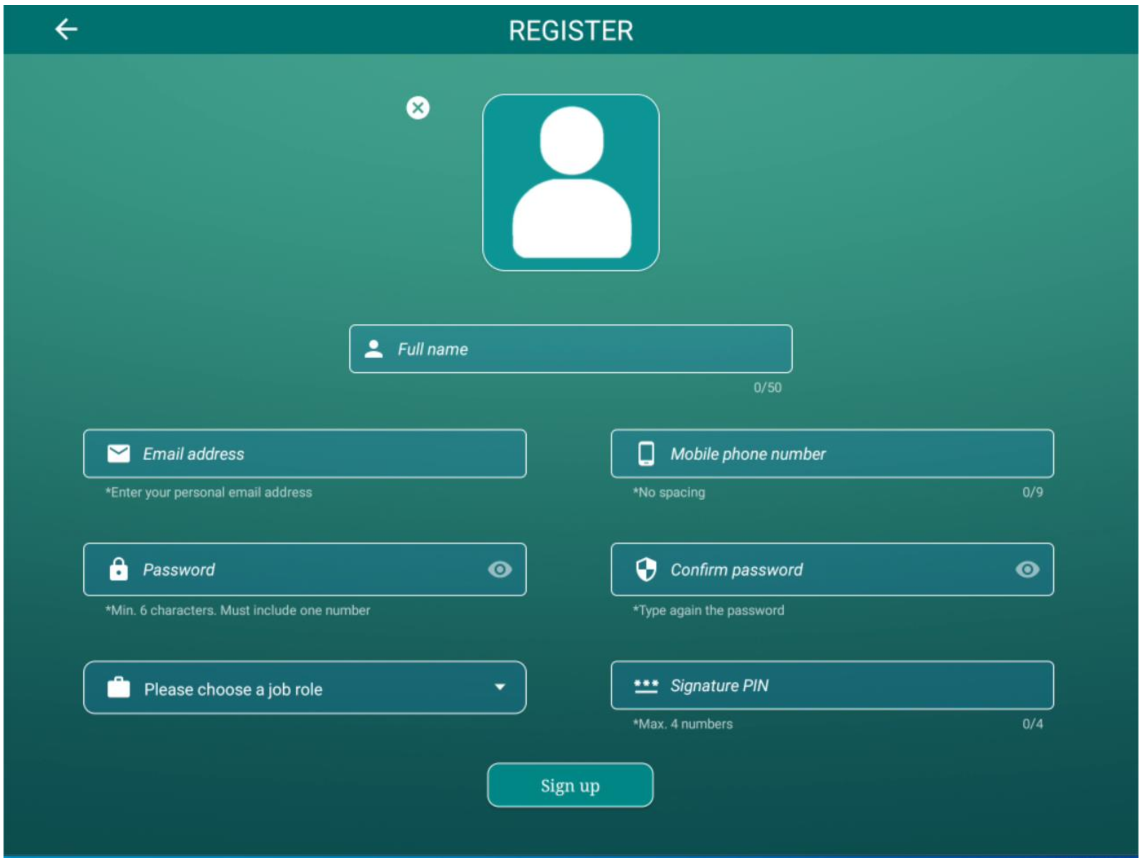


Figure 3.5 Register screen

In addition, error warnings have been included; if there are empty fields, if the password field is empty and so on. When registered a verification mail is sent to their emails. If users are not verified, they cannot access to the app, and even if they are verified the administrator must give access to new users after checking their information. This is done to give high security to the app.

```
Hello,

Follow this link to verify your email address.

https://airdocs-1414.firebaseapp.com/...

If you didn’t ask to verify this address, you can ignore this email.

Thanks,

Your AirDocs team
```

Figure 3.6 Verification email template

3.4. Home screen

Once the administration has given permission to new users, they can log into the app. Depending on the job role the user has, they will be redirected to their corresponding home menu.

- **Administrator:** When the user has an administrative role, they can access to all modules within the app. The Engineer/AMT module has all the necessary tools for the administrator to make the same task as the others, but it had not been split because this way code is shortened and faster. Additionally, they have two unique modules available to them: the user module and the aircraft fleet module.

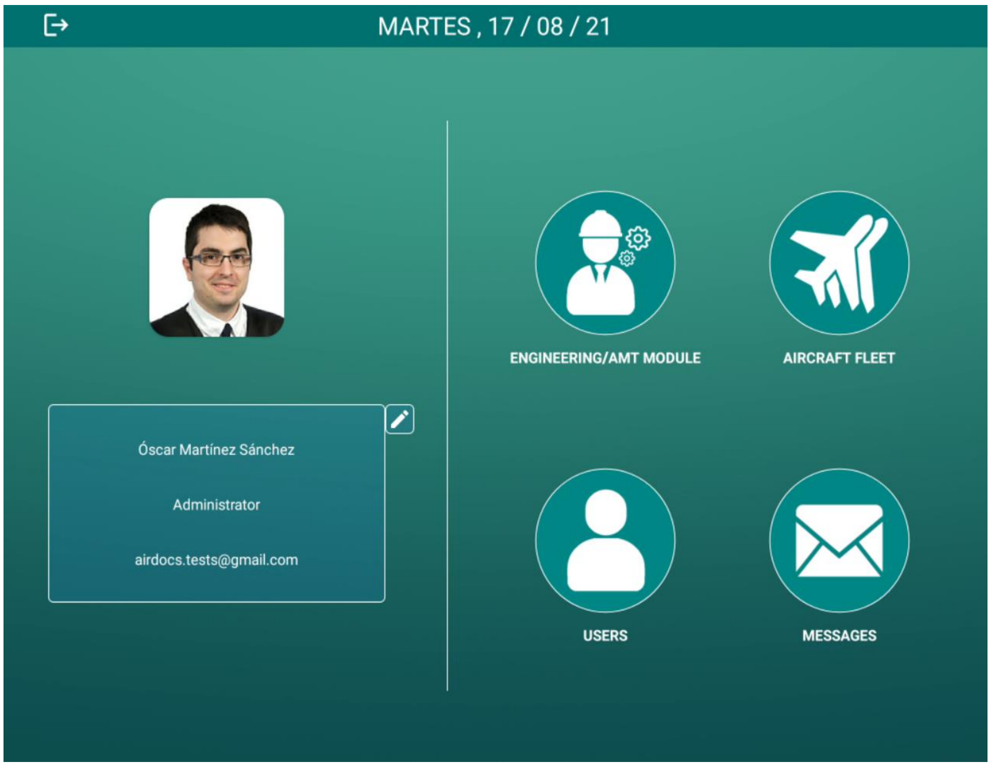
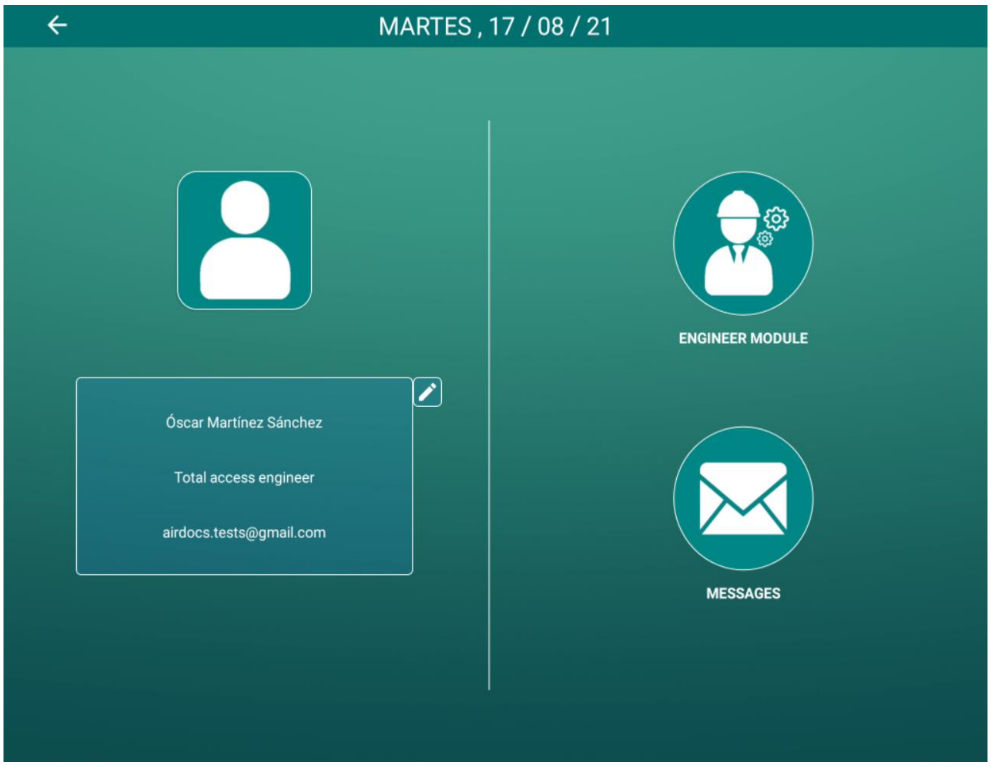


Figure 3.7 Administrator home screen

- **Engineer:** When the user has a total access engineer, CAMO or MRO engineer role, they can access only to their respective module where they can manage the aircraft check list and the actions, they can carry out depends on the user role.



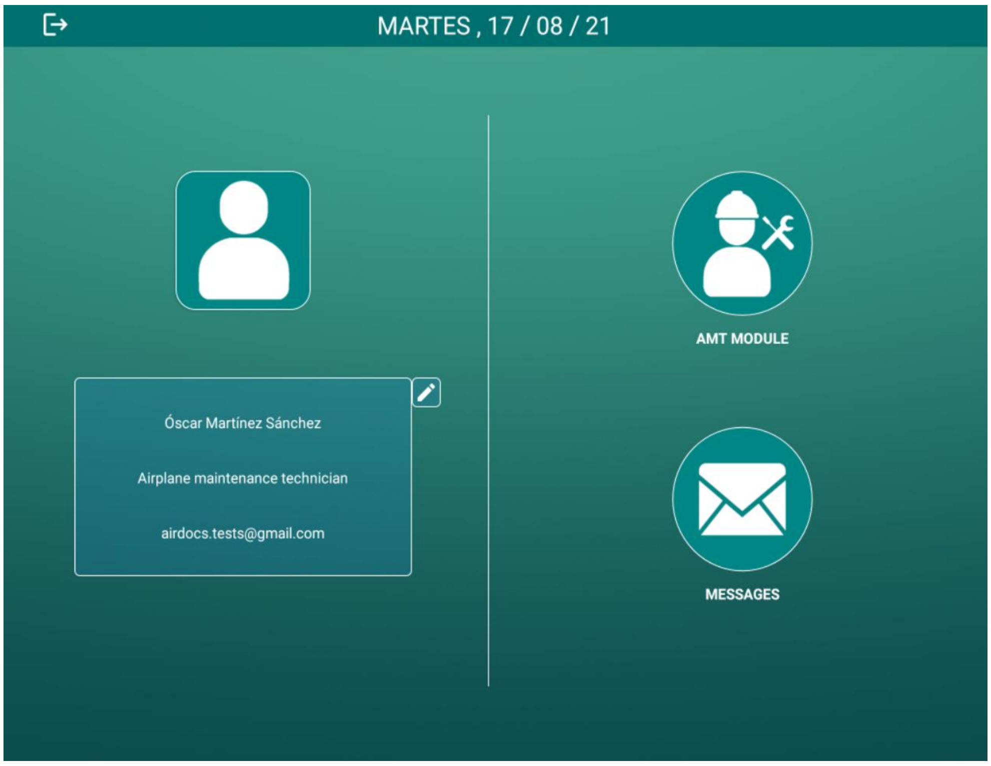
Óscar Martínez Sánchez

Total access engineer

airdocs.tests@gmail.com

Figure 3.8 Engineer home screen

- **AMT:** When the user has an AMT role, they can access only to their AMT module where they can add and manage aircraft's damages.



Óscar Martínez Sánchez

Airplane maintenance technician

airdocs.tests@gmail.com

Figure 3.9 AMT home screen

- **Cabin crew:** When the user has a cabin crew role, they can access only to cabin crew module where they can see and report aircraft's damages.

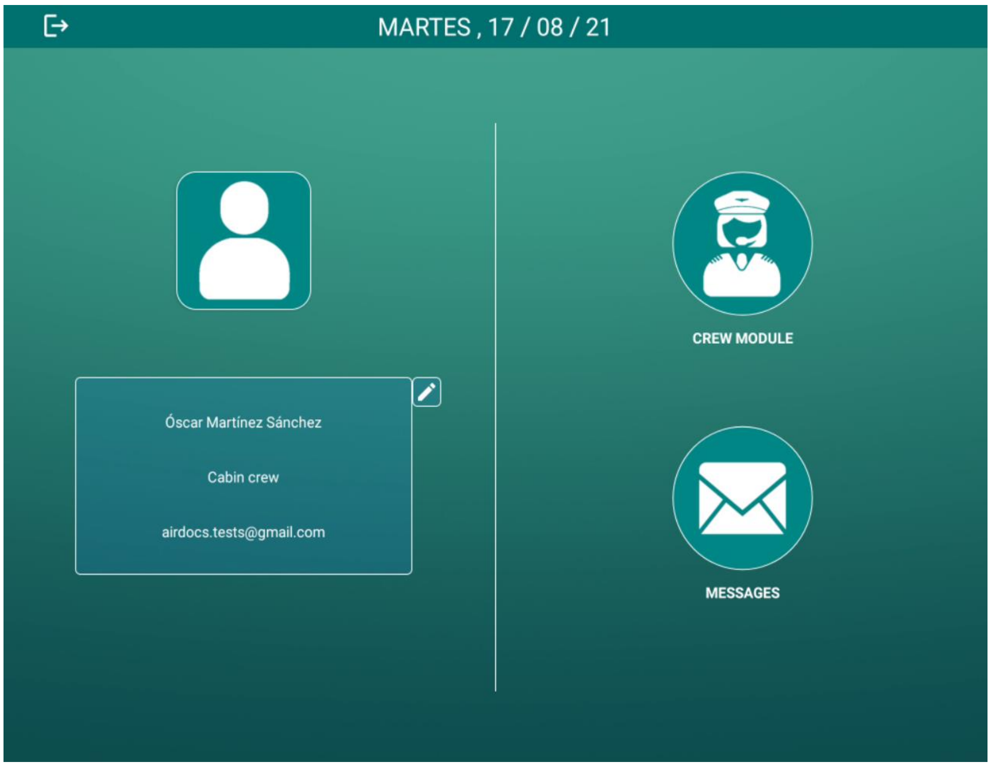
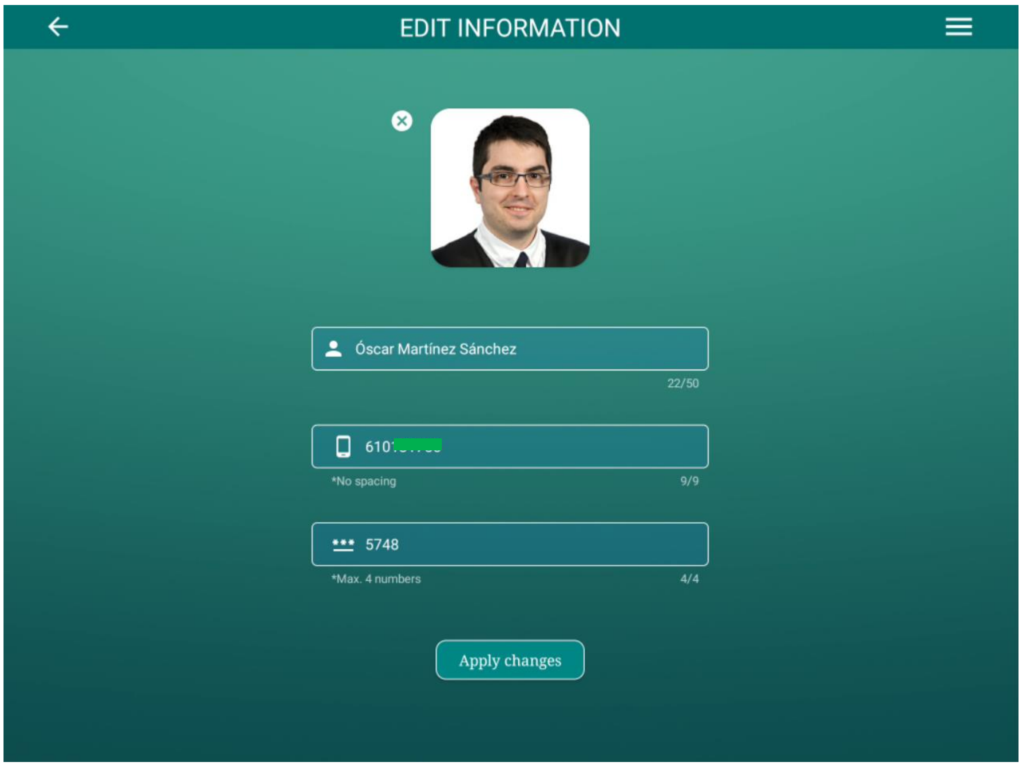


Figure 3.10 Cabin crew home screen

All users have access to the chat module where they can send emails to users within the app. This feature is explained in its corresponding section.

The other option in common is that user information can be changed. When they click on pencil shaped button located at the right-hand corner of the user information they will be redirected to another screen where their information can be edited (see Figure 3.11). If the user wants to change their password or email, they must click the menu located at the top right-hand corner of the screen where a menu will pop up with these two features.



Óscar Martínez Sánchez
22/50

610[redacted]
*No spacing 9/9

5748
*Max. 4 numbers 4/4

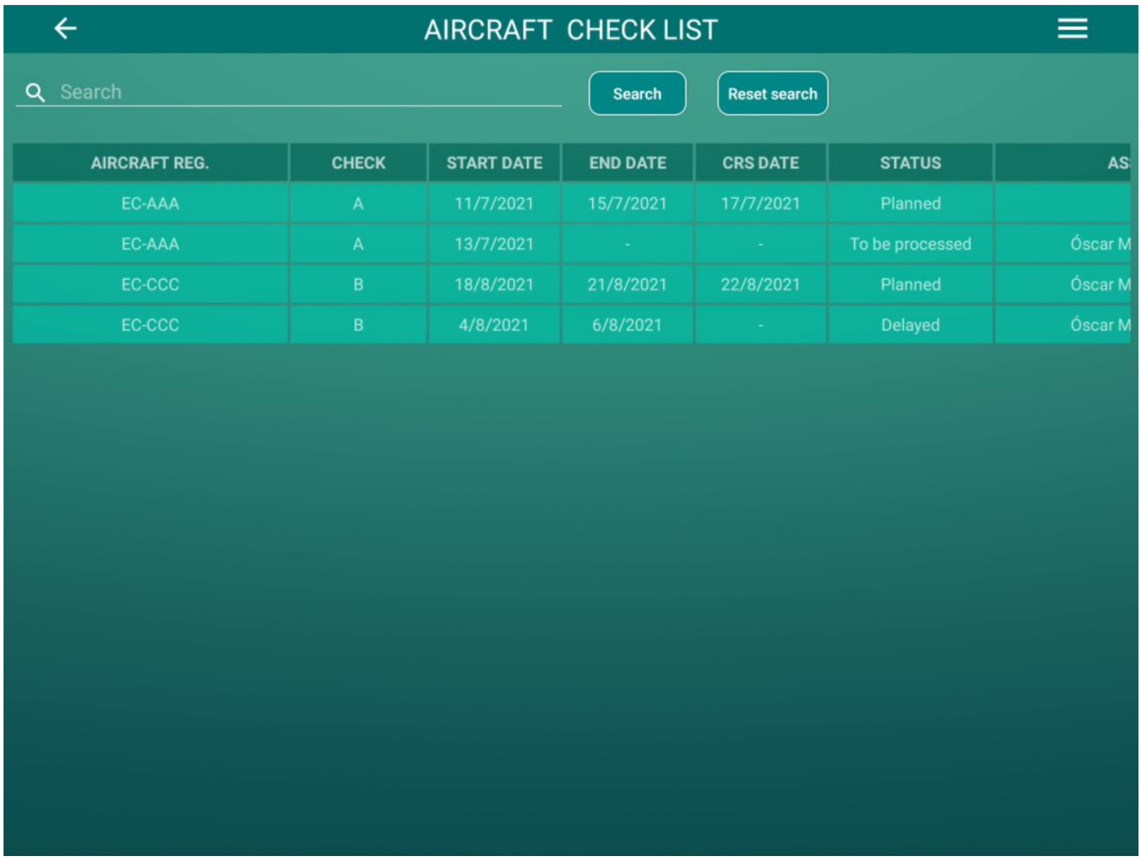
Figure 3.11 Edit user information

3.5. Engineer module

In this module the users with an engineering or administrator role can add an aircraft check. In Figure 3.13, the complete table is not shown totally because it can be scrolled horizontally and vertically when it is out of bounds.

Table 3.2 Aircraft check fields at check list screen

Aircraft reg.	Aircraft registration.
Check	Type of check.
Start date	Beginning date of the aircraft check
End date	Ending date of the aircraft check
CRS date	Date of entry into service.
Status	Check status (planned, delayed…)
Assigned to	AMT to which it has been assigned.
Comments	Indicates whether there are comments or not.
Added by	The engineer’s name that added the check.



AIRCRAFT REG.	CHECK	START DATE	END DATE	CRS DATE	STATUS	AS
EC-AAA	A	11/7/2021	15/7/2021	17/7/2021	Planned	
EC-AAA	A	13/7/2021	-	-	To be processed	Óscar M
EC-CCC	B	18/8/2021	21/8/2021	22/8/2021	Planned	Óscar M
EC-CCC	B	4/8/2021	6/8/2021	-	Delayed	Óscar M

Figure 3.12 Aircraft check list

If the user wants to add a check, they must click on the menu located at the top right-hand corner of the screen. The user is redirected to a screen where aircraft checks can be added. The information ask by the app is the same as shown in Table 3.3.

Table 3.3 Aircraft check fields at add check screen

Aircraft reg.	Aircraft registration.
Check	Type of check.
Start date	Beginning date of the aircraft check
End date	Ending date of the aircraft check
CRS date	Date of entry into service.
Status	Check status (planned, delayed…)
Assigned to	AMT to which it has been assigned.
Comments	Place where users can write comments.
Added by	The engineer’s name that added the check.

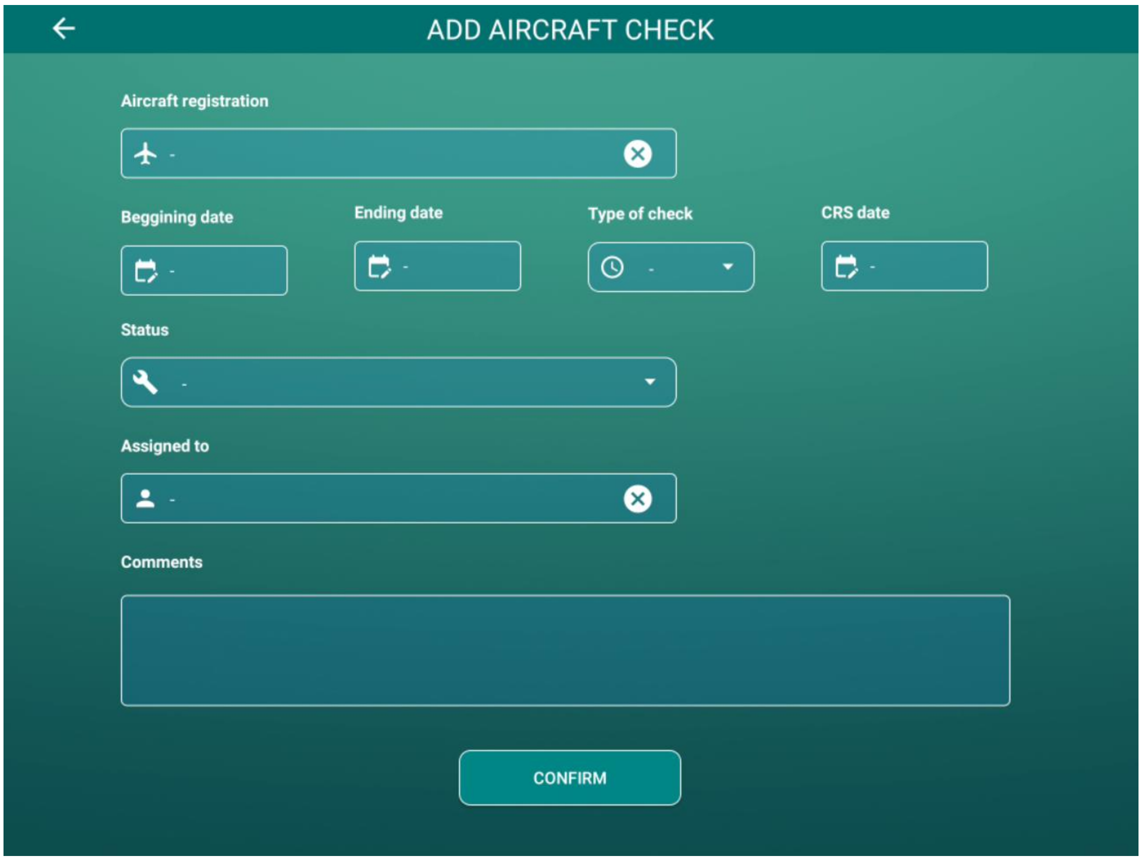


Figure 3.13 Add aircraft check screen

When the user clicks on a specific row from the list of aircraft check at Figure 3.12, a menu will pop up (Figure 3.14). The menu layout shown depends on the user role, but basically the only difference between engineers and administrators is a “Delete” function. Only administrators have the permission to delete checks while the rest of engineers can’t. This menu will show which aircraft registration has been clicked and the following submenus:

- **Edit check:** Here the users can change or add information of an aircraft check. This screen shares the same layout as Figure 3.13.

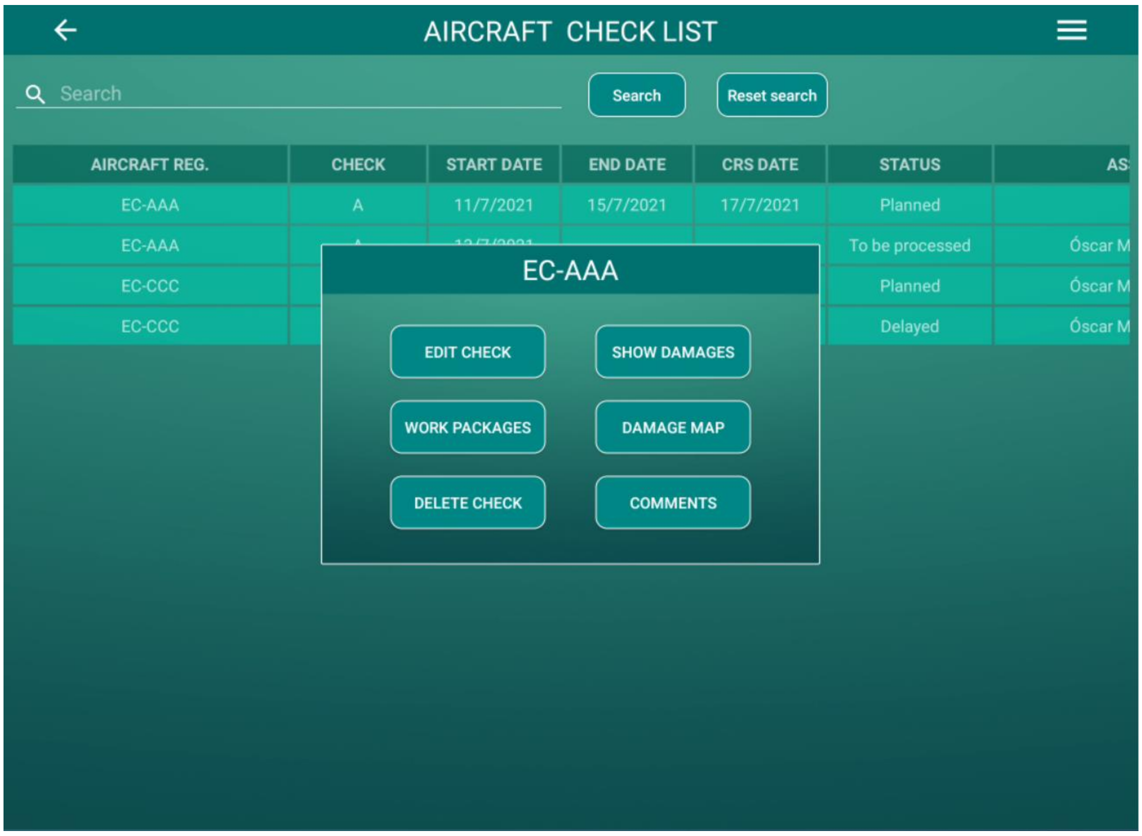
- **Show damages:** When clicked, the user is redirected to the damage list where there are all the aircraft damages reported by the AMT. Again, the actions depend on the role: the CAMO engineer can check the damage while the MRO engineer can certify them. Both administrator and total access engineer have the permission to check and certify. All the engineers and administrators can decide to include or remove the damage from the damage map from here too.

- **Work packages:** Here the engineers can see and add tasks to be carried out by the AMTs.

- **Damage map:** When the user access here they will see all the aircrafts belonging to the company. Clicking on a specific aircraft will show all the damages included by engineers.

- **Delete check:** This button is visible only for administrators. This removes the check from the list and is no longer available for the other users.

- **Comments:** Shows a pop-up screen where there are all the comments added by the users previously. The users can also write, modify or delete comments.

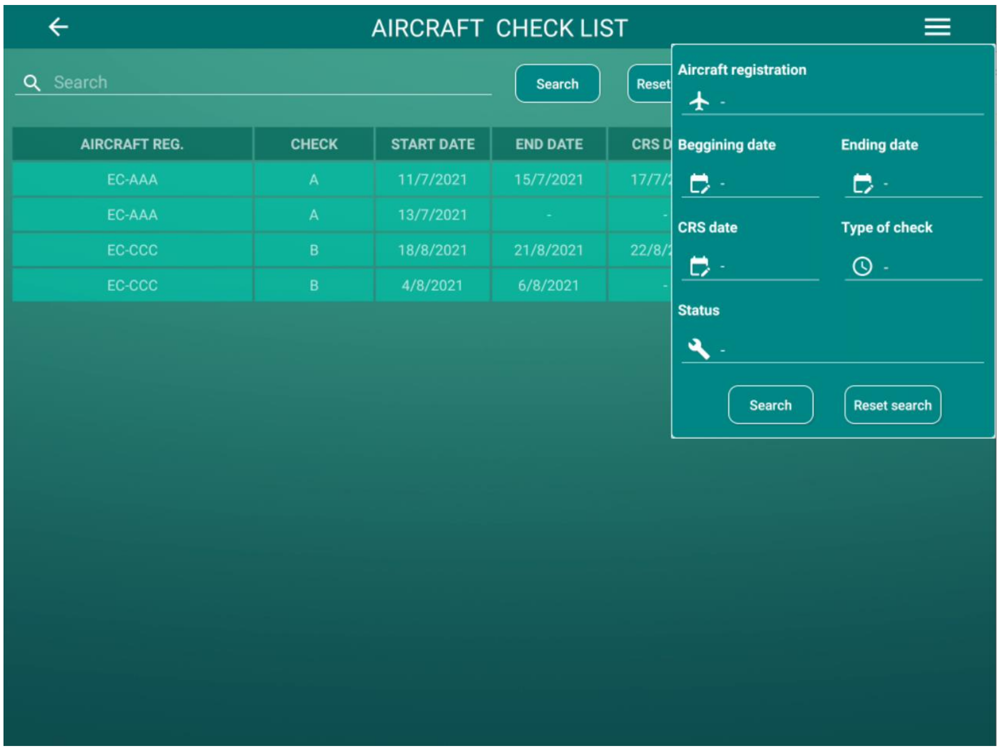


AIRCRAFT REG.	CHECK	START DATE	END DATE	CRS DATE	STATUS	AS
EC-AAA	A	11/7/2021	15/7/2021	17/7/2021	Planned	
EC-AAA	A	12/7/2021			To be processed	Óscar M
EC-CCC					Planned	Óscar M
EC-CCC					Delayed	Óscar M

Figure 3.14 Aircraft check menu

To facilitate the work of engineers, changes have been made to add information more quickly. Clicking on the aircraft registration field, users will be redirected to the fleet screen where an aircraft can be selected, then the aircraft registration selected will automatically be added saving time in searching and typing. This feature can be seen also to assign checks to users, but instead of redirecting the users to the fleet screen now they will be redirected to the user list. One or more persons can be selected. The type of check and status fields have in common is that when they are clicked, a list with information is displayed, so users only need to click on the appropriate one to add it. Regarding beginning data, ending date, and CRS date fields, clicking on them will bring up a calendar from which the user can select the appropriate date. It is worth noting that not all fields are required to add a check; both aircraft registration and beginning date are necessary while other fields can be empty and completed later.

A search bar is available to find a specific aircraft, but if users want a wider search, they can search with conditions looking for the desired information (see Figure 3.15). Within this advanced search option, you can search for an aircraft with a specific beginning date, for example.



AIRCRAFT REG.	CHECK	START DATE	END DATE	CRS D
EC-AAA	A	11/7/2021	15/7/2021	17/7/
EC-AAA	A	13/7/2021	-	-
EC-CCC	B	18/8/2021	21/8/2021	22/8/
EC-CCC	B	4/8/2021	6/8/2021	-

Aircraft registration: -

Beggining date: -

Ending date: -

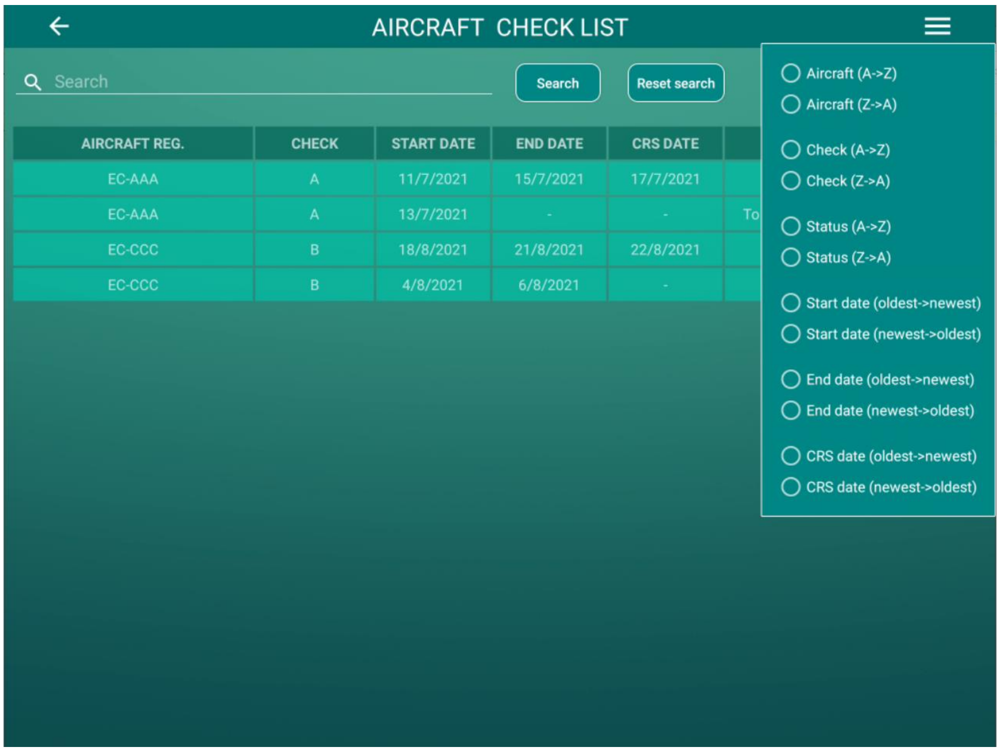
CRS date: -

Type of check: -

Status: -

Figure 3.15 Advanced search at aircraft check list screen

It is also possible to sort the table alphabetically, by date and so on (see Figure 3.16).


AIRCRAFT CHECK LIST
Search
Search
Reset search
Aircraft (A->Z)
Aircraft (Z->A)
Check (A->Z)
Check (Z->A)
Status (A->Z)
Status (Z->A)
Start date (oldest->newest)
Start date (newest->oldest)
End date (oldest->newest)
End date (newest->oldest)
CRS date (oldest->newest)
CRS date (newest->oldest)

AIRCRAFT REG.	CHECK	START DATE	END DATE	CRS DATE	
EC-AAA	A	11/7/2021	15/7/2021	17/7/2021	
EC-AAA	A	13/7/2021	-	-	To
EC-CCC	B	18/8/2021	21/8/2021	22/8/2021	
EC-CCC	B	4/8/2021	6/8/2021	-	

Figure 3.16 Sort menu at aircraft check list screen

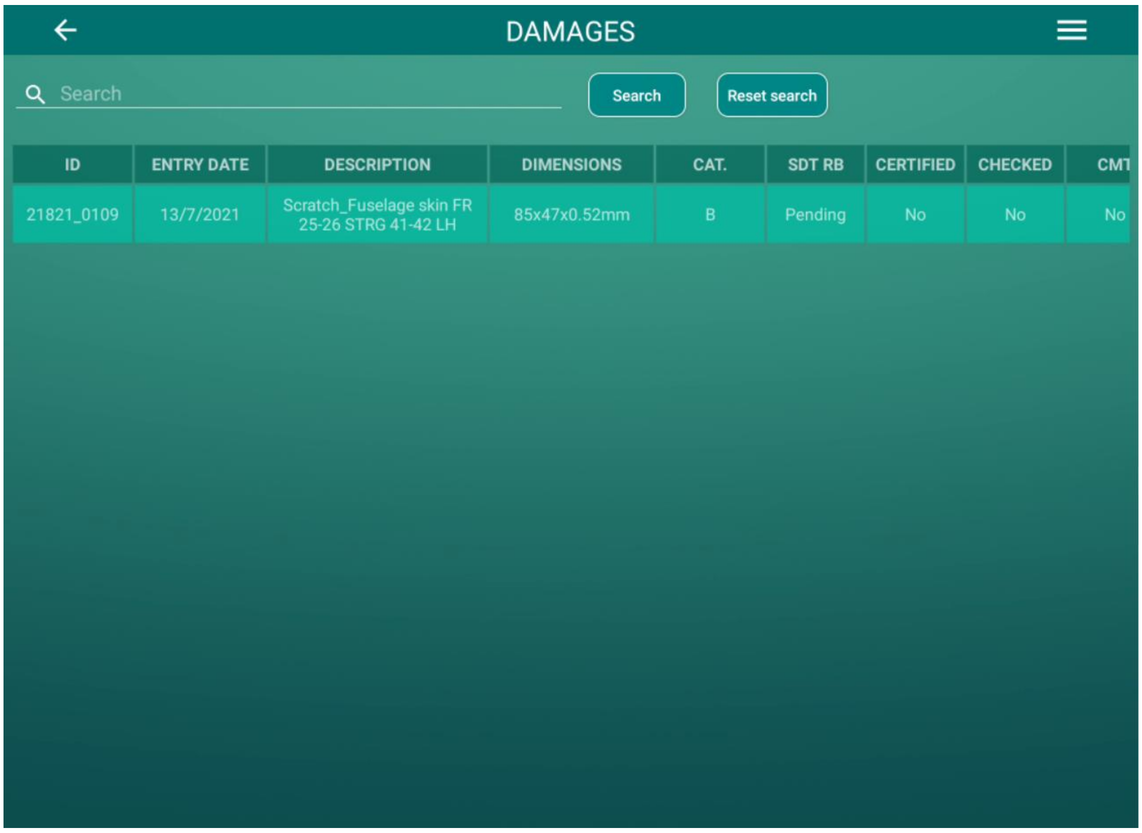
3.6. Airplane maintenance technician module

In this module the users with an AMT or administrator role can add and manage aircraft damages. On the other hand, engineers can only visualize damages and check or certify them. This module consists of a table list with damage information that AMTs find throughout the revisions. The data shown is explained at Table 3.4. In Figure 3.17, the damage table list can be scrolled horizontally and vertically when it is out of bounds.

Table 3.4 Aircraft damage fields at damage list screen

ID	Damage ID.
Entry date	Date on which the damage is found.
Description	Description about the location of the damage.
Dimensions	Damage dimensions.
CAT.	Ending date of the aircraft check
CRS date	Damage category (A, B or C).
SDT RB	Indicates if a SDT RB has been done.
Certified	Shows a ‘Yes’ or ‘No’ if the damage is certified or not.

Checked	Shows a ‘Yes’ or ‘No’ if the damage is checked or not.
CMT	Shows a ‘Yes’ or ‘No’ if there are comments or not.



ID	ENTRY DATE	DESCRIPTION	DIMENSIONS	CAT.	SDT RB	CERTIFIED	CHECKED	CMT
21821_0109	13/7/2021	Scratch_Fuselage skin FR 25-26 STRG 41-42 LH	85x47x0.52mm	B	Pending	No	No	No

Figure 3.17 Aircraft damage list

When the user clicks on a specific row, it will pop up a menu (Figure 3.18) and the layout displayed depends on the user role. The only difference between AMTs and administrators is that administrators can delete damages while AMTs can’t. The menu shown contains the damage ID and the following submenus:

- **SDT RB:** When clicked, only AMTs and administrators are redirected to “Report back” screen where data of a damage is displayed and can be edited. This option is only available for damage with categories B and C. Once the AMTs considered the damage is totally repaired they must submit the changes. When this is done, the field named “SDT RB” at Figure 3.17 changes to “Completed”. If engineers enter from their respective modules, the “SDT RB” option is not available. If they want to certify or check, they must go to “Show details” option.

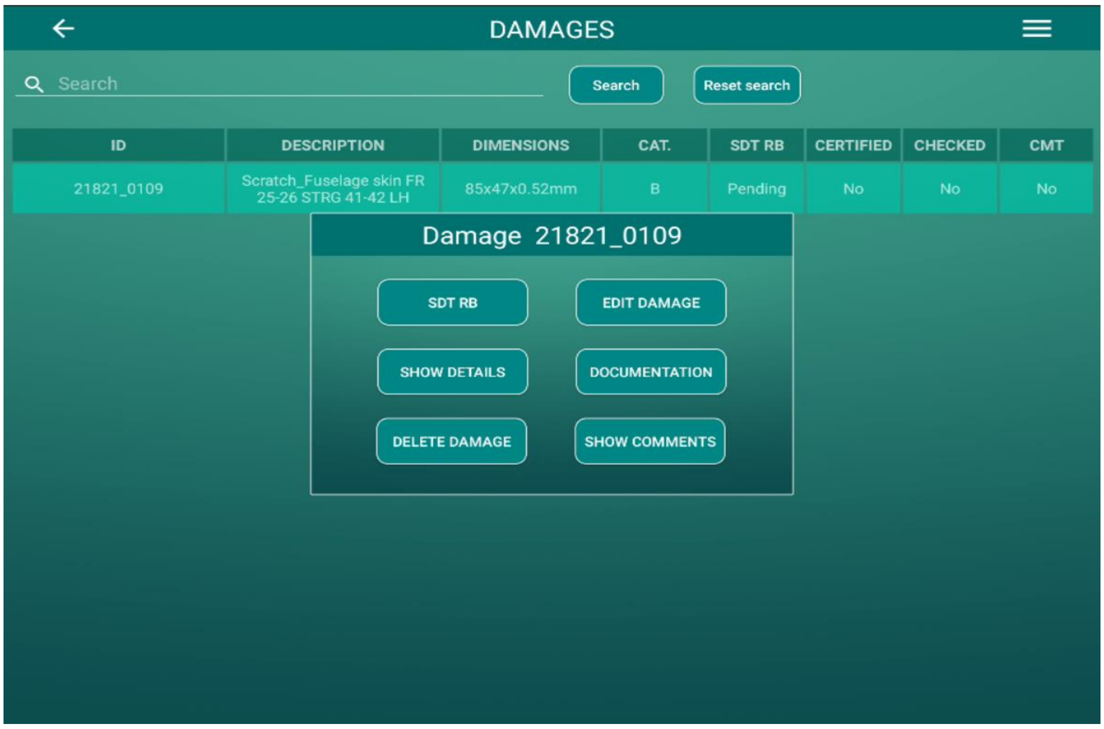
- **Edit damage:** Here the AMTs and administrators edits the damage data if they have typed wrongly some information when adding it. Instead of deleting the whole damage and rewrite it, they can change the fields that were wrongly written.

- **Show details:** Not all the damage data is shown at Figure 3.17. That is because if all information is displayed the user may be overtaken. So, for this reason there is a "Shown details" feature. When clicking on it, users will be redirected to a screen where all the damage information is shown. Here the AMTs can see the names of the engineers who have checked or certified the damaged. Additionally, when engineers and administrators access here from their respective modules, they can certify, check, include to map or delete to map damages. It should be noted that AMTs do not have these functions. Finally, administrator and total access engineers can certify and check while MRO and CAMO engineers can do one of them (depending on the externalisation).

- **Documentation:** Here the users can upload, visualize, and download the documentation related to the damage.

- **Delete check:** This button is visible only for administrators. This removes the damage from the list and is no longer available for the other users.

- **Comments:** Shows a pop-up screen where there are all the comments added by the users previously. The users can also write, modify, or delete comments.



ID	DESCRIPTION	DIMENSIONS	CAT.	SDT RB	CERTIFIED	CHECKED	CMT
21821_0109	Scratch_Fuselage skin FR 25-26 STRG 41-42 LH	85x47x0.52mm	B	Pending	No	No	No

Figure 3.18 Aircraft damage menu

If AMTs want to add damages, they must go to the menu located at the top right hand of the screen (see Figure 3.17) and select the "Add damage" option. Then they will be redirected to a screen where they can add the desired information and like other layouts the screen can be scrolled down and up to visualize all the elements (see Figure 3.19). As it was mentioned at section 3.6, changes have been made to add information more quickly. Now the users will not have to write all the data, and as a result users' performance is improved so time spend on writing is less. The most important feature added is the capacity to upload or take a photo of the damage so there is a visual reference which can lead to a successfully reparation.

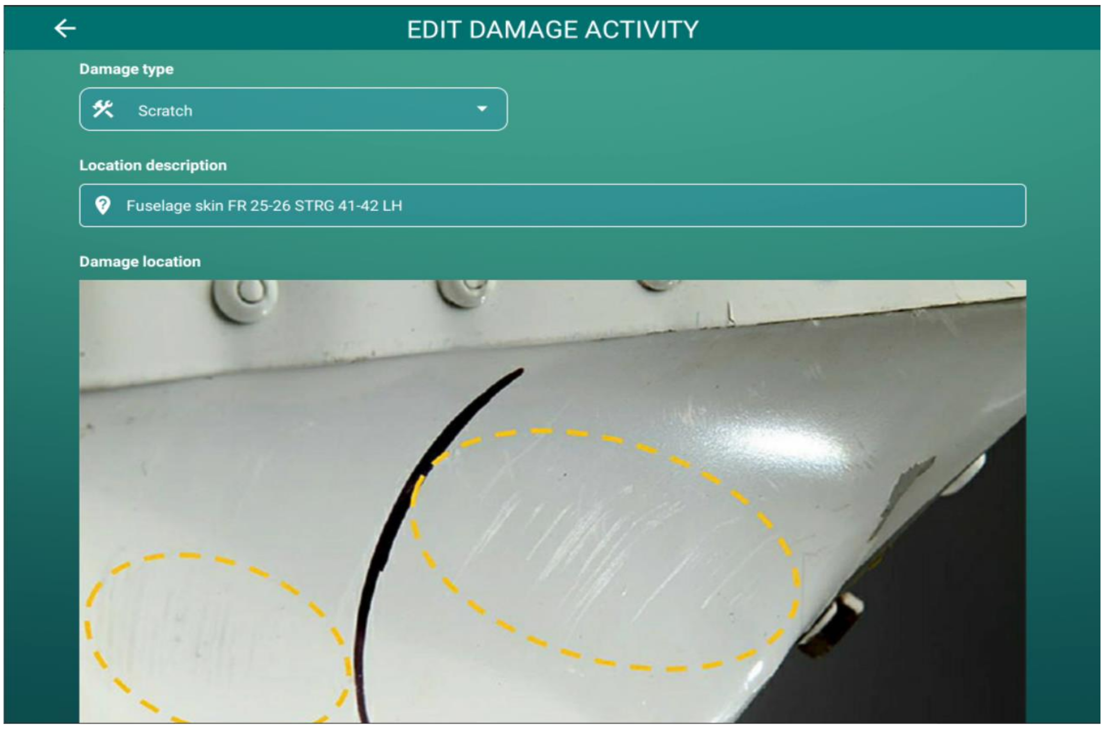
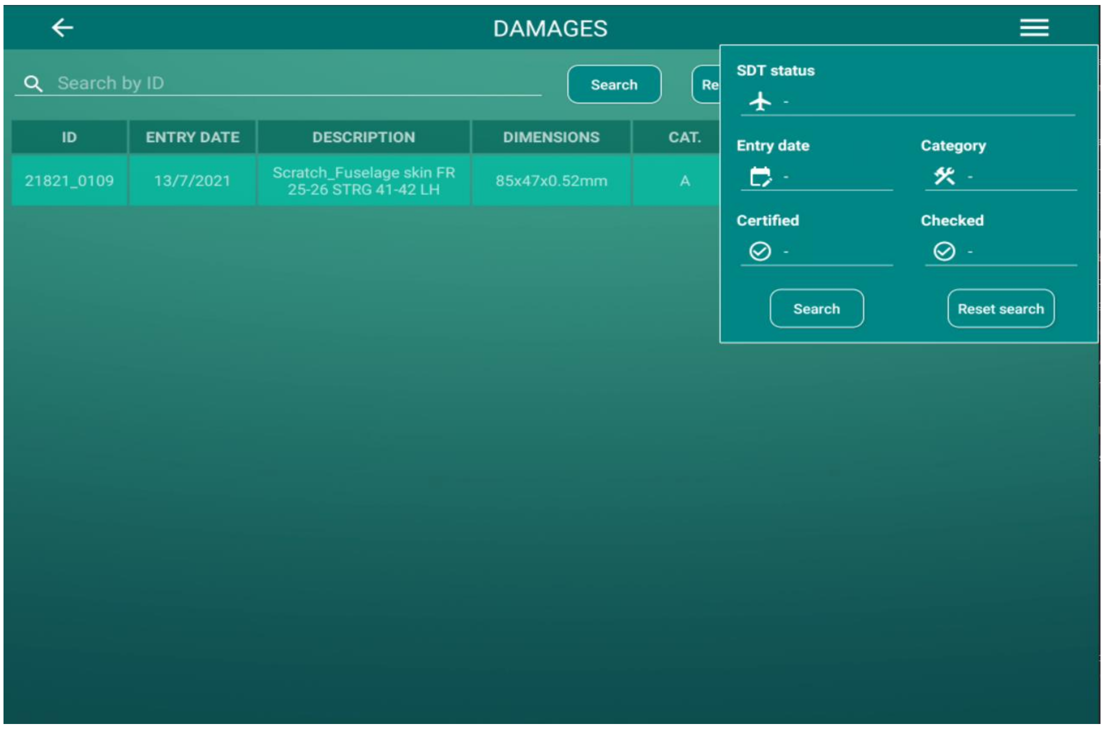


Figure 3.19 Adding damages

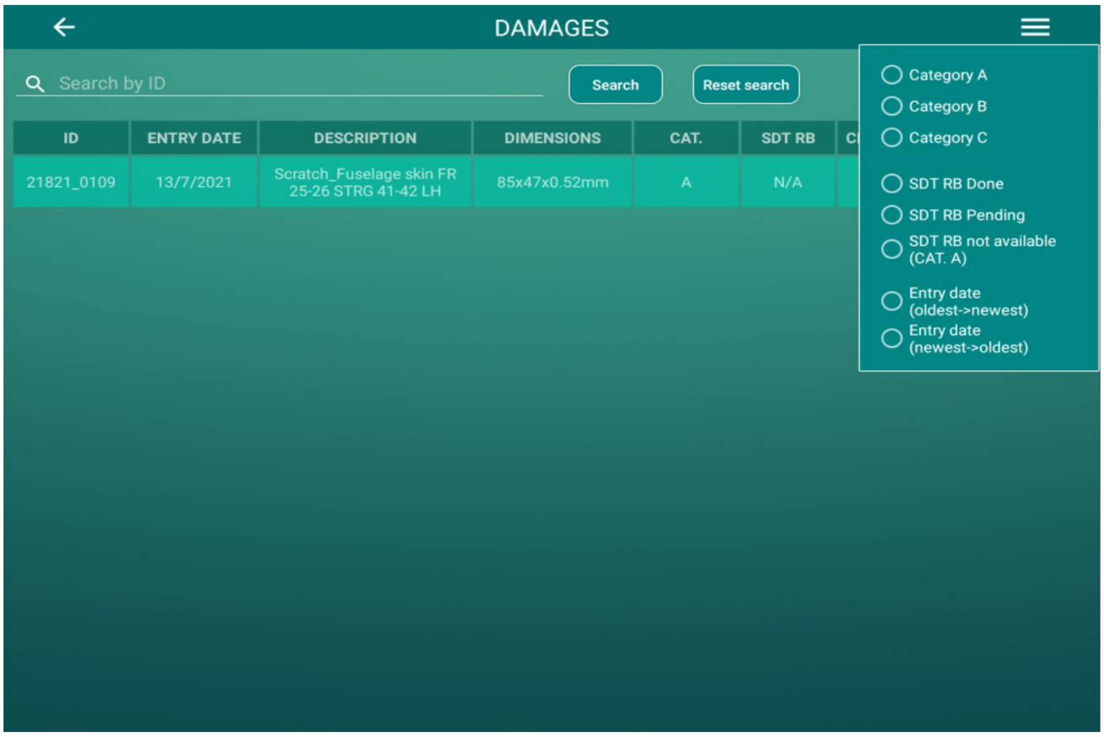
A search bar is available to find a specific damage, but if users want a wider search, they can look for any damage applying conditions such as the entry date, category and so on (see Figure 3.20).



ID	ENTRY DATE	DESCRIPTION	DIMENSIONS	CAT.
21821_0109	13/7/2021	Scratch_Fuselage skin FR 25-26 STRG 41-42 LH	85x47x0.52mm	A

Figure 3.20 Advanced search at damages screen

It is also possible to sort the table by certified, checked, not certified, not checked, or showing all the damages. In addition to this, there is also the option to sort the table by entry date, and so on (see Figure 3.21).

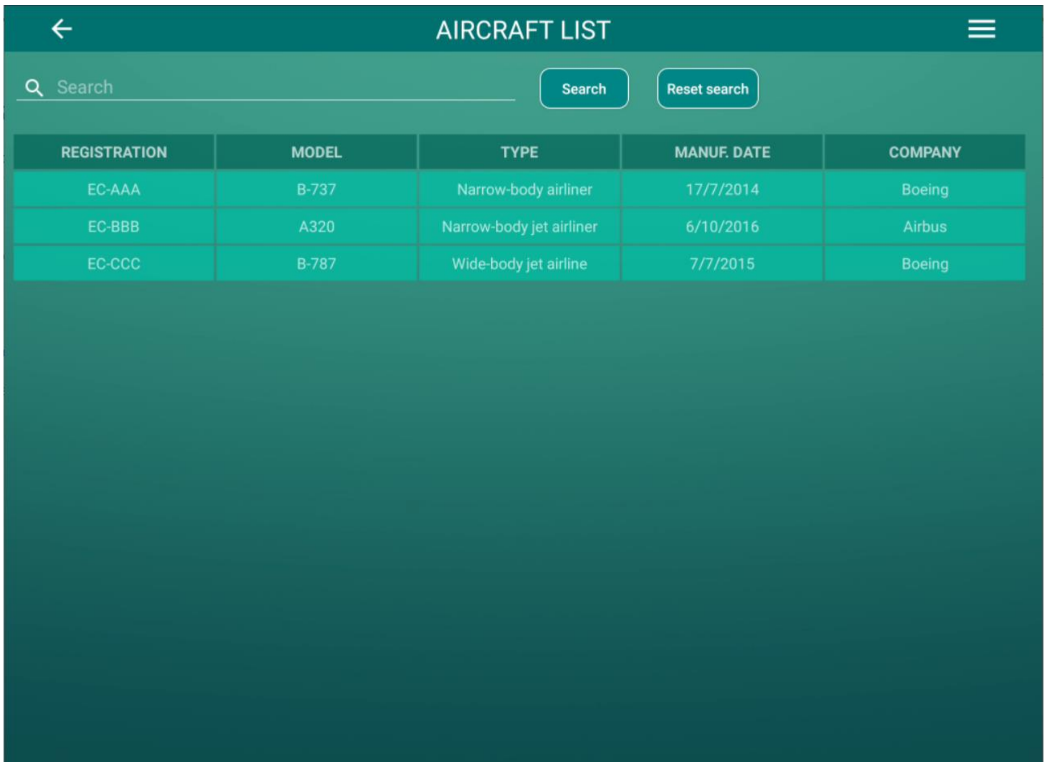


ID	ENTRY DATE	DESCRIPTION	DIMENSIONS	CAT.	SDT RB	C
21821_0109	13/7/2021	Scratch_Fuselage skin FR 25-26 STRG 41-42 LH	85x47x0.52mm	A	N/A	

Figure 3.21 Sort menu at damages screen

3.7. Fleet module

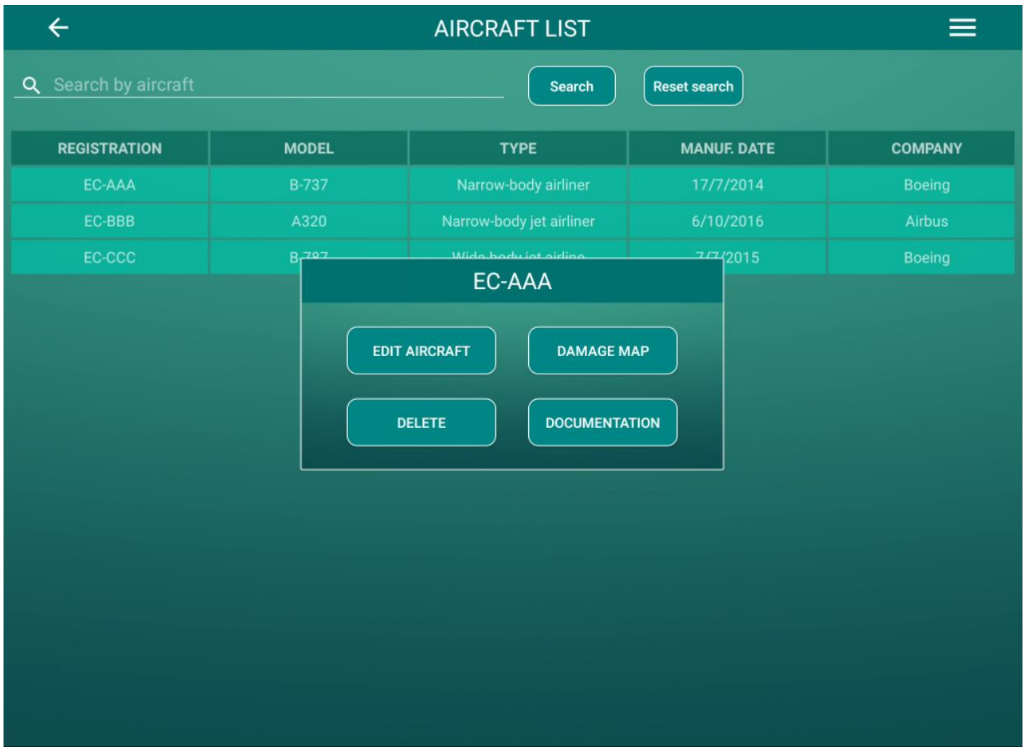
Only administrators have access to this module. Here they can see and manage (see Figure 3.22) all aircraft belonging to a company, add new aircrafts manually or remove it if necessary. This module is useful to keep track of aircrafts and their information, but it is also useful because it gives users the possibility to access to the damage map and documentation of a particular aircraft.



REGISTRATION	MODEL	TYPE	MANUF. DATE	COMPANY
EC-AAA	B-737	Narrow-body airliner	17/7/2014	Boeing
EC-BBB	A320	Narrow-body jet airliner	6/10/2016	Airbus
EC-CCC	B-787	Wide-body jet airline	7/7/2015	Boeing

Figure 3.22 Aircraft list screen

When administrators click on an aircraft, a menu will pop up (see Figure 3.23) with an edit, damage map, delete and documentation option. It will also display the aircraft registration.



REGISTRATION	MODEL	TYPE	MANUF. DATE	COMPANY
EC-AAA	B-737	Narrow-body airliner	17/7/2014	Boeing
EC-BBB	A320	Narrow-body jet airliner	6/10/2016	Airbus
EC-CCC	B-787	Wide-body jet airline	7/7/2015	Boeing

Figure 3.23 Aircraft list menu

If they want to add an aircraft, they must click on the menu located at the top right hand of the screen and a window will pop up with a form to fill (see Figure 3.24).

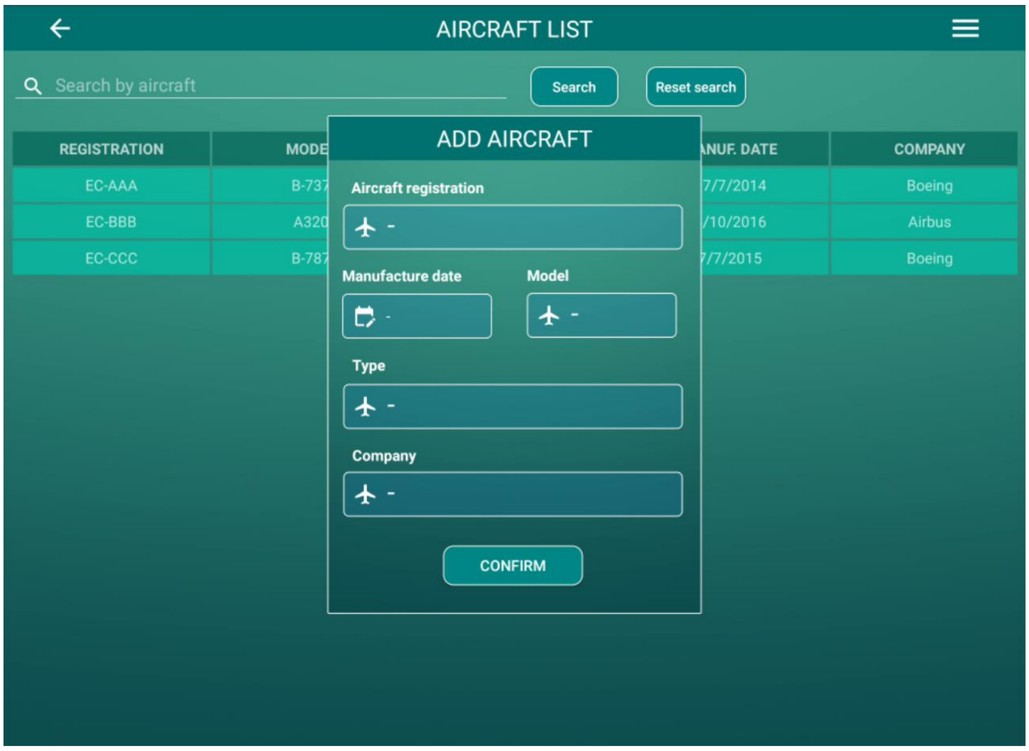


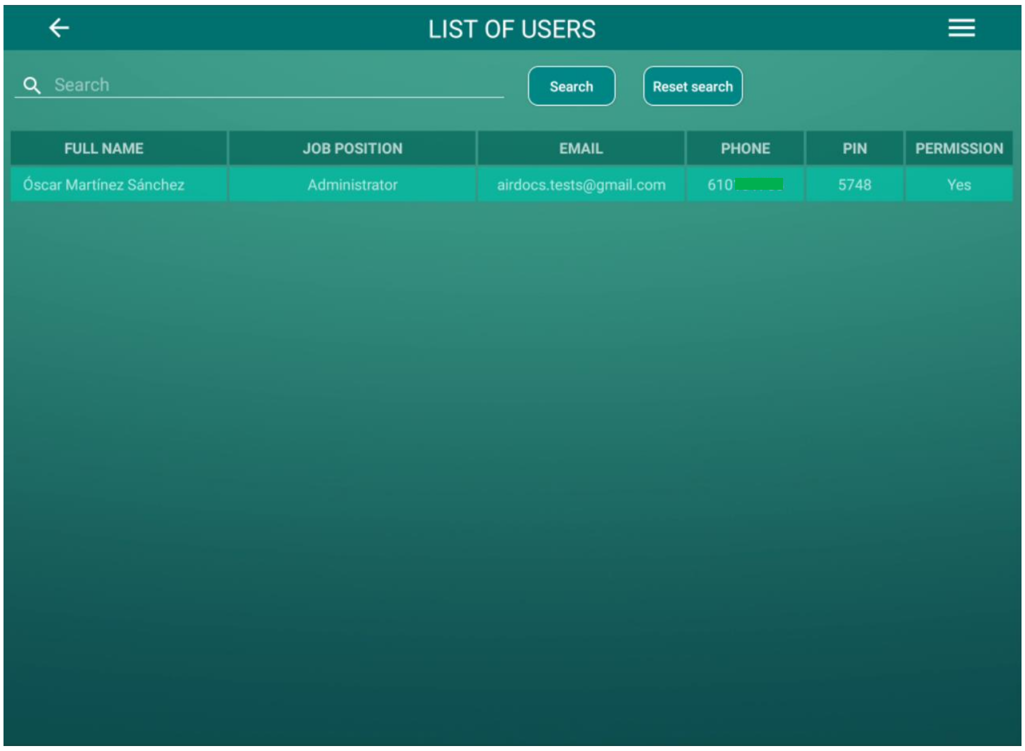
Figure 3.24 Adding a new aircraft

3.8. Flight crew module

In this module the users with a fight crew or administrator role can access to the damage map of a particular aircraft, visualize damages, and report them if detected. This is useful in case the crew detects a minor fault during the flight, such as an electronic device not working so they can report it for later repair. It is also useful so that pilots can check previous damages detected on aircrafts and if they are correctly fixed before take-off or, as mentioned above, if any device fails, they can report it. To report them, they should go to the "Messaging" module (see section 3.10) where they can send an email to the engineers informing about the problem.

3.9. Users' module

Like the fleet module, only administrators have access to this module. Here they can see and manage (see Figure 3.25) all users within the app, add new users by manually filling in a form or remove their access to AirDocs 2.0. This module is useful to keep track of who is accessing the app and to know their information. The data displayed is the same as shown at Table 1.1.



FULL NAME	JOB POSITION	EMAIL	PHONE	PIN	PERMISSION
Óscar Martínez Sánchez	Administrator	airdocs.tests@gmail.com	610[redacted]	5748	Yes

Figure 3.25 Users list screen

To add users, administrators must click on the top right hand of the screen where a menu will pop up with this option. Then they will be redirected to a screen (see Figure 3.26) to fill a form. The information which the administrator will find is explained at Table 3.6.

Table 3.5 User information at add user screen

Email	User's email.
Job position	Describes the user's job: administrator, CAMO engineer, MRO engineer, total access engineer (can do the same actions as the MRO and CAMO engineers), AMT and cabin crew.
Name	User's full name.
PIN	PIN created by the user to confirm actions inside the app.
Phone	Contact phone number.
Password	User's password to access the app
Confirm password	Field to check if password was written correctly.
Permission	This shows if a user has permission to enter the app. "Yes" if the user can access and "No" if he cannot access. This is changed by the administrator to manage to control who uses the app.

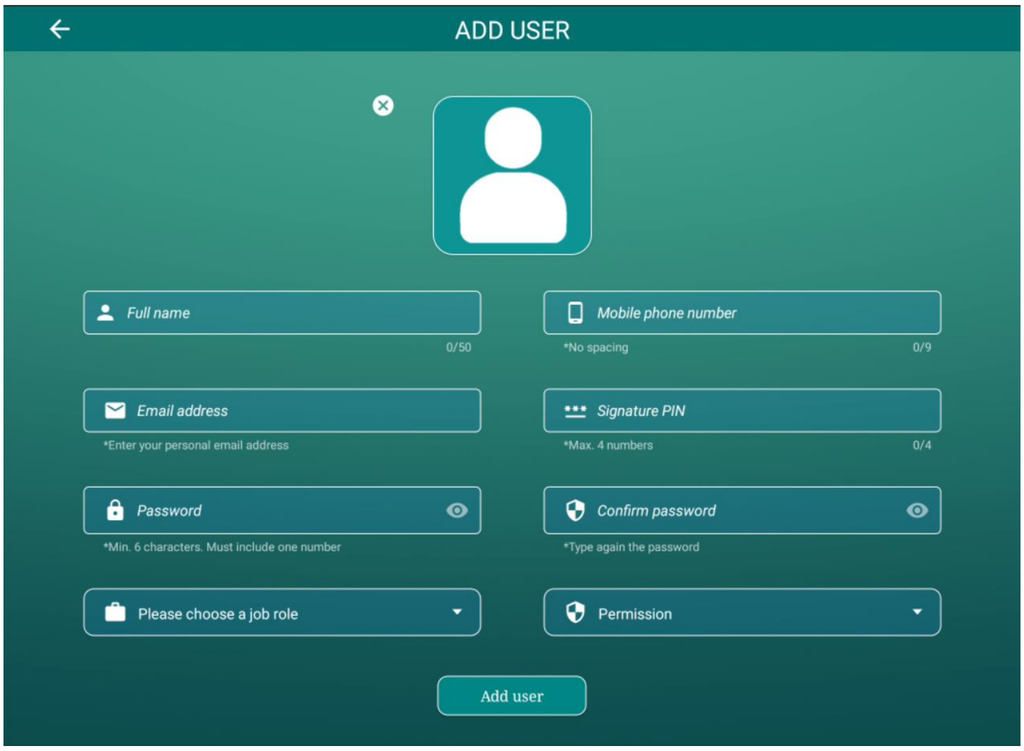
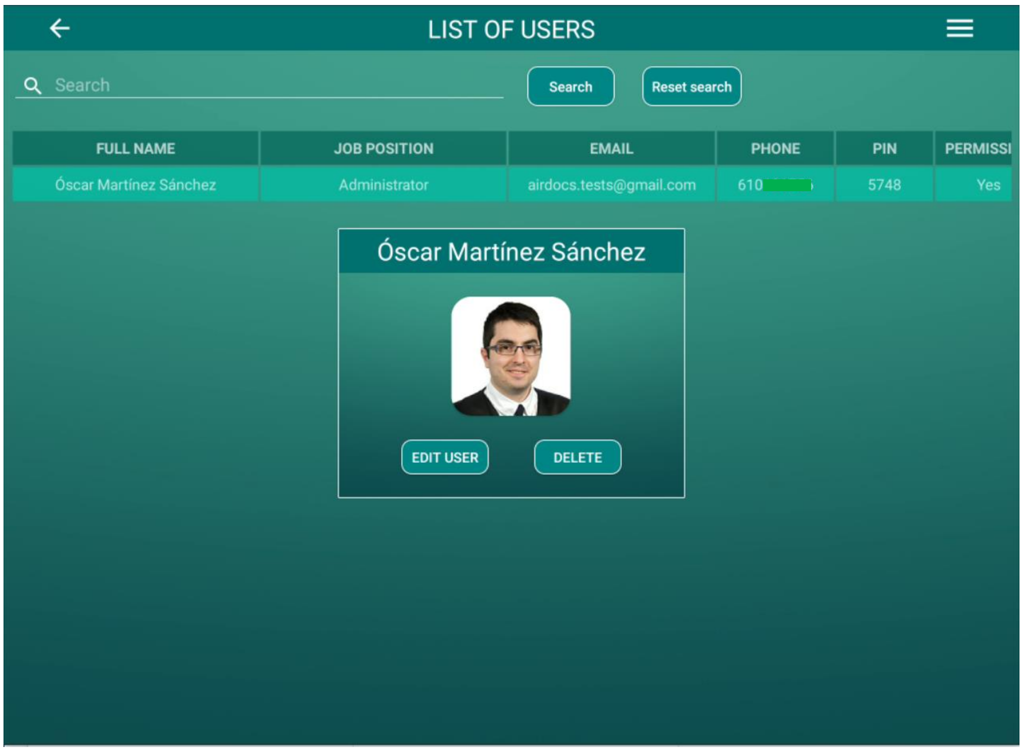


Figure 3.26 Add user screen

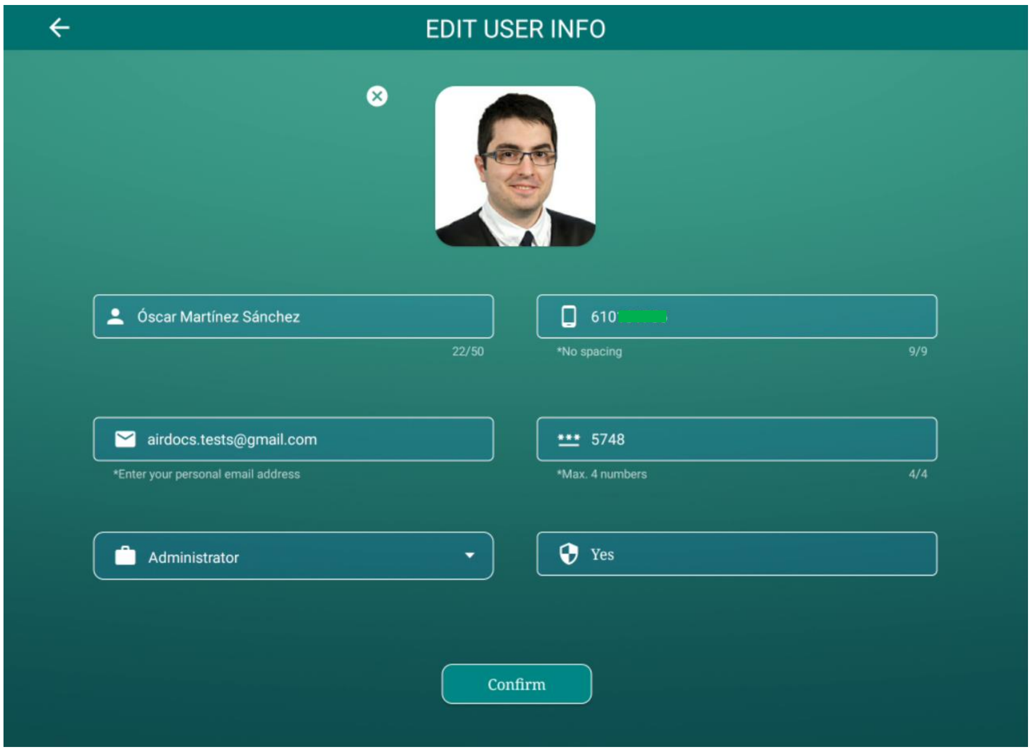
If administrators click on a user in the list, a menu will pop up with an edit and delete options (see Figure 3.27). It will also display its name and their profile photo if there is any.



FULL NAME	JOB POSITION	EMAIL	PHONE	PIN	PERMISSI
Óscar Martínez Sánchez	Administrator	airdocs.tests@gmail.com	610[redacted]	5748	Yes

Figure 3.27 User information menu

When edit user button is clicked, administrators will be redirected to another screen (see Figure 3.28) where they can change the same information seen in Table 1.1.


EDIT USER INFO

Óscar Martínez Sánchez 22/50

610[redacted] *No spacing 9/9

airdocs.tests@gmail.com *Enter your personal email address

5748 *Max. 4 numbers 4/4

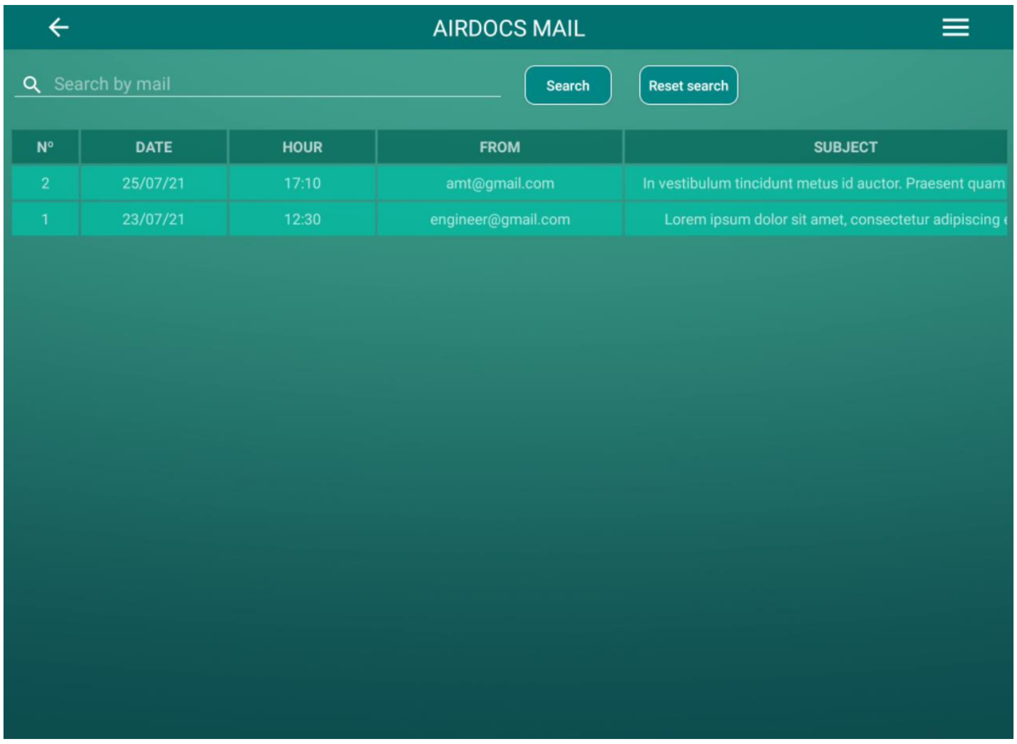
Administrator

Yes

Figure 3.28 Edit user screen

3.10. Messaging module

This module consists of sending and receiving emails (see Figure 3.29). AirDocs 2.0 offers two options to send mails: from the app or from their personal mail provider app within their device (Gmail, Hotmail and so on).



Nº	DATE	HOUR	FROM	SUBJECT
2	25/07/21	17:10	amt@gmail.com	In vestibulum tincidunt metus id auctor. Praesent quam
1	23/07/21	12:30	engineer@gmail.com	Lorem ipsum dolor sit amet, consectetur adipiscing

Figure 3.29 AirDocs 2.0 mail screen

If users select sending an email from with other service a window will pop up with data to fill shown at Fig 3.30 to send it.

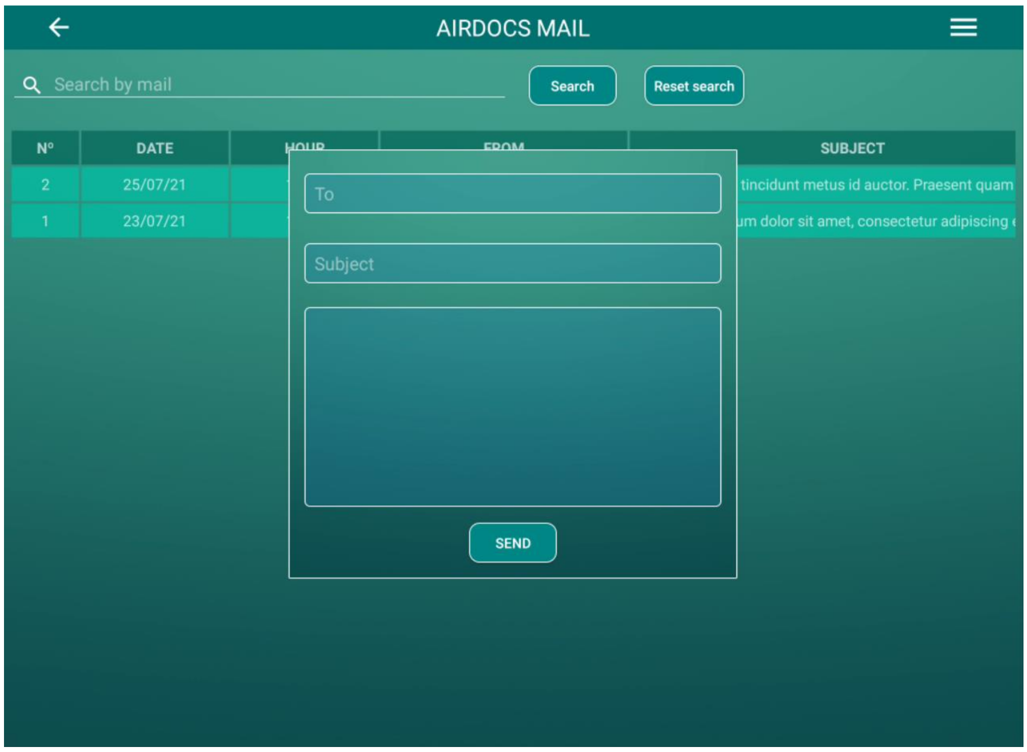
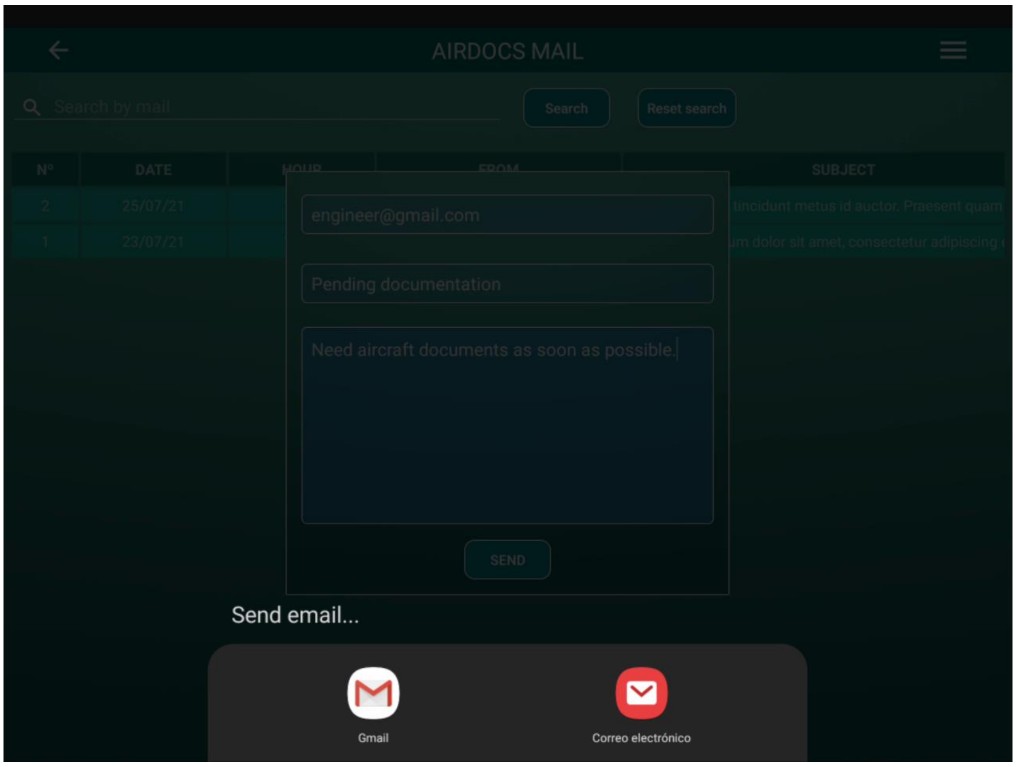


Figure 3.30 AirDocs 2.0 mail writing

Or if users select sending an email from with other service Once the user clicks send it will show the mail provider apps installed on the device (see Figure 3.31).



Nº	DATE	SUBJECT
2	25/07/21	tincidunt metus id auctor. Praesent quam
1	23/07/21	um dolor sit amet, consectetur adipiscing

engineer@gmail.com

Pending documentation

Need aircraft documents as soon as possible.

Figure 3.32 Selecting an email provider

CHAPTER 4. AIRDOCS 2.0 ON PC

In this section, it explains how to use AirDocs 2.0 on PC. The main process to do so is to mirror the device's screen on the PC. While nowadays this a feature in development, now the choice is to use a free open-source software called "scrcrpy" so the user can work using the PC keyboard and all the features that the PC offers. This is a temporal solution because, for example, Samsung devices have an official feature developed by them in its devices that can mirror the screen.

4.1. Scrcpy open source

Scrcpy is one of the best free open-source software among developers for mirroring Android device screen. It allows the user to mirror his device to the PC via USB cable (charger cable for example) and use the PC keyboard and mouse for a more comfortable experience. Features like copy and paste between the device and PC and vice versa is possible. This platform is available on GitHub and developers are updating it frequently to improve its features so new versions are launched frequently.

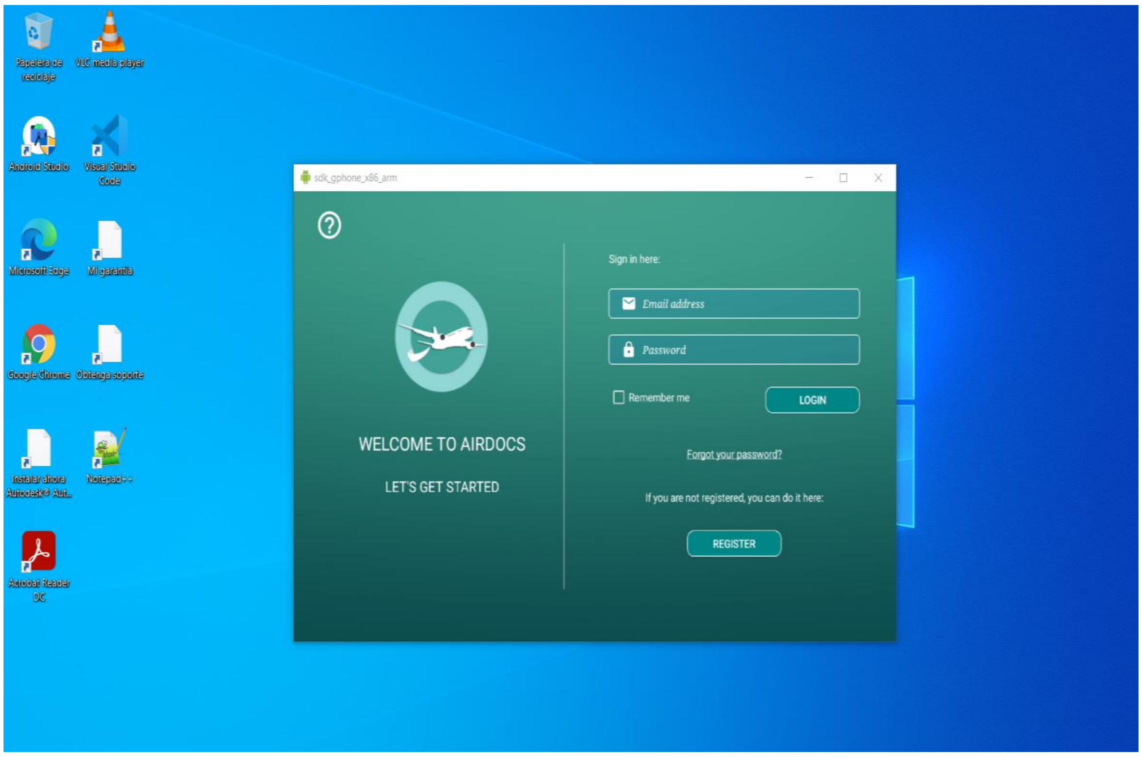
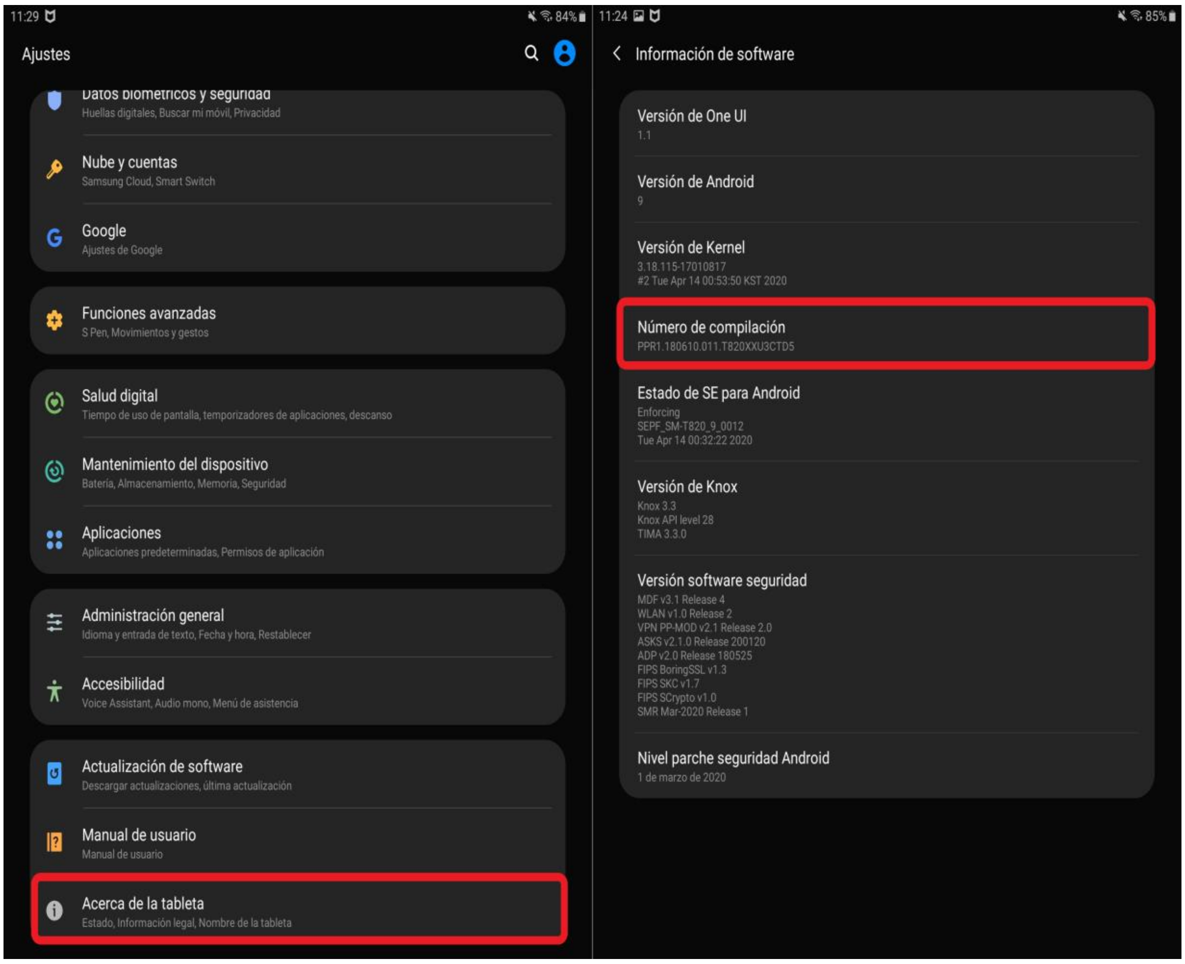


Figure 4.1 Mirroring screen device on Windows 10

4.2. Connecting to PC via USB

Before using scrcpy there are a few steps the users must take. To make it more visual, the process is shown in a Samsung device.

1. Enable USB debugging settings in the developer options on the device.
    - Go to Settings → About tablet (or phone) → Software information.
    - Tap on Build Number five or seven times (depends on the device).
    - Go back to Settings and tap on Developer options.
    - Activate USB debugging.



11:29

Ajustes

Datos biométricos y seguridad
Huellas digitales, Buscar mi móvil, Privacidad

Nube y cuentas
Samsung Cloud, Smart Switch

Google
Ajustes de Google

Funciones avanzadas
S Pen, Movimientos y gestos

Salud digital
Tiempo de uso de pantalla, temporizadores de aplicaciones, descanso

Mantenimiento del dispositivo
Batería, Almacenamiento, Memoria, Seguridad

Aplicaciones
Aplicaciones predeterminadas, Permisos de aplicación

Administración general
Idioma y entrada de texto, Fecha y hora, Restablecer

Accesibilidad
Voice Assistant, Audio mono, Menú de asistencia

Actualización de software
Descargar actualizaciones, última actualización

Manual de usuario
Manual de usuario

Acerca de la tableta
Estado, Información legal, Nombre de la tableta

11:24

Información de software

Versión de One UI
1.1

Versión de Android
9

Versión de Kernel
3.18.115-17010817
#2 Tue Apr 14 00:53:50 KST 2020

Número de compilación
PPR1.180610.011.T820XXU3CTD5

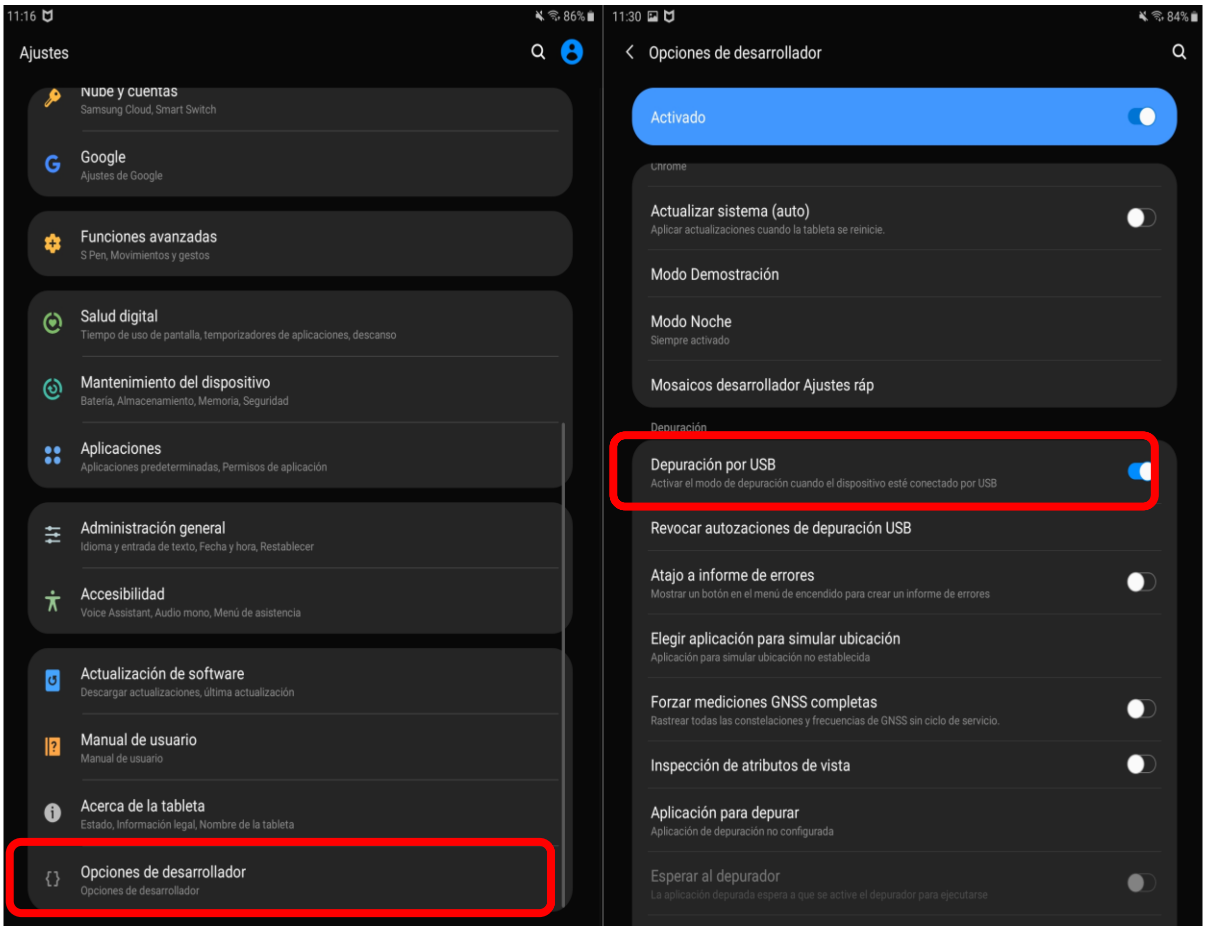
Estado de SE para Android
Enforcing
SEPF_SM-T820_9_0012
Tue Apr 14 00:32:22 2020

Versión de Knox
Knox 3.3
Knox API level 28
TIMA 3.3.0

Versión software seguridad
MDF v3.1 Release 4
WLAN v1.0 Release 2
VPN PP-MOD v2.1 Release 2.0
ASKS v2.1.0 Release 200120
ADP v2.0 Release 180525
FIPS BoringSSL v1.3
FIPS SKC v1.7
FIPS SCrypto v1.0
SMR Mar-2020 Release 1

Nivel parche seguridad Android
1 de marzo de 2020

Figure 4.2 Enabling developer mode


11:16 86%

Ajustes

Nube y cuentas
Samsung Cloud, Smart Switch

Google
Ajustes de Google

Funciones avanzadas
S Pen, Movimientos y gestos

Salud digital
Tiempo de uso de pantalla, temporizadores de aplicaciones, descanso

Mantenimiento del dispositivo
Batería, Almacenamiento, Memoria, Seguridad

Aplicaciones
Aplicaciones predeterminadas, Permisos de aplicación

Administración general
Idioma y entrada de texto, Fecha y hora, Restablecer

Accesibilidad
Voice Assistant, Audio mono, Menú de asistencia

Actualización de software
Descargar actualizaciones, última actualización

Manual de usuario
Manual de usuario

Acerca de la tableta
Estado, Información legal, Nombre de la tableta

Opciones de desarrollador
Opciones de desarrollador

11:30 84%

Opciones de desarrollador

Activado

Chrome

Actualizar sistema (auto)
Aplicar actualizaciones cuando la tableta se reinicie.

Modo Demostración

Modo Noche
Siempre activado

Mosaicos desarrollador Ajustes ráp

Depuración

Depuración por USB
Activar el modo de depuración cuando el dispositivo esté conectado por USB

Revocar autozaciones de depuración USB

Atajo a informe de errores
Mostrar un botón en el menú de encendido para crear un informe de errores

Elegir aplicación para simular ubicación
Aplicación para simular ubicación no establecida

Forzar mediciones GNSS completas
Rastrear todas las constelaciones y frecuencias de GNSS sin ciclo de servicio.

Inspección de atributos de vista

Aplicación para depurar
Aplicación de depuración no configurada

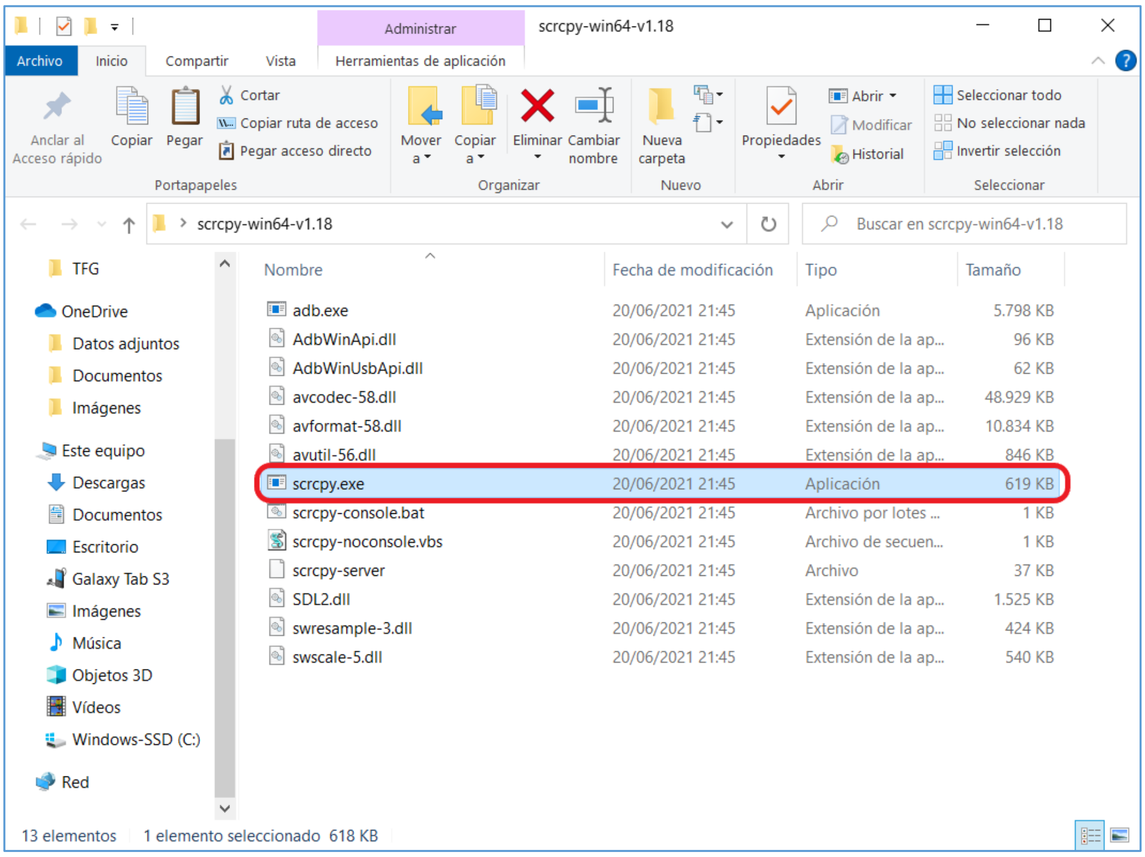
Esperar al depurador
La aplicación depurada espera a que se active el depurador para ejecutarse

Figure 4.3. Enabling USB debugging

2. Connect the device via USB. The best option is to use the charger cable of the device.

Figure 4.4 Connecting user device to PC via USB

3. Finally, the user only needs to open scrcpy to start using it. In section 4.3 there are shortcuts to improve the user experience.



Nombre	Fecha de modificación	Tipo	Tamaño
adb.exe	20/06/2021 21:45	Aplicación	5.798 KB
AdbWinApi.dll	20/06/2021 21:45	Extensión de la ap...	96 KB
AdbWinUsbApi.dll	20/06/2021 21:45	Extensión de la ap...	62 KB
avcodec-58.dll	20/06/2021 21:45	Extensión de la ap...	48.929 KB
avformat-58.dll	20/06/2021 21:45	Extensión de la ap...	10.834 KB
avutil-56.dll	20/06/2021 21:45	Extensión de la ap...	846 KB
scrcpy.exe	20/06/2021 21:45	Aplicación	619 KB
scrcpy-console.bat	20/06/2021 21:45	Archivo por lotes ...	1 KB
scrcpy-noconsole.vbs	20/06/2021 21:45	Archivo de secuen...	1 KB
scrcpy-server	20/06/2021 21:45	Archivo	37 KB
SDL2.dll	20/06/2021 21:45	Extensión de la ap...	1.525 KB
swresample-3.dll	20/06/2021 21:45	Extensión de la ap...	424 KB
swscale-5.dll	20/06/2021 21:45	Extensión de la ap...	540 KB

13 elementos 1 elemento seleccionado 618 KB

Figure 4.5 Launch scrcpy on Windows 10

4.3. Scrcpy shortcuts

In Table 4.1 there all the shortcuts that it is recommended to work with when using scrcpy to improve the experience.

Table 4.1 Scrcpy shortcuts

Action	Shortcut
Switch full screen mode	Alt + f
Rotate display left	Alt + ← *(left)*
Rotate display right	Alt + → *(right)*
Resize window to 1:1 (pixel-perfect)	Alt + g
Click on MENU (unlock screen)	Alt + m

Click on VOLUME_UP	Alt + ↑ *(up)*
Click on VOLUME_DOWN	Alt + ↓ *(down)*
Click on POWER	Alt + p
Turn device screen off (keep mirroring)	Alt + o
Turn device screen on	Alt + Shift + o
Rotate device screen	Alt + r
Collapse panels	Alt + Shift + n
Copy to clipboard	Alt + c
Cut to clipboard	Alt + x
Synchronize clipboards and paste	Alt + v
Inject computer clipboard text	Alt + Shift + v
Enable/disable FPS counter	Alt + i
Pinch-to-zoom	Ctrl + *click-and-move*
Click on HOME	Ctrl+h | Middle-click
Click on BACK	Ctrl+b | Right-click
Click on APP_SWITCH	Ctrl+m
Turn screen on	Right-click

CHAPTER 5. ECONOMIC COST

The price of making a brand-new app can range from 20.000 to 250.000 euros. The creation of a basic application would be approximately $90.000, hiring engineers for an average price of about $40 per hour. These costs are based on a series of factors:

- Type of app
- Type of operating system (iOS, Android…)
- Developing time
- Country hiring prices
- Number and complexity of features
- Number of platforms
- Design complexity

These factors have a direct influence on the final cost result, which can also increase depending on the customer's requirements to change or add elements during its developing.

Table 5.1 Average cost to create an app

Type of app	Time to develop	Cost
Simple	3-6 months	$40.000 - $60.000
Medium	6-10 months	$60.000 - $120.000
Complex	10 months or more	$120.000 - $200.000+

Regarding on which country is being developed, focusing on the hourly cost of contracting a worker, the price ranges from $35 per hour in Indonesia to $150 in the USA (see Table 5.2). Here in Western Europe the prices vary from 45$ to 85$ making a business app. The cost of hiring an app developer in Spain is around 45$ per hour (approximately 35€ per hour), so for calculating the final cost of AirDocs 2.0 a cost of 45$ per hour has been chosen.

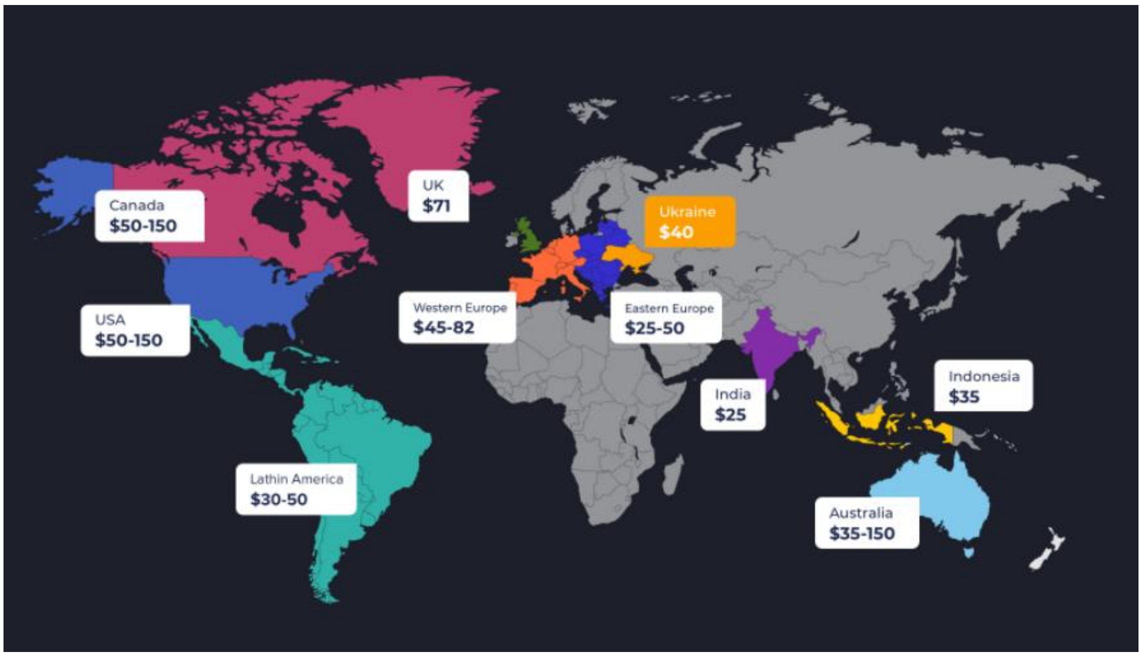


Figure 5.1 Software development rates by region. Source: Cleveroad

AirDocs 2.0 has been programmed for approximately 6 months, this are 144 days excluding weekends and holidays. This calculation will have a full working time of 8 hours, about 40 hours per week, which in total is about 1200 hours of time invested during the 6 months of development. Taking the previous cost of 45$ per hour, the final development cost of AirDocs is approximately 54.000$.

Looking in more depth, Table 5.3 explains in detail the prices and time spent for each item entered in AirDocs 2.0 (these data are calculated on an approximate basis).

Table 5.2 Approximately price and time per element added of AirDocs 2.0

App Feature	Description	Time	Cost
Login	• Login with email • Log out • Forgot password option • Remember credentials	160 h	4500$
Register	• Add/remove profile picture • Users can introduce their personal information • Verification mail	160 h	4500$

Home screen	• User information displayed • Redesign of home screen for all user job roles • Possibility of changing user personal information • Resetting password and change email feature	200 h	9000$
Engineer, administrator, AMT, and cabin crew modules	• New code • New features • User interface improved • Previous features improved	500 h	22.500$
Firebase database	• New database • Security improved • Realtime database	10 h	450$
Basic messaging	• Own messaging system • Possibility to use other mail providers	50 h	2.250$
File uploading	• Listing of all files uploaded • Downloading feature • No need to exit the application	50 h	2.250$
PC version	• Removed the PC version • App in an Android version can be connected directly to PC • More portability	10 h	450$
Android configuration	• App supports different screen sizes • Landscape and portrait mode • Mobile and tablet version	50 h	7.650$
Other features	• Splash screen • New logo • New aesthetic	10 h	450$

The final cost does not include the price of the software and hardware used during development and it’s calculated only with one engineer hired. On the other hand, the cost will vary according to the selling price, how many engineers are hired and other factors.

.

CHAPTER 6. CONCLUSIONS AND PROJECT CONTINUITY

This project arose from the idea of improving the user experience in AirDocs first version, previously developed by Laura Huguet González. The main objective of this work is adding or change the features already implemented so that once the user is using the application, they feel a good working experience and satisfies the needs of the users.

The new version of AirDocs 2.0 has been made possible thanks to new technologies and methods that make it possible to code applications for all types of devices so that it is possible to work anywhere and eliminate the use of paper in the whole process so information can be shared quickly and smoothly between users registered in the application.

This new version has been developed and created from zero entirely with Android Studio, which is exclusively for the development of Android applications and offers a great deal of customisation. To be able to carry out the programming task, it has been necessary to learn from scratch a new language called Kotlin, which developers are beginning to substitute for older languages, and in the future, we will see many applications developed with Kotlin.

In terms of data storage, Google's Firebase has been implemented. Firebase offers a dedicated server located on Google's servers offering great security and storage space. This method was chosen because this service is always operational, i.e., the person in charge of the application does not have to log in and be online so that other users can work with AirDocs 2.0 because previously the administrator needed to be logged in always to DropBox to be able to work.

For this application a PC version has not been created but is replaced by a direct connection via a cable to the PC and working on the screen with AirDocs 2.0. This is possible thanks to a programme called "scrcpy" with which it is possible to project the screen of the device onto the computer. This has been implemented with the future in mind, as users will always have their mobile device with them and will only need to connect it to the PC to work. Another reason is the update of the application. While the old version required several codes for the different platforms, which entailed a risk of errors if modified, now it will only be necessary to modify one code and apply the changes in the application. It should be noted that the iOS version will also disappear. This has been decided due to the large growth of devices using Android, an operating system that offers a great deal of customisation.

As for the continuity of the project, during and at the end of the development of the new version of AirDocs, new ideas emerged that could be added in the future. Ideas such as the introduction of a 3D modelling of an aircraft, the removal of keyboard typing for a full graphical interface and several other improvements emerged. These improvements could be added in the new version by whoever takes over this project.

The ultimate objective of AirDocs 2.0 is to offer a complete experience, with the possibility of improving and adding enhancements according to the customer's needs over the coming years, thanks to the fact that Android will allow future technologies to be adapted to those of the present.

WEB BIBLIOGRAPHY

[1]: Kotlin Programming Language: https://kotlinlang.org/

[2]: Android Developers: https://developer.android.com/

[3]: Best features for Android Studio developers: https://www.admecindia.co.in/miscellaneous/top-10-features-android-studio-developers-not-miss/

[4]: Pros and cons of Android app development: https://www.altexsoft.com/blog/engineering/pros-and-cons-of-android-app-development/

[5]: Pros and cons of using Firebase: https://www.altexsoft.com/blog/firebase-review-pros-cons-alternatives/

[6]: What is Firebase?: https://medium.com/firebase-developers/what-is-firebase-the-complete-story-abridged-bcc730c5f2c0

[7]: González Huguet, Laura, “MRO 4.0: Diseño de una aplicación para gestionar documentación relativa al mantenimiento de aeronaves”, 2020: https://upcommons.upc.edu/handle/2117/334907

Annexed A: AirDocs 2.0 code

A.1. Add check activity

```
class AddCheckActivity : AppCompatActivity() {

    var timestamp_end : String? = null
    var timestamp_crs : String? = null

    override fun onCreate(savedInstanceState: Bundle?) {

        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_addcheck)

        val title: TextView = findViewById(R.id.tool_title)
        title.text = "Add aircraft check"

        val btn_menu : ImageButton = findViewById(R.id.btn_menu)
        btn_menu.visibility = View.GONE

        val aircraft = findViewById<EditText>(R.id.inputAircraft)
        aircraft.onFocusChangeListener = View.OnFocusChangeListener { b,
hasFocus ->
            if (!hasFocus) {
                aircraft.setHint(R.string.NoHint)
                hideKeyboard(b)
            } else aircraft.hint = ""
        }

        val assign = findViewById<EditText>(R.id.inputAssign)
        assign.onFocusChangeListener = View.OnFocusChangeListener { a,
hasFocus ->
            if (!hasFocus) {
                assign.setHint(R.string.NoHint)
                hideKeyboard(a)
            } else assign.hint = ""
        }

        val revision = resources.getStringArray(R.array.revision_array)
        val spinner: Spinner = findViewById(R.id.spinner_check)
        val adapter = ArrayAdapter(this, R.layout.spinner_item_selected, revision)
        adapter.setDropDownViewResource(R.layout.spinner_dropdown_item)
        spinner.adapter = adapter
        spinner.avoidDropdownFocus()
        spinner.onItemSelectedListener = object :
AdapterView.OnItemSelectedListener {
            override fun onItemSelected(
                parent: AdapterView<*>,
                view: View?,
                position: Int,
                id: Long
            ) {
                if (parent.getItemAtPosition(position) == "-") {
                } else {
                    val SHARED_PREF_CHECK= "CHECK";
                    val KEY_CHECK= "key_CHECK";
                    val mSpnValue: String = spinner.selectedItem.toString()
                    val myPrefs: SharedPreferences =
getSharedPreferences(SHARED_PREF_CHECK, MODE_PRIVATE)
                    val prefsEditor = myPrefs.edit()
                    prefsEditor.putString(KEY_CHECK, mSpnValue)
                    prefsEditor.apply()
                }
            }

            override fun onNothingSelected(parent: AdapterView<*>?) {}
        }

        val status = resources.getStringArray(R.array.status_array)
        val spinner2: Spinner = findViewById(R.id.spinner_status)
        val adapter2 = ArrayAdapter(this, R.layout.spinner_item_selected, status)
        adapter2.setDropDownViewResource(R.layout.spinner_dropdown_item)
        spinner2.adapter = adapter2
        spinner.avoidDropdownFocus()
        spinner2.onItemSelectedListener = object :
AdapterView.OnItemSelectedListener {
            override fun onItemSelected(
                parent: AdapterView<*>,
                view: View?,
                position: Int,
                id: Long
            ) {
                if (parent.getItemAtPosition(position) == "-") {
                } else {
                    val SHARED_PREF_STATUS = "STATUS";
                    val KEY_STATUS= "key_STATUS";
                    val mSpnValue: String = spinner2.selectedItem.toString()
                    val myPrefs: SharedPreferences =
getSharedPreferences(SHARED_PREF_STATUS, MODE_PRIVATE)
                    val prefsEditor = myPrefs.edit()
                    prefsEditor.putString(KEY_STATUS, mSpnValue)
                    prefsEditor.apply()
                }
            }

            override fun onNothingSelected(parent: AdapterView<*>?) {}
        }
        val calendar = Calendar.getInstance()
        val edit = findViewById<View>(R.id.inputBegin) as EditText
        val yy = calendar.get(Calendar.YEAR)
        val mm = calendar.get(Calendar.MONTH)
        val dd = calendar.get(Calendar.DAY_OF_MONTH)
        val datePicker = DatePickerDialog(
                this@AddCheckActivity,
                { _, year, monthOfYear, dayOfMonth ->
                    val date =
                            (dayOfMonth.toString() + "/" + (monthOfYear + 1).toString() +
"/" + year.toString())
                    edit.setText(date)
                },
                yy,
                mm,
                dd
        )
        datePicker.setButton(DialogInterface.BUTTON_NEUTRAL, "CLEAR") { _,
which ->
            if (which == DialogInterface.BUTTON_NEUTRAL) {
                edit.setText("-")
            }
        }

        edit.setOnClickListener() {
            datePicker.show()
        }

        val calendar2 = Calendar.getInstance()
        val edit2 = findViewById<View>(R.id.inputEnd) as EditText
        val yy2 = calendar2.get(Calendar.YEAR)
        val mm2 = calendar2.get(Calendar.MONTH)
        val dd2 = calendar2.get(Calendar.DAY_OF_MONTH)
        val datePicker2 = DatePickerDialog(
                this@AddCheckActivity,
                { _, year, monthOfYear, dayOfMonth ->
                    val date2 =
                            (dayOfMonth.toString() + "/" + (monthOfYear + 1).toString() +
"/" + year.toString())
                    edit2.setText(date2)
                },
                yy2,
                mm2,
                dd2
        )
        datePicker2.setButton(DialogInterface.BUTTON_NEUTRAL, "CLEAR") {
_, which ->
            if (which == DialogInterface.BUTTON_NEUTRAL) {
                edit2.setText("-")
            }
        }

        edit2.setOnClickListener() {
            datePicker2.show()
        }

        val calendar3 = Calendar.getInstance()
        val edit3 = findViewById<View>(R.id.inputCRS) as EditText
        val yy3 = calendar3.get(Calendar.YEAR)
        val mm3 = calendar3.get(Calendar.MONTH)
        val dd3 = calendar3.get(Calendar.DAY_OF_MONTH)
        val datePicker3 = DatePickerDialog(
                this@AddCheckActivity,
                DatePickerDialog.OnDateSetListener { _, year, monthOfYear,
dayOfMonth ->
                    val date3 =
                            (dayOfMonth.toString() + "/" + (monthOfYear + 1).toString() +
"/" + year.toString())
                    edit3.setText(date3)
                },
                yy3,
                mm3,
                dd3
        )
        datePicker3.setButton(DialogInterface.BUTTON_NEUTRAL, "CLEAR") {
_, which ->
            if (which == DialogInterface.BUTTON_NEUTRAL) {
                edit3.setText("-")
            }
        }

        edit3.setOnClickListener() {
            datePicker3.show()
        }

        val add: Button = findViewById(R.id.btn_Add)
        add.setOnClickListener {
            saveCheck()
        }

        aircraft.setOnClickListener {
            saveEditText()
            val i = Intent(this@AddCheckActivity, AddPlaneToCheck::class.java)
            startActivity(i)
            finish()
        }

        assign.setOnClickListener {
            saveEditText()
            val i = Intent(this@AddCheckActivity, AddUserToCheck::class.java)
            startActivity(i)
            finish()
        }

        val pref = getSharedPreferences("FLEET", MODE_PRIVATE)
        val fleet = pref.getString("plane", "")
        if (fleet == null) {
        } else {
            aircraft.setText(fleet)
        }

        val pref2 = getSharedPreferences("USER", MODE_PRIVATE)
        val name = pref2.getString("user", "")
        if(name  == null) { }
        else{
            assign.setText(name)
        }

        val remove_plane : ImageButton = findViewById(R.id.ivCancel)
        remove_plane.setOnClickListener {
            aircraft.setText("-")
        }

        val remove_assign : ImageButton = findViewById(R.id.ivCancel2)
        remove_assign.setOnClickListener {
            assign.setText("-")
        }

        displayEditText()

        val back = findViewById<View>(R.id.btn_back) as ImageButton
        back.setOnClickListener {
            pref.edit().remove("plane").apply()
            pref2.edit().remove("user").apply()
            deletePreference()
            reset()
            openActivity1()
        }

    }

    private fun setSpinText(spin: Spinner, text: String?) {
        for (i in 0 until spin.adapter.count) {
            if (spin.adapter.getItem(i).toString().contains(text.toString())) {
                spin.setSelection(i)
            }
        }
    }

    private fun Spinner.avoidDropdownFocus() {
        try {
            val isAppCompat = this is
androidx.appcompat.widget.AppCompatSpinner
            val spinnerClass = if (isAppCompat)
androidx.appcompat.widget.AppCompatSpinner::class.java else
Spinner::class.java
            val popupWindowClass = if (isAppCompat)
androidx.appcompat.widget.ListPopupWindow::class.java else
android.widget.ListPopupWindow::class.java

            val listPopup = spinnerClass
                    .getDeclaredField("mPopup")
                    .apply { isAccessible = true }
                    .get(this)
            if (popupWindowClass.isInstance(listPopup)) {
                val popup = popupWindowClass
                        .getDeclaredField("mPopup")
                        .apply { isAccessible = true }
                        .get(listPopup)
                if (popup is PopupWindow) {
                    popup.isFocusable = false
                }
            }
        } catch (e: Exception) {
            e.printStackTrace()
        }
    }

    override fun onBackPressed() {
        super.onBackPressed()
        val intent = Intent(this, CheckListActivity::class.java)
        startActivity(intent)
    }

    private fun hideKeyboard(view: View) {
        val inputMethodManager =
getSystemService(INPUT_METHOD_SERVICE) as InputMethodManager
        inputMethodManager.hideSoftInputFromWindow(view.windowToken, 0)
    }

    private fun saveCheck() {

        val aircraft: EditText = findViewById(R.id.inputAircraft)
        val begin: EditText = findViewById(R.id.inputBegin)
        val end = findViewById<EditText>(R.id.inputEnd)
        val crs: EditText = findViewById(R.id.inputCRS)
        val assigned: EditText = findViewById(R.id.inputAssign)

        val spinner = findViewById<View>(R.id.spinner_check) as Spinner
        val spinner2 = findViewById<View>(R.id.spinner_status) as Spinner
        val check = spinner.selectedItem.toString()
        val status = spinner2.selectedItem.toString()
        val comment : EditText = findViewById(R.id.inputComment)

        val date = begin.text.toString()
        val items1 = date.split("/".toRegex()).toTypedArray()
        val day = items1[0]
        val month = items1[1]
        val year = items1[2]

        val time = "$month-$day-$year"
        val formatter: DateFormat = SimpleDateFormat("MM-dd-yyyy")
        val date_convert = formatter.parse(time) as Date
        val output = date_convert.time / 1000L
        val str = output.toString()
        val timestamp_begin = str.toLong() * 1000

        if (end.text.toString().isEmpty()){
            timestamp_end = "0"
        } else {
            val date_end = end.text.toString()
            val items2 = date_end.split("/".toRegex()).toTypedArray()
            val day2 = items2[0]
            val month2 = items2[1]
            val year2 = items2[2]
            val time2 = "$month2-$day2-$year2"
            val date_convert_end = formatter.parse(time2) as Date
            val output_end = date_convert_end.time / 1000L
            val str_end = output_end.toString()
            timestamp_end = ((str_end.toLong() * 1000).toString())
        }

        if (crs.text.toString().isEmpty()){
            timestamp_crs = "0"
        } else {
            val date_crs = crs.text.toString()
            val items3 = date_crs.split("/".toRegex()).toTypedArray()
            val day3 = items3[0]
            val month3 = items3[1]
            val year3 = items3[2]
            val time3 = "$month3-$day3-$year3"
            val date_convert_crs = formatter.parse(time3) as Date
            val output_crs = date_convert_crs.time / 1000L
            val str_crs = output_crs.toString()
            timestamp_crs = ((str_crs.toLong() * 1000).toString())
        }

        val timestamp_current = Calendar.getInstance().timeInMillis

        val db = FirebaseFirestore.getInstance()
```

```
        val df = db.collection("Aircraft revision").document(aircraft.text.toString()
+ "_" + check + "_" + day + month + year)
        val user = FirebaseAuth.getInstance().currentUser
        val userEmail = user.email
        val df2= db.collection("Users").document(userEmail.toString())
        df2.get().addOnSuccessListener { documentSnapshot ->
            val added = documentSnapshot.getString("Name")
            df.get().addOnSuccessListener { documentSnapshot ->
                if (documentSnapshot.getString("Start") == begin.text.toString()
                        && documentSnapshot.getString("Aircraft") ==
aircraft.text.toString()) {
                    val builder = AlertDialog.Builder(this)
                    builder.setTitle("Error data")
                    builder.setMessage("There is already an aircraft check that
begins the same date.")
                    builder.setNegativeButton("Cancel", null)
                    builder.create().show()
                } else {
                    if (aircraft.text.isEmpty() || check == "-" || begin.text.toString() == "-
") {
                        val builder = AlertDialog.Builder(this)
                        builder.setTitle("Error data")
                        builder.setMessage("Aircraft, begin date and check fields
must not be empty.")
                        builder.setNegativeButton("Close", null)
                        builder.create().show()
                    } else {
                        val df2 = db.collection("Aircraft revision")
                        df2.get().addOnSuccessListener {
                            if (spinner2.selectedItem.toString() == "-" ||
                                    end.text.toString().isEmpty() ||
crs.text.toString().isEmpty() || assigned.text.toString().isEmpty()) {
                                if (begin.text.toString().isEmpty()) {
                                    begin.setText("-")
                                }
                                if (end.text.toString().isEmpty()) {
                                    end.setText("-")
                                }
                                if (crs.text.toString().isEmpty()) {
                                    crs.setText("-")
                                }
                                if (assigned.text.toString().isEmpty()) {
                                    assigned.setText("-")
                                }
                                val builder = AlertDialog.Builder(this)
                                builder.setTitle("Confirm aircraft revision data")
                                builder.setMessage("Some fields are empty. Do you
want to continue?")
                                builder.setPositiveButton(
                                        "Accept"
                        )
                        { _, _ ->

                            if (comment.text.toString() == "") {
                                val revisionInfo = hashMapOf(
                                        "Aircraft" to aircraft.text.toString(),
                                        "Check" to check,
                                        "Start" to begin.text.toString(),
                                        "End" to end.text.toString(),
                                        "CRS" to crs.text.toString(),
                                        "Status" to status,
                                        "Assigned" to assigned.text.toString(),
                                        "ID" to aircraft.text.toString() + "_" + check + "_"
+ day + month + year,
                                        "Comments" to comment.text.toString(),
                                        "Text" to "No",
                                        "Added" to added.toString(),
                                        "timestamp_begin" to
timestamp_begin.toString(),
                                        "timestamp_end" to timestamp_end,
                                        "timestamp_crs" to timestamp_crs,
                                        "timestamp_current" to
timestamp_current.toString())
                                db.collection("Aircraft revision")
                                        .document(aircraft.text.toString() + "_" + check +
"_" + day + month + year)
                                        .set(revisionInfo)
                                showSuccess()

                            } else {
                                val revisionInfo = hashMapOf(
                                        "Aircraft" to aircraft.text.toString(),
                                        "Check" to check,
                                        "Start" to begin.text.toString(),
                                        "End" to end.text.toString(),
                                        "CRS" to crs.text.toString(),
                                        "Status" to status,
                                        "Assigned" to assigned.text.toString(),
                                        "ID" to aircraft.text.toString() + "_" + check + "_"
+ day + month + year,
                                        "Comments" to comment.text.toString(),
                                        "Text" to "Yes",
                                        "Added" to added.toString(),
                                        "timestamp_begin" to
timestamp_begin.toString(),
                                        "timestamp_end" to timestamp_end,
                                        "timestamp_crs" to timestamp_crs,
                                        "timestamp_current" to
timestamp_current.toString())
                                db.collection("Aircraft revision")
```

```
                                    .document(aircraft.text.toString() + "_" + check +
"_" + day + month + year)
                                    .set(revisionInfo)
                              showSuccess()

                          }
                      }
                      builder.setNegativeButton(
                            "Cancel"
                      ) { dialog, which -> }
                      builder.create().show()
                  } else if (spinner2.selectedItem.toString() != "-" ||
                        end.text.toString().isNotEmpty() ||
                        crs.text.toString().isNotEmpty() ||
assigned.text.toString().isNotEmpty()) {

                      if (comment.text.toString() == "") {
                          val revisionInfo = hashMapOf(
                                "Aircraft" to aircraft.text.toString(),
                                "Check" to check,
                                "Start" to begin.text.toString(),
                                "End" to end.text.toString(),
                                "CRS" to crs.text.toString(),
                                "Status" to status,
                                "Assigned" to assigned.text.toString(),
                                "ID" to aircraft.text.toString() + "_" + check + "_" +
day + month + year,
                                "Comments" to comment.text.toString(),
                                "Text" to "No",
                                "Added" to added.toString(),
                                "timestamp_begin" to timestamp_begin.toString(),
                                "timestamp_end" to timestamp_end,
                                "timestamp_crs" to timestamp_crs,
                                "timestamp_current" to
timestamp_current.toString())
                          db.collection("Aircraft revision")
                                .document(aircraft.text.toString() + "_" + check +
"_" + day + month + year)
                                .set(revisionInfo)
                          showSuccess()

                      } else  {
                          val revisionInfo = hashMapOf(
                                "Aircraft" to aircraft.text.toString(),
                                "Check" to check,
                                "Start" to begin.text.toString(),
                                "End" to end.text.toString(),
                                "CRS" to crs.text.toString(),
                                "Status" to status,
                                "Assigned" to assigned.text.toString(),
                                      "ID" to aircraft.text.toString() + "_" + check + "_" +
day + month + year,
                                      "Comments" to comment.text.toString(),
                                      "Text" to "Yes",
                                      "Added" to added.toString(),
                                      "timestamp_begin" to timestamp_begin.toString(),
                                      "timestamp_end" to timestamp_end,
                                      "timestamp_crs" to timestamp_crs,
                                      "timestamp_current" to
timestamp_current.toString())
                               db.collection("Aircraft revision")
                                      .document(aircraft.text.toString() + "_" + check +
"_" + day + month + year)
                                      .set(revisionInfo)
                               showSuccess()

                           }
                       }
                   }
               }
           }
       }
   }
}

   private fun deletePreference() {

       val SHARED_PREF_START = "start";
       val sp = getSharedPreferences(SHARED_PREF_START,
MODE_PRIVATE)
       sp.edit().clear().apply()

       val SHARED_PREF_END = "END";
       val sp1 = getSharedPreferences(SHARED_PREF_END,
MODE_PRIVATE)
       sp1.edit().clear().apply()

       val SHARED_PREF_CRS = "CRS";
       val sp2 = getSharedPreferences(SHARED_PREF_CRS,
MODE_PRIVATE)
       sp2.edit().clear().apply()

       val SHARED_PREF_check = "CHECK"
       val sp3 = getSharedPreferences(SHARED_PREF_check,
MODE_PRIVATE)
       sp3.edit().clear().apply()

       val SHARED_PREF_status = "STATUS"
       val sp4 = getSharedPreferences(SHARED_PREF_status,
MODE_PRIVATE)
```

```
        sp4.edit().clear().apply()

        val SHARED_PREF_COM = "COM"
        val sp5 = getSharedPreferences(SHARED_PREF_COM,
MODE_PRIVATE)
        sp5.edit().clear().apply()

    }

    private fun displayEditText() {

        val SHARED_PREF_START = "start";
        val KEY_START = "key_start";
        val sp = getSharedPreferences(SHARED_PREF_START,
MODE_PRIVATE)
        val start = sp.getString(KEY_START, null)
        val editText1 = findViewById<TextView>(R.id.inputBegin)
        if (start != null) {
            editText1.text = "$start"
        }

        val SHARED_PREF_END = "END";
        val KEY_END = "END";
        val sp1 = getSharedPreferences(SHARED_PREF_END,
MODE_PRIVATE)
        val END = sp1.getString(KEY_END, null)
        val editText2 = findViewById<TextView>(R.id.inputEnd)
        if (END  != null) {
            editText2.text = "$END"
        }

        val SHARED_PREF_CRS = "CRS";
        val KEY_CRS = "CRS";
        val sp3 = getSharedPreferences(SHARED_PREF_CRS,
MODE_PRIVATE)
        val CRS = sp3.getString(KEY_CRS, null)
        val editText3 = findViewById<TextView>(R.id.inputCRS)
        if (CRS  != null) {
            editText3.text = "$CRS"
        }

        val SHARED_PREF_COM = "COM";
        val KEY_COM = "COM";
        val editText4 = findViewById<TextView>(R.id.inputComment)
        val sp4 = getSharedPreferences(SHARED_PREF_COM,
MODE_PRIVATE)
        val COM = sp4.getString(KEY_COM, null)
        if (COM  != null) {
            editText4.text = "$COM"
        }
        val SHARED_PREF_STATUS = "STATUS";
        val KEY_STATUS= "key_STATUS";
        val myPrefs = getSharedPreferences(SHARED_PREF_STATUS,
MODE_PRIVATE)
        val myString = myPrefs.getString(KEY_STATUS, null)
        val mSpinner = findViewById<Spinner>(R.id.spinner_status)
        setSpinText(mSpinner, myString)

        val SHARED_PREF_CHECK = "CHECK";
        val KEY_CHECK= "key_CHECK";
        val myPrefs1 = getSharedPreferences(SHARED_PREF_CHECK,
MODE_PRIVATE)
        val myString1 = myPrefs1.getString(KEY_CHECK, null)
        val mSpinner1 = findViewById<Spinner>(R.id.spinner_check)
        setSpinText(mSpinner1, myString1)
    }

    private fun saveEditText() {

        val SHARED_PREF_START = "start";
        val KEY_START = "key_start";
        val editText = findViewById<TextView>(R.id.inputBegin)
        val start: String = editText.text.toString()
        val sp = getSharedPreferences(SHARED_PREF_START,
MODE_PRIVATE)
        val editor = sp.edit()
        editor.putString(KEY_START, start)
        editor.apply()

        val SHARED_PREF_END = "END";
        val KEY_END = "END";
        val editText1 = findViewById<TextView>(R.id.inputEnd)
        val end : String = editText1.text.toString()
        val sp1 = getSharedPreferences(SHARED_PREF_END,
MODE_PRIVATE)
        val editor1 = sp1.edit()
        editor1.putString(KEY_END, end)
        editor1.apply()

        val SHARED_PREF_CRS = "CRS";
        val KEY_CRS = "CRS";
        val editText2 = findViewById<TextView>(R.id.inputCRS)
        val crs : String = editText2.text.toString()
        val sp2 = getSharedPreferences(SHARED_PREF_CRS,
MODE_PRIVATE)
        val editor2 = sp2.edit()
        editor2.putString(KEY_CRS, crs)
        editor2.apply()
        val SHARED_PREF_COM = "COM";
        val KEY_COM = "COM";
        val editText3 = findViewById<TextView>(R.id.inputComment)
        val com : String = editText3.text.toString()
        val sp3 = getSharedPreferences(SHARED_PREF_COM,
MODE_PRIVATE)
        val editor3 = sp3.edit()
        editor3.putString(KEY_COM, com)
        editor3.apply()

    }

    private fun reset() {
        val aircraft: EditText = findViewById(R.id.inputAircraft)
        val begin: EditText = findViewById(R.id.inputBegin)
        val end = findViewById<EditText>(R.id.inputEnd)
        val crs: EditText = findViewById(R.id.inputCRS)
        val assigned: EditText = findViewById(R.id.inputAssign)
        val comment : EditText = findViewById(R.id.inputComment)

        val spinner = findViewById<View>(R.id.spinner_check) as Spinner
        val spinner2 = findViewById<View>(R.id.spinner_status) as Spinner
        aircraft.text.clear()
        aircraft.hint = "-"
        begin.text.clear()
        begin.hint = "-"
        end.text.clear()
        end.hint = "-"
        crs.text.clear()
        crs.hint = "-"
        assigned.text.clear()
        assigned.hint="-"
        spinner.setSelection(0)
        spinner2.setSelection(0)
        comment.text.clear()
        comment.hint = ""
    }


    private fun openActivity1() {
        val intent = Intent(this, CheckListActivity::class.java)
        startActivity(intent)
        val SHARED_PREF_START = "start";
        val sp = getSharedPreferences(SHARED_PREF_START,
MODE_PRIVATE)
        sp.edit().clear().apply()
    }

    private fun showSuccess() {
        val builder = AlertDialog.Builder(this)
        builder.setTitle("Information")
        builder.setMessage("Check added successfully.")
        builder.setPositiveButton("Accept") { dialog, which ->
            reset()
            val i = Intent(this@AddCheckActivity, CheckListActivity::class.java)
            startActivity(i)
            finish()
        }
        val dialog: AlertDialog = builder.create()
        dialog.show()
    }

}
```

A.2. Add damage activity

```
class AddDamageActivity : AppCompatActivity() {

    private val PICK_IMAGE = 100
    private var imageUri: Uri? = null
    private var uriFilePath: Uri? = null
    private val GALLERY = 1
    private  var CAMERA:Int = 2
    var timestamp_entry : String? = null
    var timestamp_closing : String? = null

    override fun onCreate(savedInstanceState: Bundle?) {

        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_add_damages)

        val title: TextView = findViewById(R.id.tool_title)
        title.text = "Add new damage"

        val btn_menu: ImageButton = findViewById(R.id.btn_menu)
        btn_menu.visibility = View.GONE

        val location = findViewById<EditText>(R.id.inputLocation)
        val dimen1 = findViewById<EditText>(R.id.input_dimen_1)
        val dimen2 = findViewById<EditText>(R.id.input_dimen_2)
        val dimen3 = findViewById<EditText>(R.id.input_dimen_3)
        val comment = findViewById<EditText>(R.id.inputComment)
        val ref: EditText = findViewById(R.id.input_ref)
        val sn: EditText = findViewById(R.id.input_SN)
        val pn: EditText = findViewById(R.id.input_PN)
        val fh: EditText = findViewById(R.id.input_FH)
        val fc: EditText = findViewById(R.id.input_FC)
        val days: EditText = findViewById(R.id.input_Days)
        val fh2: EditText = findViewById(R.id.input_FH_repair)
        val fc2: EditText = findViewById(R.id.input_FC_repair)
        val days2: EditText = findViewById(R.id.input_Days_repair)
        val dimen = resources.getStringArray(R.array.dimen_array)
        val spinner: Spinner = findViewById(R.id.spinner_dimen)
        val adapter = ArrayAdapter(this, R.layout.spinner_item_selected2, dimen)
        val type = resources.getStringArray(R.array.type_array)
        val spinner_type: Spinner = findViewById(R.id.spinner_Type)
        val adapter_type = ArrayAdapter(this, R.layout.spinner_item_selected,
type)

        ref.onFocusChangeListener = OnFocusChangeListener { b, hasFocus ->
            if (!hasFocus) {
                ref.setHint(R.string.NoHint); hideKeyboard(b)
            } else ref.hint = ""
        }

        comment.onFocusChangeListener = OnFocusChangeListener { b,
hasFocus ->
            if (!hasFocus) {
                comment.setHint(R.string.NoHint); hideKeyboard(b)
            } else comment.hint = ""
        }

        sn.onFocusChangeListener = OnFocusChangeListener { b, hasFocus ->
            if (!hasFocus) {
                sn.setHint(R.string.NoHint); hideKeyboard(b)
            } else sn.hint = ""
        }

        pn.onFocusChangeListener = OnFocusChangeListener { b, hasFocus ->
            if (!hasFocus) {
                pn.setHint(R.string.NoHint); hideKeyboard(b)
            } else pn.hint = ""
        }

        location.onFocusChangeListener = OnFocusChangeListener { b,
hasFocus ->
            if (!hasFocus) {
                location.setHint(R.string.NoHint); hideKeyboard(b)
            } else location.hint = ""
        }

        dimen1.onFocusChangeListener = OnFocusChangeListener { b, hasFocus
->
            if (!hasFocus) {
                dimen1.setHint(R.string.NoHint); hideKeyboard(b)
            } else dimen1.hint = ""
        }

        dimen2.onFocusChangeListener = OnFocusChangeListener { b, hasFocus
->
         if (!hasFocus) {
            dimen2.setHint(R.string.NoHint); hideKeyboard(b)
         } else dimen2.hint = ""
      }

      dimen3.onFocusChangeListener = View.OnFocusChangeListener { b,
hasFocus ->
         if (!hasFocus) {
            dimen3.setHint(R.string.NoHint); hideKeyboard(b)
         } else dimen3.hint = ""
      }


      fh.onFocusChangeListener = View.OnFocusChangeListener { b, hasFocus
->
         if (!hasFocus) {
            fh.setHint(R.string.NoHint)
            hideKeyboard(b)
         } else fh.hint = ""
      }
      fc.onFocusChangeListener = View.OnFocusChangeListener { b, hasFocus
->
         if (!hasFocus) {
            fc.setHint(R.string.NoHint)
            hideKeyboard(b)
         } else fc.hint = ""
      }
      days.onFocusChangeListener = View.OnFocusChangeListener { b,
hasFocus ->
         if (!hasFocus) {
            days.setHint(R.string.NoHint)
            hideKeyboard(b)
         } else days.hint = ""
      }

      fh2.onFocusChangeListener = View.OnFocusChangeListener { b,
hasFocus ->
         if (!hasFocus) {
            fh2.setHint(R.string.NoHint)
            hideKeyboard(b)
         } else fh2.hint = ""
      }
      fc2.onFocusChangeListener = View.OnFocusChangeListener { b,
hasFocus ->
         if (!hasFocus) {
            fc2.setHint(R.string.NoHint)
            hideKeyboard(b)
         } else fc2.hint = ""
      }
```

```
        days2.onFocusChangeListener = View.OnFocusChangeListener { b,
hasFocus ->
            if (!hasFocus) {
                days2.setHint(R.string.NoHint)
                hideKeyboard(b)
            } else days2.hint = ""
        }

        adapter.setDropDownViewResource(R.layout.spinner_dropdown_item)
        spinner.adapter = adapter
        spinner.avoidDropdownFocus()
        spinner.onItemSelectedListener = object : OnItemSelectedListener {
            override fun onItemSelected(parent: AdapterView<*>, view: View?,
position: Int, id: Long) {
                if (parent.getItemAtPosition(position) == "-") {
                } else {
                    val SHARED_PREF_DIMEN = "DIMEN";
                    val KEY_DIMEN = "key_DIMEN";
                    val mSpnValue: String = spinner.selectedItem.toString()
                    val myPrefs: SharedPreferences =
getSharedPreferences(SHARED_PREF_DIMEN, MODE_PRIVATE)
                    val prefsEditor = myPrefs.edit()
                    prefsEditor.putString(KEY_DIMEN, mSpnValue)
                    prefsEditor.apply()
                }
            }

            override fun onNothingSelected(parent: AdapterView<*>?) {}
        }


adapter_type.setDropDownViewResource(R.layout.spinner_dropdown_item)
        spinner_type.adapter = adapter_type
        spinner_type.avoidDropdownFocus()
        spinner_type.onItemSelectedListener = object : OnItemSelectedListener {
            override fun onItemSelected(parent: AdapterView<*>, view: View?,
position: Int, id: Long
            ) {
                if (parent.getItemAtPosition(position) == "-") {
                }
            }

            override fun onNothingSelected(parent: AdapterView<*>?) {}
        }

        val edit2 = findViewById<View>(R.id.input_closing) as EditText
        val calendar2 = Calendar.getInstance()
        val yy2 = calendar2.get(Calendar.YEAR)
        val mm2 = calendar2.get(Calendar.MONTH)
        val dd2 = calendar2.get(Calendar.DAY_OF_MONTH)
        val datePicker2 = DatePickerDialog(
                this@AddDamageActivity,
                { _, year, monthOfYear, dayOfMonth ->
                    val date2 =
                        (dayOfMonth.toString() + "/" + (monthOfYear + 1).toString() +
"/" + year.toString())
                    edit2.setText(date2)
                }, yy2, mm2, dd2
        )
        datePicker2.setButton(DialogInterface.BUTTON_NEUTRAL, "CLEAR") {
_, which ->
            if (which == DialogInterface.BUTTON_NEUTRAL) {
                edit2.setText("-")
            }
        }

        edit2.setOnClickListener() {
            datePicker2.show()
        }

        val cat = resources.getStringArray(R.array.cat_array)
        val spinner1: Spinner = findViewById(R.id.spinner_cat)
        val adapter1 = ArrayAdapter(this, R.layout.spinner_item_selected, cat)
        adapter1.setDropDownViewResource(R.layout.spinner_dropdown_item)
        spinner1.adapter = adapter1
        spinner1.avoidDropdownFocus()
        spinner1.onItemSelectedListener = object : OnItemSelectedListener {
            override fun onItemSelected(parent: AdapterView<*>, view: View?,
position: Int, id: Long
            ) {
                if (parent.getItemAtPosition(position).toString() == "B"
                        || parent.getItemAtPosition(position).toString() == "C") {
                    edit2.setBackgroundResource(R.color.black_overlay)
                    edit2.setOnClickListener {
                        datePicker2.hide()
                    }
                    if (parent.getItemAtPosition(position).toString() == "B") {

                        fh.isEnabled = true
                        fh.setBackgroundResource(R.drawable.custom_input)
                        fc.isEnabled = true
                        fc.setBackgroundResource(R.drawable.custom_input)
                        days.isEnabled = true
                        days.setBackgroundResource(R.drawable.custom_input)

                        fh2.isEnabled = false
                        fh2.setBackgroundResource(R.color.black_overlay)
                        fc2.isEnabled = false
                        fc2.setBackgroundResource(R.color.black_overlay)
                        days2.isEnabled = false
                        days2.setBackgroundResource(R.color.black_overlay)
                    }

                    if (parent.getItemAtPosition(position).toString() == "C") {

                        fh.isEnabled = true
                        fh.setBackgroundResource(R.drawable.custom_input)
                        fc.isEnabled = true
                        fc.setBackgroundResource(R.drawable.custom_input)
                        days.isEnabled = true
                        days.setBackgroundResource(R.drawable.custom_input)

                        fh2.isEnabled = true
                        fh2.setBackgroundResource(R.drawable.custom_input)
                        fc2.isEnabled = true
                        fc2.setBackgroundResource(R.drawable.custom_input)
                        days2.isEnabled = true
                        days2.setBackgroundResource(R.drawable.custom_input)
                    }

                }

                if (parent.getItemAtPosition(position).toString() == "A") {
                    edit2.setBackgroundResource(R.drawable.custom_input)
                    edit2.setOnClickListener {
                        datePicker2.show()
                    }

                    fh.isEnabled = false
                    fh.setBackgroundResource(R.color.black_overlay)
                    fc.isEnabled = false
                    fc.setBackgroundResource(R.color.black_overlay)
                    days.isEnabled = false
                    days.setBackgroundResource(R.color.black_overlay)

                    fh2.isEnabled = false
                    fh2.setBackgroundResource(R.color.black_overlay)
                    fc2.isEnabled = false
                    fc2.setBackgroundResource(R.color.black_overlay)
                    days2.isEnabled = false
                    days2.setBackgroundResource(R.color.black_overlay)


                }
            }

            override fun onNothingSelected(parent: AdapterView<*>?) {}
        }

        val calendar = Calendar.getInstance()
```

```
        val edit = findViewById<View>(R.id.input_entry) as EditText
        val yy = calendar.get(Calendar.YEAR)
        val mm = calendar.get(Calendar.MONTH)
        val dd = calendar.get(Calendar.DAY_OF_MONTH)
        val datePicker = DatePickerDialog(
            this@AddDamageActivity,
            { _, year, monthOfYear, dayOfMonth ->
                val date =
                    (dayOfMonth.toString() + "/" + (monthOfYear + 1).toString() +
"/" + year.toString())
                edit.setText(date)
            },
            yy,
            mm,
            dd
        )
        datePicker.setButton(DialogInterface.BUTTON_NEUTRAL, "CLEAR") { _,
which ->
            if (which == DialogInterface.BUTTON_NEUTRAL) {
                edit.setText("-")
            }
        }

        edit.setOnClickListener {
            datePicker.show()
        }

        val add: Button = findViewById(R.id.btn_Add)
        add.setOnClickListener {

            if (check()) {

                val inflater3 = LayoutInflater.from(this)
                val popupView3: View = inflater3.inflate(R.layout.pop_window_pin,
null)
                val focusable3 = true
                val popupWindow3 = PopupWindow(popupView3, 900,
ViewGroup.LayoutParams.WRAP_CONTENT, focusable3)
                popupWindow3.showAtLocation(popupView3, Gravity.CENTER, 0, 0)


popupWindow3.contentView.findViewById<TextView>(R.id.tool_title).text =
"Write PIN"

popupWindow3.contentView.findViewById<ImageButton>(R.id.btn_menu).visibil
ity = View.GONE

popupWindow3.contentView.findViewById<ImageButton>(R.id.btn_back).visibili
ty = View.GONE
```

```
            val db = FirebaseFirestore.getInstance()
            val user = FirebaseAuth.getInstance().currentUser
            val userEmail = user.email
            val df = db.collection("Users").document(userEmail.toString())
            df.get().addOnSuccessListener { documentSnapshot ->
               val txtpin =
popupWindow3.contentView.findViewById<TextView>(R.id.inputPIN)
               val pin: Button =
popupWindow3.contentView.findViewById(R.id.pop_pin)
               pin.setOnClickListener {

                  if (documentSnapshot.getString("PIN") == txtpin.text.toString()) {
                     if (dimen1.text.toString().isEmpty()) {
                        dimen1.setText("_")
                     }
                     if (dimen2.text.toString().isEmpty()) {
                        dimen2.setText("_")
                     }
                     if (dimen3.text.toString().isEmpty()) {
                        dimen3.setText("_")
                     }
                     saveDamage()

                  } else {
                     val builder1 = AlertDialog.Builder(this)
                     builder1.setTitle("Error")
                     builder1.setMessage("Incorrect PIN, please try again.")
                     builder1.setPositiveButton("Accept", null)
                     val dialog1: AlertDialog = builder1.create()
                     dialog1.show()
                  }
               }
            }
         }
      }

      val upload = findViewById<View>(R.id.button_upload) as Button
      upload.setOnClickListener {
         openGallery()
      }

      val take = findViewById<View>(R.id.button_photo) as Button
      take.setOnClickListener {
         takePhotoFromCamera()
      }

      val imageView: ImageView = findViewById(R.id.imageView2)
      val button2: Button = findViewById(R.id.button_clear)
      button2.setOnClickListener {
         imageView.setImageDrawable(null)
```

```
        }

        val back: ImageButton = findViewById(R.id.btn_back)
        back.setOnClickListener {
            val intent = Intent(this, DamageActivity::class.java)
            startActivity(intent)
            finish()
        }

        displayInfo()
        ata()

    }

    private fun takePhotoFromCamera() {
        val intent = Intent(MediaStore.ACTION_IMAGE_CAPTURE)
        startActivityForResult(intent, CAMERA)
    }

    private fun openGallery() {
        val gallery = Intent(Intent.ACTION_PICK,
MediaStore.Images.Media.INTERNAL_CONTENT_URI)
        startActivityForResult(gallery, PICK_IMAGE)
    }

    override fun onActivityResult(requestCode: Int, resultCode: Int, data:
Intent?) {
        val imageView = findViewById<ImageView>(R.id.imageView2)
        super.onActivityResult(requestCode, resultCode, data)
        if (resultCode == RESULT_OK && requestCode == PICK_IMAGE) {
            imageUri = data?.data
            imageView!!.setImageURI(imageUri)
        }
        else if (requestCode == CAMERA) {
            imageUri = data?.data
            imageView!!.setImageURI(imageUri)
        }
    }

    private fun check(): Boolean {
        val cat : Spinner = findViewById(R.id.spinner_cat)
        val type : Spinner = findViewById(R.id.spinner_Type)
        val loc : EditText = findViewById(R.id.inputLocation)
        val entry : EditText = findViewById(R.id.input_entry)

        if (cat.selectedItem.toString() == "-" ||
                entry.text.toString().isEmpty()
                || type.selectedItem.toString() == "-"
                || loc.text.toString().isEmpty())
        {
            val builder = AlertDialog.Builder(this)
            builder.setTitle("Error")
            builder.setMessage("Damage category, type, location and entry date
must be filled.")
            builder.setPositiveButton("Accept", null)
            val dialog: AlertDialog = builder.create()
            dialog.show()
            return false
        } else {
            return true
        }
    }

    private fun ata() {
        val editText = findViewById<TextView>(R.id.inputATA)
        val rel: RelativeLayout = findViewById(R.id.form_layout)
        editText.setOnClickListener {
            val inflater = LayoutInflater.from(this)
            val popupView: View = inflater.inflate(R.layout.ata_list, null)
            val focusable = true
            val popupWindow = PopupWindow(popupView, 1000, 1000, focusable)
            popupWindow.showAtLocation(rel, Gravity.CENTER, 0, 0)


popupWindow.contentView.findViewById<RadioGroup>(R.id.radioATA).setOnC
heckedChangeListener { group, checkedId ->
                val radioButton = group.findViewById<View>(checkedId) as
RadioButton
                val text = radioButton.text.toString()
                val SHARED_PREF_ATA = "ATA";
                val KEY_ATA = "key_ATA";
                val sp = getSharedPreferences(SHARED_PREF_ATA,
MODE_PRIVATE)
                val editor = sp.edit()
                editor.putString(KEY_ATA, text)
                editor.apply()
                editText.text = text
                sp.edit().clear().apply()
            }
        }
    }

    private fun setSpinText(spin: Spinner, text: String?) {
        for (i in 0 until spin.adapter.count) {
            if (spin.adapter.getItem(i).toString().contains(text.toString())) {
                spin.setSelection(i)
            }
        }
    }
    private fun Spinner.avoidDropdownFocus() {
        try {
            val isAppCompat = this is
androidx.appcompat.widget.AppCompatSpinner
            val spinnerClass = if (isAppCompat)
androidx.appcompat.widget.AppCompatSpinner::class.java else
Spinner::class.java
            val popupWindowClass = if (isAppCompat)
androidx.appcompat.widget.ListPopupWindow::class.java else
android.widget.ListPopupWindow::class.java

            val listPopup = spinnerClass
                .getDeclaredField("mPopup")
                .apply { isAccessible = true }
                .get(this)
            if (popupWindowClass.isInstance(listPopup)) {
                val popup = popupWindowClass
                    .getDeclaredField("mPopup")
                    .apply { isAccessible = true }
                    .get(listPopup)
                if (popup is PopupWindow) {
                    popup.isFocusable = false
                }
            }
        } catch (e: Exception) {
            e.printStackTrace()
        }
    }

    private fun hideKeyboard(view: View) {
        val inputMethodManager =
getSystemService(INPUT_METHOD_SERVICE) as InputMethodManager
        inputMethodManager.hideSoftInputFromWindow(view.windowToken, 0)
    }

    private fun displayInfo() {

        val SHARED_PREF_AIRCRAFT = "aircraft"
        val KEY_AIRCRAFT = "key_aircraft"
        val sp = getSharedPreferences(SHARED_PREF_AIRCRAFT,
MODE_PRIVATE)
        val aircraft = sp.getString(KEY_AIRCRAFT, null)
        val editText: TextView = findViewById(R.id.inputAircraft)
        if (aircraft != null) {
            editText.text = "$aircraft"
        }
    }

    private fun hasNullOrEmptyDrawable(iv: ImageView?): Boolean {
        val drawable = iv!!.drawable
        val bitmapDrawable = if (drawable is BitmapDrawable) drawable else null

        return bitmapDrawable == null || bitmapDrawable.bitmap == null
    }

    private fun saveDamage() {

        val SHARED_PREF_AIRCRAFT = "aircraft"
        val KEY_AIRCRAFT = "key_aircraft"
        val sp = getSharedPreferences(SHARED_PREF_AIRCRAFT,
MODE_PRIVATE)
        val aircraft = sp.getString(KEY_AIRCRAFT, null)

        val SHARED_PREF_START = "start"
        val KEY_START = "key_start"
        val sp1 = getSharedPreferences(SHARED_PREF_START,
MODE_PRIVATE)
        val start = sp1.getString(KEY_START, null)

        val SHARED_PREF_check = "check"
        val KEY_check = "key_check"
        val sp2 = getSharedPreferences(SHARED_PREF_check,
MODE_PRIVATE)
        val check = sp2.getString(KEY_check, null)

        val items1 = start?.split("/".toRegex())?.toTypedArray()
        val day = items1?.get(0)
        val month = items1?.get(1)
        val year = items1?.get(2)

        val currentDate: String = SimpleDateFormat("d-M-yy",
Locale.getDefault()).format(Date())
        val currentTime = SimpleDateFormat("HH:mm:ss",
Locale.getDefault()).format(Date())

        val items2 = currentDate.split("-".toRegex()).toTypedArray()
        val d = items2[0]
        val m = items2[1]
        val y = items2[2]

        val items3 = currentTime.split(":".toRegex()).toTypedArray()
        val hour = items3[0]
        val min = items3[1]

        val spinner = findViewById<View>(R.id.spinner_cat) as Spinner
        val cat = spinner.selectedItem.toString()
        val spinner2: Spinner = findViewById(R.id.spinner_dimen)
        val dimen = spinner2.selectedItem.toString()
        val spinner3 = findViewById<View>(R.id.spinner_Type) as Spinner
        val type = spinner3.selectedItem.toString()
        val ata = findViewById<EditText>(R.id.inputATA)
        val location = findViewById<TextView>(R.id.inputLocation)
        val dimen1: TextView = findViewById(R.id.input_dimen_1)
        val dimen2: TextView = findViewById(R.id.input_dimen_2)
        val dimen3: TextView = findViewById(R.id.input_dimen_3)
        val fh: EditText = findViewById(R.id.input_FH)
        val fc: EditText = findViewById(R.id.input_FC)
        val days: EditText = findViewById(R.id.input_Days)
        val fh2: EditText = findViewById(R.id.input_FH_repair)
        val fc2: EditText = findViewById(R.id.input_FC_repair)
        val days2: EditText = findViewById(R.id.input_Days_repair)
        val entry: EditText = findViewById(R.id.input_entry)
        val closing: EditText = findViewById(R.id.input_closing)
        val pn: EditText = findViewById(R.id.input_PN)
        val sn: EditText = findViewById(R.id.input_SN)
        val ref: EditText = findViewById(R.id.input_ref)
        val comment: EditText = findViewById(R.id.inputComment)
        val regis: EditText = findViewById(R.id.inputAircraft)

        val damage_id = d + m + y + "_" + hour + min
        val fileRef =
FirebaseStorage.getInstance().reference.child("damage/$damage_id")
        val imageView = findViewById<ImageView>(R.id.imageView2)

        if (entry.text.toString().isEmpty()){
            timestamp_entry = "0"
        } else {
            val formatter: DateFormat = SimpleDateFormat("MM-dd-yyyy")
            val date_entry = entry.text.toString()
            val items2 = date_entry.split("/".toRegex()).toTypedArray()
            val day2 = items2[0]
            val month2 = items2[1]
            val year2 = items2[2]
            val time2 = "$month2-$day2-$year2"
            val date_convert_entry = formatter.parse(time2) as Date
            val output_entry = date_convert_entry.time / 1000L
            val str_entry = output_entry.toString()
            timestamp_entry = ((str_entry.toLong() * 1000).toString())
        }

        if (closing.text.toString().isEmpty()){
            timestamp_closing = "0"
        } else {
            val formatter2: DateFormat = SimpleDateFormat("MM-dd-yyyy")
            val date_closing = closing.text.toString()
            val items3 = date_closing.split("/".toRegex()).toTypedArray()
            val day3 = items3[0]
            val month3 = items3[1]
            val year3 = items3[2]
            val time3 = "$month3-$day3-$year3"
            val date_convert_closing = formatter2.parse(time3) as Date
            val output_closing = date_convert_closing.time / 1000L
            val str_closing = output_closing.toString()
            timestamp_closing = ((str_closing.toLong() * 1000).toString())
        }

        if (hasNullOrEmptyDrawable(imageView)) {

        } else {
            val bitmap = (imageView.drawable as BitmapDrawable).bitmap
            imageView.isDrawingCacheEnabled = true
            imageView.buildDrawingCache()
            val baos = ByteArrayOutputStream()
            bitmap.compress(Bitmap.CompressFormat.JPEG, 100, baos)
            val data = baos.toByteArray()
            fileRef.putBytes(data)
        }

        if (comment.text.toString().isEmpty()) {
            val id = d + m + y + "_" + hour + min
            val db = FirebaseFirestore.getInstance()
            val df = db.collection("Aircraft revision").document(aircraft + "_" +
check + "_" + day + month + year).collection("New damages")
            if (cat == "B") {
                showSuccess()
                val revisionInfo = hashMapOf(
                        "Aircraft" to regis.text.toString(),
                        "Inspection FH" to fh.text.toString(),
                        "Inspection FC" to fc.text.toString(),
                        "Inspection days" to days.text.toString(),
                        "Repair FH" to "-",
                        "Repair FC" to "-",
                        "Repair days" to "-",
                        "Dim" to dimen,
                        "Dim1" to dimen1.text.toString(),
                        "Dim2" to dimen2.text.toString(),
                        "Dim3" to dimen3.text.toString(),
                        "Type" to type,
                        "References" to ref.text.toString(),
                        "Dimension" to dimen1.text.toString() + "x"
                                + dimen2.text.toString() + "x"
                                + dimen3.text.toString() + dimen,
                        "Description" to type + "_" + location.text.toString(),
                        "Entry" to entry.text.toString(),
                        "Closing" to "-",
                        "Comments" to comment.text.toString(),
                        "Commentary" to "No",
                        "PN" to pn.text.toString(),
                        "SN" to sn.text.toString(),
                        "Category" to cat,
                        "ATA" to ata.text.toString(),
                        "Location" to location.text.toString(),
                        "Dimension" to dimen1.text.toString() + "x"
                                + dimen2.text.toString() + "x"
                                + dimen3.text.toString() + dimen,
                        "ID" to d + m + y + "_" + hour + min,
                        "RB" to "Pending",
                        "Certified" to "No",
                        "Checked" to "No",
                        "Map" to "No",
                        "CertifyBY" to "-",
                        "CheckedBY" to "-",
                        "timestamp_entry" to timestamp_entry,
                        "timestamp_closing" to timestamp_closing
                )
                df.document(id).set(revisionInfo)
                showSuccess()
            }
            if (cat == "C") {
                val revisionInfo = hashMapOf(
                        "Aircraft" to regis.text.toString(),
                        "Inspection FH" to fh.text.toString(),
                        "Inspection FC" to fc.text.toString(),
                        "Inspection days" to days.text.toString(),
                        "Repair FH" to fh2.text.toString(),
                        "Dim" to dimen,
                        "Dim1" to dimen1.text.toString(),
                        "Dim2" to dimen2.text.toString(),
                        "Dim3" to dimen3.text.toString(),
                        "Repair FC" to fc2.text.toString(),
                        "Repair days" to days2.text.toString(),
                        "Entry" to entry.text.toString(),
                        "References" to ref.text.toString(),
                        "Closing" to "-",
                        "Type" to type,
                        "Description" to type + "_" + location.text.toString(),
                        "Comments" to comment.text.toString(),
                        "Commentary" to "No",
                        "PN" to pn.text.toString(),
                        "SN" to sn.text.toString(),
                        "Category" to cat,
                        "ATA" to ata.text.toString(),
                        "Location" to location.text.toString(),
                        "Dimension" to dimen1.text.toString() + "x"
                                + dimen2.text.toString() + "x"
                                + dimen3.text.toString() + dimen,
                        "ID" to d + m + y + "_" + hour + min,
                        "RB" to "Pending",
                        "Certified" to "No",
                        "Checked" to "No",
                        "Map" to "No",
                        "CertifyBY" to "-",
                        "CheckedBY" to "-",
                        "timestamp_entry" to timestamp_entry,
                        "timestamp_closing" to timestamp_closing
                )
                df.document(id).set(revisionInfo)
                showSuccess()
            }
            if (cat == "A") {
                val revisionInfo = hashMapOf(
                        "Aircraft" to regis.text.toString(),
                        "Inspection FH" to "-",
                        "Inspection FC" to "-",
                        "Inspection days" to "-",
                        "Repair FH" to "-",
                        "Repair FC" to "-",
                        "Repair days" to "-",
                        "Dim" to dimen,
                        "Dim1" to dimen1.text.toString(),
                        "Dim2" to dimen2.text.toString(),
                        "Dim3" to dimen3.text.toString(),
                        "Type" to type,
                        "Description" to type + "_" + location.text.toString(),
                        "Comments" to comment.text.toString(),
                        "Commentary" to "No",
                        "Entry" to entry.text.toString(),
                        "Closing" to closing.text.toString(),
                        "References" to ref.text.toString(),
                        "PN" to pn.text.toString(),
                        "SN" to sn.text.toString(),
                        "Category" to cat,
                        "ATA" to ata.text.toString(),
                        "Location" to location.text.toString(),
                        "Dimension" to dimen1.text.toString() + "x"
                                + dimen2.text.toString() + "x"
                                + dimen3.text.toString() + dimen,
                        "ID" to d + m + y + "_" + hour + min,
                        "RB" to "N/A",
                        "Certified" to "No",
                        "Checked" to "No",
                        "Map" to "No",
                        "CertifyBY" to "-",
                        "CheckedBY" to "-",
                        "timestamp_entry" to timestamp_entry,
                        "timestamp_closing" to timestamp_closing
                )
                df.document(id).set(revisionInfo)
                showSuccess()
            }
```

```
        }
        if (comment.text.toString().isNotEmpty()) {
            val db = FirebaseFirestore.getInstance()
            val df = db.collection("Aircraft revision").document(aircraft + "_" +
check + "_" + day + month + year).collection("New damages")
            val id = d + m + y + "_" + hour + min
            if (cat == "B") {
                showSuccess()
                val revisionInfo = hashMapOf(
                        "Aircraft" to regis.text.toString(),
                        "Inspection FH" to fh.text.toString(),
                        "Inspection FC" to fc.text.toString(),
                        "Inspection days" to days.text.toString(),
                        "Repair FH" to "-",
                        "Repair FC" to "-",
                        "Repair days" to "-",
                        "Dim" to dimen,
                        "Dim1" to dimen1.text.toString(),
                        "Dim2" to dimen2.text.toString(),
                        "Dim3" to dimen3.text.toString(),
                        "Type" to type,
                        "References" to ref.text.toString(),
                        "Dimension" to dimen1.text.toString() + "x"
                                + dimen2.text.toString() + "x"
                                + dimen3.text.toString() + dimen,
                        "Description" to type + "_" + location.text.toString(),
                        "Entry" to entry.text.toString(),
                        "Closing" to "-",
                        "Comments" to comment.text.toString(),
                        "Commentary" to "Yes",
                        "PN" to pn.text.toString(),
                        "SN" to sn.text.toString(),
                        "Category" to cat,
                        "ATA" to ata.text.toString(),
                        "Location" to location.text.toString(),
                        "Dimension" to dimen1.text.toString() + "x"
                                + dimen2.text.toString() + "x"
                                + dimen3.text.toString() + dimen,
                        "ID" to d + m + y + "_" + hour + min,
                        "RB" to "Pending",
                        "Certified" to "No",
                        "Checked" to "No",
                        "Map" to "No",
                        "CertifyBY" to "-",
                        "CheckedBY" to "-",
                        "timestamp_entry" to timestamp_entry,
                        "timestamp_closing" to timestamp_closing
                )
                df.document(id).set(revisionInfo)
                showSuccess()
```

```
}
if (cat == "C") {
   val revisionInfo = hashMapOf(
         "Aircraft" to regis.text.toString(),
         "Inspection FH" to fh.text.toString(),
         "Inspection FC" to fc.text.toString(),
         "Inspection days" to days.text.toString(),
         "Repair FH" to fh2.text.toString(),
         "Dim" to dimen,
         "Dim1" to dimen1.text.toString(),
         "Dim2" to dimen2.text.toString(),
         "Dim3" to dimen3.text.toString(),
         "Repair FC" to fc2.text.toString(),
         "Repair days" to days2.text.toString(),
         "Entry" to entry.text.toString(),
         "References" to ref.text.toString(),
         "Closing" to "-",
         "Type" to type,
         "Description" to type + "_" + location.text.toString(),
         "Comments" to comment.text.toString(),
         "Commentary" to "Yes",
         "PN" to pn.text.toString(),
         "SN" to sn.text.toString(),
         "Category" to cat,
         "ATA" to ata.text.toString(),
         "Location" to location.text.toString(),
         "Dimension" to dimen1.text.toString() + "x"
               + dimen2.text.toString() + "x"
               + dimen3.text.toString() + dimen,
         "ID" to d + m + y + "_" + hour + min,
         "RB" to "Pending",
         "Certified" to "No",
         "Checked" to "No",
         "Map" to "No",
         "CertifyBY" to "-",
         "CheckedBY" to "-",
         "timestamp_entry" to timestamp_entry,
         "timestamp_closing" to timestamp_closing
   )
   df.document(id).set(revisionInfo)
   showSuccess()
}
if (cat == "A") {
   val revisionInfo = hashMapOf(
         "Aircraft" to regis.text.toString(),
         "Inspection FH" to "-",
         "Inspection FC" to "-",
         "Inspection days" to "-",
         "Repair FH" to "-",
         "Repair FC" to "-",
                "Repair days" to "-",
                "Dim" to dimen,
                "Dim1" to dimen1.text.toString(),
                "Dim2" to dimen2.text.toString(),
                "Dim3" to dimen3.text.toString(),
                "Type" to type,
                "Description" to type + "_" + location.text.toString(),
                "Comments" to comment.text.toString(),
                "Commentary" to "Yes",
                "Entry" to entry.text.toString(),
                "Closing" to closing.text.toString(),
                "References" to ref.text.toString(),
                "PN" to pn.text.toString(),
                "SN" to sn.text.toString(),
                "Category" to cat,
                "ATA" to ata.text.toString(),
                "Location" to location.text.toString(),
                "Dimension" to dimen1.text.toString() + "x"
                    + dimen2.text.toString() + "x"
                    + dimen3.text.toString() + dimen,
                "ID" to d + m + y + "_" + hour + min,
                "RB" to "N/A",
                "Certified" to "No",
                "Checked" to "No",
                "Map" to "No",
                "CertifyBY" to "-",
                "CheckedBY" to "-",
                "timestamp_entry" to timestamp_entry,
                "timestamp_closing" to timestamp_closing
            )
            df.document(id).set(revisionInfo)
            showSuccess()
        }
    }
}

private fun showSuccess() {
    val builder = AlertDialog.Builder(this)
    builder.setTitle("Information")
    builder.setMessage("Damage report added successfully.")
    builder.setPositiveButton("Accept") { dialog, which ->
        val intent = Intent(this, DamageActivity::class.java)
        startActivity(intent)
        finish()
    }
    val dialog: AlertDialog = builder.create()
    dialog.show()
}
}
```

A.3. Add user activity

```
class AddUserActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {

        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_adduser)

        val title : TextView = findViewById(R.id.tool_title)
        title.text = "Add user"

        val btn_menu : ImageButton = findViewById(R.id.btn_menu)
        btn_menu.visibility = View.GONE

        val name: EditText = findViewById(R.id.inputNameRegister)
        val phone: EditText = findViewById(R.id.inputPhone)
        val email: EditText = findViewById(R.id.inputEmailRegister)

        name.onFocusChangeListener = OnFocusChangeListener { b, hasFocus -
>
            if (!hasFocus) {
                name.setHint(R.string.NameHint)
                hideKeyboard(b)
            } else name.hint = ""
        }
        phone.onFocusChangeListener = OnFocusChangeListener { b, hasFocus
->
            if (!hasFocus) {
                phone.setHint(R.string.PhoneHint)
                hideKeyboard(b)
            } else phone.hint = ""
        }
        email.onFocusChangeListener = OnFocusChangeListener { a, hasFocus -
>
            if (!hasFocus) {
                email.setHint(R.string.EmailHint)
                hideKeyboard(a)
            } else email.hint = ""
        }


        val roles = resources.getStringArray(R.array.job_array)
        val spinner: Spinner = findViewById(R.id.spinner_edit_user)
        val adapter2 = ArrayAdapter(this, R.layout.spinner_item_selected, roles)
        adapter2.setDropDownViewResource(R.layout.spinner_dropdown_item)
        spinner.adapter = adapter2
        spinner.avoidDropdownFocus()
        spinner.onItemSelectedListener = object : OnItemSelectedListener {
            override fun onItemSelected(parent: AdapterView<*>, view: View?,
position: Int, id: Long) {
                if (parent.getItemAtPosition(position) == "Please choose a job role")
{
                }
            }
            override fun onNothingSelected(parent: AdapterView<*>?) {}
        }

        val permission = resources.getStringArray(R.array.permission_array)
        val spinner3: Spinner = findViewById(R.id.spinner_permission)
        val adapter3 = ArrayAdapter(this, R.layout.spinner_item_selected,
permission)
        adapter3.setDropDownViewResource(R.layout.spinner_dropdown_item)
        spinner3.adapter = adapter3
        spinner3.avoidDropdownFocus()
        spinner3.onItemSelectedListener = object : OnItemSelectedListener {
            override fun onItemSelected(parent: AdapterView<*>, view: View?,
position: Int, id: Long) {
                if (parent.getItemAtPosition(position) == "Permission") {
                }
            }
            override fun onNothingSelected(parent: AdapterView<*>?) {}
        }

        val back : ImageButton = findViewById(R.id.btn_back)
        back.setOnClickListener {
            val intent = Intent(this, UserActivity::class.java)
            startActivity(intent)
            finish()
        }

        val btn_add : Button = findViewById(R.id.btn_edit)
        btn_add.setOnClickListener {
            if (spinner3.selectedItem.toString() == "Permission") {
                val builder = AlertDialog.Builder(this)
                builder.setTitle("Error")
                builder.setMessage("Permission field must be 'Yes' or 'No'")
                builder.setPositiveButton("Accept") { dialog, which ->}
                val dialog: AlertDialog = builder.create()
                dialog.show()
            } else {
                saveUserData()
            }
        }

    }

    private fun hideKeyboard(view: View) {
        val inputMethodManager =
getSystemService(INPUT_METHOD_SERVICE) as InputMethodManager
```

```
        inputMethodManager.hideSoftInputFromWindow(view.windowToken, 0)
    }

    private fun Spinner.avoidDropdownFocus() {
        try {
            val isAppCompat = this is
androidx.appcompat.widget.AppCompatSpinner
            val spinnerClass = if (isAppCompat)
androidx.appcompat.widget.AppCompatSpinner::class.java else
Spinner::class.java
            val popupWindowClass = if (isAppCompat)
androidx.appcompat.widget.ListPopupWindow::class.java else
android.widget.ListPopupWindow::class.java

            val listPopup = spinnerClass
                    .getDeclaredField("mPopup")
                    .apply { isAccessible = true }
                    .get(this)
            if (popupWindowClass.isInstance(listPopup)) {
                val popup = popupWindowClass
                        .getDeclaredField("mPopup")
                        .apply { isAccessible = true }
                        .get(listPopup)
                if (popup is PopupWindow) {
                    popup.isFocusable = false
                }
            }
        } catch (e: Exception) {
            e.printStackTrace()
        }
    }

    private fun saveUserData() {

        val name: EditText = findViewById(R.id.inputNameRegister)
        val phone: EditText = findViewById(R.id.inputPhone)
        val email = findViewById<EditText>(R.id.inputEmailRegister)
        val password : EditText = findViewById(R.id.inputPasswordRegister)
        val spinner2 = findViewById<View>(R.id.spinner_permission) as Spinner
        val permission = spinner2.selectedItem.toString()
        val spinner = findViewById<View>(R.id.spinner_edit_user) as Spinner
        val job = spinner.selectedItem.toString()
        val pin : EditText = findViewById(R.id.inputPIN)
        val db = FirebaseFirestore.getInstance()

        val user = FirebaseAuth.getInstance().currentUser
        val userEmail = user.email

        if (checkName() and checkPhone() and checkEmail()
                and checkPassword() and checkConfirm() and checkJob() and
```

```
checkPIN()) {

            FirebaseAuth.getInstance().createUserWithEmailAndPassword(
                    email.text.toString(),
                    password.text.toString()
            ).addOnCompleteListener {
                if (it.isSuccessful) {

                    val builder = AlertDialog.Builder(this)
                    builder.setTitle("Information")
                    builder.setMessage("Account created successfully")
                    builder.setPositiveButton("Accept") { dialog, id ->

                        val userInfo = hashMapOf(
                                "Name" to name.text.toString(),
                                "Phone" to phone.text.toString(),
                                "Email" to email.text.toString(),
                                "PIN" to pin.text.toString(),
                                "Job" to job,
                                "Permission" to permission)

db.collection("Users").document(email.text.toString()).set(userInfo)

                        val it = Intent(Intent.ACTION_SEND)
                        it.putExtra(Intent.EXTRA_EMAIL, arrayOf(email.text.toString()))
                        it.putExtra(Intent.EXTRA_SUBJECT, "AirDocs account data")
                        it.putExtra(Intent.EXTRA_TEXT, "Hi $name, Welcome to
AirDocs. Your account data is the following: " +
                                "Name: $name" +
                                "Phone: $phone" +
                                "Email: $email" +
                                "Signature PIN: $pin"+
                                "Job role: $job" +
                                "Permission: $permission")
                        it.type = "message/rfc822"

                        val SHARED_PREF_PASSWORD = "password"
                        val KEY_PASSWORD = "key_password";
                        val sp = getSharedPreferences(SHARED_PREF_PASSWORD,
MODE_PRIVATE)
                        val pass = sp.getString(KEY_PASSWORD, null)

                        Log.d("myTag", "$pass")


FirebaseAuth.getInstance().signInWithEmailAndPassword(userEmail.toString(),
                                pass.toString())

                        val intent = Intent(this, UserActivity::class.java)
                        startActivity(intent)
                    }
                    val dialog: AlertDialog = builder.create()
                    dialog.show()

                }
            }

        } else {
            val builder = AlertDialog.Builder(this)
            builder.setTitle("Error")
            builder.setMessage("An error was found while creating an account.
Please try again.")
            builder.setPositiveButton("Accept", null)
        }
}

private fun checkName(): Boolean {
        val box: TextInputLayout = findViewById(R.id.inputNameBox)
        val name: EditText = findViewById(R.id.inputNameRegister)
        if (name.text.toString() == "") {
            box.error = "Please enter full name"
            return false
        } else if  (name.length() >= 50) {
            box.error = "Full name too long"
            return false
        } else {
            box.error = null
            return true
        }
}

private fun checkPIN(): Boolean {
        val box: TextInputLayout = findViewById(R.id.boxPIN)
        val pin: EditText = findViewById(R.id.inputPIN)
        if (pin.text.toString() == "") {
            box.error = "Please enter a custom PIN"
            return false
        } else if  (pin.length() > 4) {
            box.error = "PIN not valid"
            return false
        } else {
            box.error = null
            return true
        }
}

private fun checkPhone(): Boolean {
        val box: TextInputLayout = findViewById(R.id.boxPhone)
        val phone: EditText = findViewById(R.id.inputPhone)
        if (phone.text.isEmpty()) {
            box.error = "Please enter your mobile phone number"
            return false
        } else if (phone.length() >= 10) {
            box.error = "Mobile phone number too long"
            return false
        } else {
            box.error = null
            return true
        }
    }

    private fun checkEmail(): Boolean {
        val email: EditText = findViewById(R.id.inputEmailRegister)
        val box: TextInputLayout = findViewById(R.id.boxEmail)
        if (email.text.isEmpty()) {
            box.error = "Please enter email address"
            return false
        } else {
            box.error = null
            return true
        }
    }

    private fun checkPassword(): Boolean {
        val password: EditText = findViewById(R.id.inputPasswordRegister)
        val passwordREGEX = Pattern.compile("^(?=.*[0-
9])(?=\\S+$).{6,}").toRegex()
        val box: TextInputLayout = findViewById(R.id.boxPassword)
        if (password.text.isEmpty()) {
            box.error = "Please enter password"
            return false
        } else if (passwordREGEX.matches(password.toString())) {
            box.error = "At least 6 characters, minimum 1 letter and 1 number"
            return false
        } else {
            box.error=null
            return true
        }
    }

    private fun checkConfirm(): Boolean {
        val password: EditText = findViewById(R.id.inputPasswordRegister)
        val confirm : EditText = findViewById(R.id.inputConfirmPassword)
        val box: TextInputLayout = findViewById(R.id.boxConfirm)
        if (confirm.text.isEmpty()) {
            box.error = "Please confirm your password"
            return false
        }
        else if (password.text.toString() != confirm.text.toString()) {
            box.error = "Password does not match"
            return false
        } else {
            box.error=null
            return true
        }
    }

    private fun checkJob(): Boolean {
        val spinner = findViewById<View>(R.id.spinner_edit_user) as Spinner
        if (spinner.selectedItem.toString() == "Please choose a job role")
        {
            val builder = AlertDialog.Builder(this)
            builder.setTitle("Error")
            builder.setMessage("Necessary to select a job role")
            builder.setPositiveButton("Accept", null)
            val dialog: AlertDialog = builder.create()
            dialog.show()
            return false
        } else {
            return true
        }
    }

}
```

A.4. Administrator home activity

```
class AdminHomeActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {

        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_home_admin)

        val menu : ImageButton = findViewById(R.id.btn_menu)
        menu.visibility = View.GONE

        val btnUsers = findViewById<ImageButton>(R.id.btn_users)
        btnUsers.setOnClickListener {
            val intent = Intent(this, UserActivity::class.java)
            startActivity(intent)
        }

        val btnEngineer = findViewById<ImageButton>(R.id.btn_engineer)
        btnEngineer.setOnClickListener {
            val intent = Intent(this, CheckListActivity::class.java)
            startActivity(intent) }

        val btnFleet = findViewById<ImageButton>(R.id.btn_fleet)
        btnFleet.setOnClickListener {
            val intent = Intent(this, FleetActivity::class.java)
            startActivity(intent) }

        val btnMail = findViewById<ImageButton>(R.id.btn_message)
        btnMail.setOnClickListener {
            val intent = Intent(this, MailActivity::class.java)
            startActivity(intent) }

        val calendar = Calendar.getInstance()
        val date = calendar.time

        val stringYear= DateFormat.format("yy", date) as String
        val stringMonth = DateFormat.format("MM", date) as String
        val stringNumber = DateFormat.format("dd", date) as String
        val stringDay = DateFormat.format("EEEE", date) as String
        val textDay = findViewById<TextView>(R.id.tool_title)
        textDay.text = ("$stringDay , $stringNumber / $stringMonth / $stringYear")

        val back : ImageButton = findViewById(R.id.btn_back)
        back.setBackgroundResource(R.drawable.ic_baseline_logout_24)

        back.setOnClickListener {
            val builder = AlertDialog.Builder(this)
            builder.setTitle("Warning!").setMessage("You sure, that you want to
logout?")
            builder.setPositiveButton(
                    "Yes"
            ) { dialog, id ->
                val i = Intent(this, WelcomeActivity::class.java)
                startActivity(i)
            }
            builder.setNegativeButton(
                    "No"
            ) { dialog, id -> dialog.cancel() }
            val alert = builder.create()
            alert.show()
        }

        val edit : ImageButton = findViewById(R.id.btn_edit_info)
        edit.setOnClickListener {
            val intent = Intent(this, EditUserInformation::class.java)
            startActivity(intent)
        }

        displayEmailJob() }
     override fun onBackPressed() {
        val builder = AlertDialog.Builder(this)
        builder.setTitle("Warning!").setMessage("Are you sure, that you want
to logout?")
```

```
        builder.setPositiveButton(
            "Yes"
        ) { dialog, id ->
            FirebaseAuth.getInstance().signOut()
            val i = Intent(this, WelcomeActivity::class.java)
            startActivity(i)
        }
        builder.setNegativeButton(
            "No"
        ) { dialog, id -> dialog.cancel() }
        val alert = builder.create()
        alert.show()
    }

    private fun displayEmailJob() {
        val user = Firebase.auth.currentUser
        val email= user.email
        val emailTextView = findViewById<TextView>(R.id.mailTextView)
        val jobTextView = findViewById<TextView>(R.id.loginAsTextView)
        val nameTextView = findViewById<TextView>(R.id.welcomeTextView)
        emailTextView.text = (email)
        val db = FirebaseFirestore.getInstance()
        val df: DocumentReference = db.collection("Users").document(email)
        df.get().addOnSuccessListener { documentSnapshot ->
            val job = documentSnapshot.getString("Job")
            jobTextView.text = ("$job")
            val name = documentSnapshot.getString("Name")
            nameTextView.text= ("$name")
        }

            val ref =
FirebaseStorage.getInstance().reference.child("profile/$email")
            val profile : CardView = this.findViewById(R.id.card_profile)
            val imageView = profile.findViewById<ImageView>(R.id.profilePicture)
            imageView.drawable
            ref.downloadUrl.addOnSuccessListener { Uri ->
                imageView.setBackgroundColor(Color.TRANSPARENT);
                val imageURL = Uri.toString()
                Glide.with(this)
                        .load(imageURL)
                        .into(imageView)
            }.addOnFailureListener {}

    }


}
```

A.5. AMT home activity

```
class AMTHomeActivity : AppCompatActivity() {

   override fun onCreate(savedInstanceState: Bundle?) {

      super.onCreate(savedInstanceState)
      setContentView(R.layout.activity_home_amt)

      val menu : ImageButton = findViewById(R.id.btn_menu)
      menu.visibility = View.GONE

      val btnAMT = findViewById<ImageButton>(R.id.btn_amt)
      btnAMT.setOnClickListener { openActivity1() }

      val btnMail = findViewById<ImageButton>(R.id.btn_message)
      btnMail.setOnClickListener {
         val intent = Intent(this, MailActivity::class.java)
         startActivity(intent) }

      val calendar = Calendar.getInstance()
      val date = calendar.time

      val stringYear= DateFormat.format("yy", date) as String
      val stringMonth = DateFormat.format("MM", date) as String
      val stringNumber = DateFormat.format("dd", date) as String
      val stringDay = DateFormat.format("EEEE", date) as String
      val textDay = findViewById<TextView>(R.id.tool_title)
      textDay.text = ("$stringDay , $stringNumber / $stringMonth / $stringYear")

      val back : ImageButton = findViewById(R.id.btn_back)
      back.setBackgroundResource(R.drawable.ic_baseline_logout_24)

      back.setOnClickListener {
         val builder = AlertDialog.Builder(this)
         builder.setTitle("Warning!").setMessage("You sure, that you want to
logout?")
         builder.setPositiveButton(
               "Yes"
         ) { dialog, id ->
            val i = Intent(this, WelcomeActivity::class.java)
            startActivity(i)
         }
         builder.setNegativeButton(
               "No"
         ) { dialog, id -> dialog.cancel() }
         val alert = builder.create()
         alert.show()
      }
```

```
        val edit : ImageButton = findViewById(R.id.btn_edit_info)
        edit.setOnClickListener {
            val intent = Intent(this, EditUserInformation::class.java)
            startActivity(intent)
        }

        displayEmailJob()

    }

    private fun openActivity1() {
        val intent = Intent(this, CheckListActivity::class.java)
        startActivity(intent)
    }

    private fun displayEmailJob() {
        val user = Firebase.auth.currentUser
        val email= user.email
        val emailTextView = findViewById<TextView>(R.id.mailTextView)
        val jobTextView = findViewById<TextView>(R.id.loginAsTextView)
        val nameTextView = findViewById<TextView>(R.id.welcomeTextView)
        emailTextView.text = ("$email")
        val db = FirebaseFirestore.getInstance()
        val df: DocumentReference = db.collection("Users").document(email)
        df.get().addOnSuccessListener { documentSnapshot ->
            val job = documentSnapshot.getString("Job")
            jobTextView.text = ("$job")
            val name = documentSnapshot.getString("Name")
            nameTextView.text= ("$name")
        }

        val ref = FirebaseStorage.getInstance().reference.child("profile/$email")
        val profile : CardView = this.findViewById(R.id.card_profile)
        val imageView = profile.findViewById<ImageView>(R.id.profilePicture)
        imageView.drawable
        ref.downloadUrl.addOnSuccessListener { Uri ->
            val imageURL = Uri.toString()
            Glide.with(this)
                    .load(imageURL)
                    .into(imageView)
        }.addOnFailureListener {
        }
    }

}
```

A.6. Check list activity

```
class CheckListActivity : AppCompatActivity() , MyItemClickListener,
MyItemLongClickListener {

   private var mAdapter: AircraftAdapter? = null

   override fun onCreate(savedInstanceState: Bundle?) {

      super.onCreate(savedInstanceState)
      setContentView(R.layout.activity_engineer)

      val back = findViewById<View>(R.id.btn_back) as ImageButton
      back.setOnClickListener { openActivity1() }

      val text: TextView = findViewById(R.id.tool_title)
      text.text = "Aircraft  check list"

      initView()
      initData()

      showMenu()

      val btn_search : Button = findViewById(R.id.button_search)
      btn_search.onFocusChangeListener = View.OnFocusChangeListener { a,
hasFocus ->
         if (!hasFocus) {
            btn_search.setHint("Search")
            hideKeyboard(a)
         } else btn_search.hint = ""
      }

      btn_search.setOnClickListener {
         val search : EditText = findViewById(R.id.search)
         if (search.text.toString().isEmpty()) {

         } else {
            searchBar()
         }
      }

      val btn_reset: Button = findViewById(R.id.button_reset)
      btn_reset.setOnClickListener {
         initData()
         btn_search.setHint("Search")
      }
   }

   override fun onBackPressed() {
        super.onBackPressed()
        openActivity1()
    }

    private fun openActivity1() {

        val user = FirebaseAuth.getInstance().currentUser
        val userEmail = user.email
        val db = FirebaseFirestore.getInstance()
        val df = db.collection("Users").document(userEmail)
        df.get().addOnSuccessListener { documentSnapshot ->

            if (documentSnapshot.getString("Job") == "MRO engineer" ||
                    documentSnapshot.getString("Job") == "CAMO engineer"
                    || documentSnapshot.getString("Job") == "Total access
engineer") {
                val intent = Intent(this, EngineerHomeActivity::class.java)
                startActivity(intent)
                finish()
            }

            if (documentSnapshot.getString("Job") == "Administrator") {
                val intent = Intent(this, AdminHomeActivity::class.java)
                startActivity(intent)
                finish()
            }

            if (documentSnapshot.getString("Job") == "Airplane maintenance
technician") {
                val intent = Intent(this, AMTHomeActivity::class.java)
                startActivity(intent)
                finish()
            }

        }
    }

    private fun initView() {
        val rvAircrafts =
findViewById<RecyclerView>(R.id.recyclerViewAircraftList)
        rvAircrafts.layoutManager = LinearLayoutManager(this)
        rvAircrafts.itemAnimator = DefaultItemAnimator()
    }

    private fun initData() {
        val db: FirebaseFirestore = FirebaseFirestore.getInstance()
        val collection: CollectionReference = db.collection("Aircraft revision")
        val query: Query = collection
        val rvAircrafts =
findViewById<RecyclerView>(R.id.recyclerViewAircraftList)
```

```
        query.get().addOnCompleteListener { task ->

            if (task.isSuccessful) {
                val mData: ArrayList<AircraftModel> = ArrayList()
                for (document in task.result) {
                    val taskItem: AircraftModel =
document.toObject(AircraftModel::class.java)
                    mData.add(taskItem)
                }
                mAdapter = AircraftAdapter(mData)
                rvAircrafts.adapter = mAdapter
                mAdapter!!.setOnItemClickListener(this)
                mAdapter!!.setOnItemLongClickListener(this)
                mAdapter!!.notifyDataSetChanged()
            }
        }

    }

    override fun onItemClick(view: View?, position: Int) {
        val db: FirebaseFirestore = FirebaseFirestore.getInstance()
        val collection: CollectionReference = db.collection("Aircraft revision")
        val query: Query = collection
        query.get().addOnCompleteListener { task ->
            if (task.isSuccessful) {
                val mData: ArrayList<AircraftModel> = ArrayList()
                for (document in task.result) {
                    val taskItem: AircraftModel =
document.toObject(AircraftModel::class.java)
                    mData.add(taskItem)
                }

                val bean: AircraftModel = mData[position - 1]
                val aircraft = bean.Aircraft
                val start = bean.Start
                val check = bean.Check
                val assigned = bean.Assigned

                val user = FirebaseAuth.getInstance().currentUser
                val userEmail = user.email
                val db = FirebaseFirestore.getInstance()
                val df = db.collection("Users").document(userEmail)
                df.get().addOnSuccessListener { documentSnapshot ->
                    if (documentSnapshot.getString("Job") == "MRO engineer"
                            || documentSnapshot.getString("Job") == "CAMO engineer"
                            || documentSnapshot.getString("Job") == "Total access
engineer") {
                        val inflater = LayoutInflater.from(this)
                        val popupView: View =
inflater.inflate(R.layout.pop_window_mro, null)
```

```
            val focusable = true

            val popAir: TextView = popupView.findViewById(R.id.tool_title)
            val popupWindow = PopupWindow(popupView, 900,
ViewGroup.LayoutParams.WRAP_CONTENT, focusable)
            popAir.text = aircraft
            popupWindow.showAtLocation(view, Gravity.CENTER, 0, 0)


popupWindow.contentView.findViewById<ImageButton>(R.id.btn_menu).visibilit
y = View.GONE

popupWindow.contentView.findViewById<ImageButton>(R.id.btn_back).visibilit
y = View.GONE


popupWindow.contentView.findViewById<Button>(R.id.pop_damage_map).set
OnClickListener {
                val intent = Intent(this, DamageMapActivity::class.java)
                startActivity(intent)
            }


popupWindow.contentView.findViewById<Button>(R.id.pop_documents).setOn
ClickListener {
                val intent = Intent(this, DocumentsActivity::class.java)
                startActivity(intent)
            }


popupWindow.contentView.findViewById<Button>(R.id.pop_new_damages).set
OnClickListener {

                val SHARED_PREF_AIRCRAFT = "aircraft"
                val KEY_AIRCRAFT = "key_aircraft"
                val sp = getSharedPreferences(SHARED_PREF_AIRCRAFT,
MODE_PRIVATE)
                val editor = sp.edit()
                editor.putString(KEY_AIRCRAFT, aircraft)
                editor.apply()

                val SHARED_PREF_START = "start"
                val KEY_START = "key_start"
                val sp1 = getSharedPreferences(SHARED_PREF_START,
MODE_PRIVATE)
                val editor1 = sp1.edit()
                editor1.putString(KEY_START, start)
                editor1.apply()

                val SHARED_PREF_check = "check"
```

```
                    val KEY_check = "key_check"
                    val sp2 = getSharedPreferences(SHARED_PREF_check,
MODE_PRIVATE)
                    val editor2 = sp2.edit()
                    editor2.putString(KEY_check, check)
                    editor2.apply()

                    val intent = Intent(this, DamageActivity::class.java)
                    startActivity(intent)
                }


popupWindow.contentView.findViewById<Button>(R.id.pop_edit).setOnClickList
ener {

                    val SHARED_PREF_AIRCRAFT = "aircraft"
                    val KEY_AIRCRAFT = "key_aircraft"
                    val sp = getSharedPreferences(SHARED_PREF_AIRCRAFT,
MODE_PRIVATE)
                    val editor = sp.edit()
                    editor.putString(KEY_AIRCRAFT, aircraft)
                    editor.apply()

                    val SHARED_PREF_START = "start"
                    val KEY_START = "key_start"
                    val sp1 = getSharedPreferences(SHARED_PREF_START,
MODE_PRIVATE)
                    val editor1 = sp1.edit()
                    editor1.putString(KEY_START, start)
                    editor1.apply()

                    val SHARED_PREF_check = "check"
                    val KEY_check = "key_check"
                    val sp2 = getSharedPreferences(SHARED_PREF_check,
MODE_PRIVATE)
                    val editor2 = sp2.edit()
                    editor2.putString(KEY_check, check)
                    editor2.apply()

                    val save = getSharedPreferences("USER",
MODE_PRIVATE)
                    val editor3 = save.edit()
                    editor3.putString("user", assigned)
                    editor3.apply()

                    val intent = Intent(this, EditCheckActivity::class.java)
                    startActivity(intent)
                }
```

```
popupWindow.contentView.findViewById<Button>(R.id.pop_comment).setOnCli
ckListener {

                        popupWindow.dismiss()
                        val inflater2 = LayoutInflater.from(this)
                        val popupView2: View =
inflater2.inflate(R.layout.pop_window_comment, null)
                        val focusable2 = true
                        val popupWindow2 = PopupWindow(popupView2, 900, 800,
focusable2)
                        popupWindow2.showAtLocation(view, Gravity.CENTER, 0, 0)
                        val date = start.toString()
                        val items1 = date.split("/".toRegex()).toTypedArray()
                        val day = items1[0]
                        val month = items1[1]
                        val year = items1[2]


popupWindow2.contentView.findViewById<TextView>(R.id.tool_title).text =
"Comments"

popupWindow2.contentView.findViewById<ImageButton>(R.id.btn_menu).visibil
ity = View.GONE

popupWindow2.contentView.findViewById<ImageButton>(R.id.btn_back).visibili
ty = View.GONE

popupWindow2.contentView.findViewById<Button>(R.id.pop_update).visibility =
View.GONE

popupWindow2.contentView.findViewById<Button>(R.id.pop_clear_all).visibility
= View.GONE

                        val db = FirebaseFirestore.getInstance()
                        val df = db.collection("Aircraft revision").document(aircraft +
"_" + check + "_" + day + month + year)
                        df.get().addOnSuccessListener { documentSnapshot ->
                            val editComment: TextView =
popupWindow2.contentView.findViewById(R.id.inputComment)
                            val value = documentSnapshot.getString("Comments")
                            editComment.text = value
                            val comment = editComment.text
                            val pop_update: Button =
popupWindow2.contentView.findViewById(R.id.pop_update)
                            pop_update.setOnClickListener {
                                if (comment.toString().isEmpty()) {
                                    db.collection("Aircraft revision")
                                            .document(aircraft + "_" + check + "_" + day +
month + year)
                                            .update("Comments", comment.toString(),
                                            "Text", "No")
                                    val builder = AlertDialog.Builder(this)
                                    builder.setTitle("Information")
                                    builder.setMessage("Succesfully updated")
                                    builder.setPositiveButton("Accept") { dialog, which ->
                                        mAdapter!!.notifyDataSetChanged()
                                        popupWindow2.dismiss()
                                    }
                                    val dialog: AlertDialog = builder.create()
                                    dialog.show()
                                }
                                if (comment.toString().isNotEmpty()) {
                                    db.collection("Aircraft revision")
                                            .document(aircraft + "_" + check + "_" + day +
month + year)
                                            .update(
                                            "Comments", comment.toString(),
                                            "Text", "Yes")
                                    val builder = AlertDialog.Builder(this)
                                    builder.setTitle("Information")
                                    builder.setMessage("Succesfully updated")
                                    builder.setPositiveButton("Accept") { dialog, which ->
                                            mAdapter!!.notifyDataSetChanged()
                                            popupWindow2.dismiss()
                                    }
                                    val dialog: AlertDialog = builder.create()
                                    dialog.show()
                                }

                            }

                            val pop_clear: Button =
popupWindow2.contentView.findViewById(R.id.pop_clear_all)
                            pop_clear.setOnClickListener {
                                editComment.text = ""
                            }
                        }
                    }

}

if (documentSnapshot.getString("Job") == "Airplane
maintenance technician") {

                    val inflater = LayoutInflater.from(this)
                    val popupView: View =
inflater.inflate(R.layout.pop_window_amt, null)
                    val focusable = true
                    val popAir: TextView = popupView.findViewById(R.id.tool_title)
                    val popupWindow = PopupWindow(popupView, 700,
ViewGroup.LayoutParams.WRAP_CONTENT, focusable)
                popAir.text = aircraft
                popupWindow.showAtLocation(view, Gravity.CENTER, 0, 0)


popupWindow.contentView.findViewById<ImageButton>(R.id.btn_menu).visibilit
y = View.GONE

popupWindow.contentView.findViewById<ImageButton>(R.id.btn_back).visibilit
y = View.GONE


popupWindow.contentView.findViewById<Button>(R.id.pop_new_damages).set
OnClickListener {
                    val intent = Intent(this, DamageActivity::class.java)
                    startActivity(intent)
                }


popupWindow.contentView.findViewById<Button>(R.id.pop_documents).setOn
ClickListener {
                    val intent = Intent(this, DocumentsActivity::class.java)
                    startActivity(intent)
                }


popupWindow.contentView.findViewById<Button>(R.id.pop_damage_map).set
OnClickListener {

                    val SHARED_PREF_AIRCRAFT = "aircraft"
                    val KEY_AIRCRAFT = "key_aircraft"
                    val sp = getSharedPreferences(SHARED_PREF_AIRCRAFT,
MODE_PRIVATE)
                    val editor = sp.edit()
                    editor.putString(KEY_AIRCRAFT, aircraft)
                    editor.apply()

                    val SHARED_PREF_START = "start"
                    val KEY_START = "key_start"
                    val sp1 = getSharedPreferences(SHARED_PREF_START,
MODE_PRIVATE)
                    val editor1 = sp1.edit()
                    editor1.putString(KEY_START, start)
                    editor1.apply()

                    val SHARED_PREF_check = "check"
                    val KEY_check = "key_check"
                    val sp2 = getSharedPreferences(SHARED_PREF_check,
MODE_PRIVATE)
                    val editor2 = sp2.edit()
                    editor2.putString(KEY_check, check)
                    editor2.apply()

                    val intent = Intent(this, DamageMapActivity::class.java)
                    startActivity(intent)
                 }


popupWindow.contentView.findViewById<Button>(R.id.pop_comment).setOnCli
ckListener {
                    popupWindow.dismiss()
                    val inflater2 = LayoutInflater.from(this)
                    val popupView2: View =
inflater2.inflate(R.layout.pop_window_comment, null)
                    val focusable2 = true
                    val popupWindow2 = PopupWindow(popupView2, 900, 800,
focusable2)
                    popupWindow2.showAtLocation(view, Gravity.CENTER, 0, 0)

                    val date = start.toString()
                    val items1 = date.split("/".toRegex()).toTypedArray()
                    val day = items1[0]
                    val month = items1[1]
                    val year = items1[2]


popupWindow2.contentView.findViewById<TextView>(R.id.tool_title).text =
"Comments"

popupWindow2.contentView.findViewById<ImageButton>(R.id.btn_menu).visibil
ity = View.GONE

popupWindow2.contentView.findViewById<ImageButton>(R.id.btn_back).visibili
ty = View.GONE

popupWindow2.contentView.findViewById<Button>(R.id.pop_update).visibility =
View.GONE

popupWindow2.contentView.findViewById<Button>(R.id.pop_clear_all).visibility
= View.GONE

                    val db = FirebaseFirestore.getInstance()
                    val df = db.collection("Aircraft revision").document(aircraft +
"_" + check + "_" + day + month + year)
                    df.get().addOnSuccessListener { documentSnapshot ->
                        val editComment: TextView =
popupWindow2.contentView.findViewById(R.id.inputComment)
                        val value = documentSnapshot.getString("Comments")
                        editComment.text = value
                    }
                    }

popupWindow.contentView.findViewById<Button>(R.id.pop_new_damages).set
OnClickListener {
                        val intent = Intent(this, DamageActivity::class.java)
                        startActivity(intent)
                        finish()
                    }

                }

                if (documentSnapshot.getString("Job") == "Administrator") {

                    val inflater = LayoutInflater.from(this)
                    val popupView: View =
inflater.inflate(R.layout.pop_window_engineer, null)
                    val focusable = true

                    val popAir: TextView = popupView.findViewById(R.id.tool_title)
                    val popupWindow = PopupWindow(popupView, 900,
ViewGroup.LayoutParams.WRAP_CONTENT, focusable)
                    popAir.text = aircraft
                    popupWindow.showAtLocation(view, Gravity.CENTER, 0, 0)


popupWindow.contentView.findViewById<ImageButton>(R.id.btn_menu).visibilit
y = View.GONE

popupWindow.contentView.findViewById<ImageButton>(R.id.btn_back).visibilit
y = View.GONE


popupWindow.contentView.findViewById<Button>(R.id.pop_damage_map).set
OnClickListener {
                        val intent = Intent(this, DamageMapActivity::class.java)
                        startActivity(intent)
                    }


popupWindow.contentView.findViewById<Button>(R.id.pop_documents).setOn
ClickListener {
                        val intent = Intent(this, DocumentsActivity::class.java)
                        startActivity(intent)
                    }


popupWindow.contentView.findViewById<Button>(R.id.pop_edit).setOnClickList
ener {
```

```
                val SHARED_PREF_AIRCRAFT = "aircraft"
                val KEY_AIRCRAFT = "key_aircraft"
                val sp = getSharedPreferences(SHARED_PREF_AIRCRAFT,
MODE_PRIVATE)
                val editor = sp.edit()
                editor.putString(KEY_AIRCRAFT, aircraft)
                editor.apply()

                val SHARED_PREF_START = "start"
                val KEY_START = "key_start"
                val sp1 = getSharedPreferences(SHARED_PREF_START,
MODE_PRIVATE)
                val editor1 = sp1.edit()
                editor1.putString(KEY_START, start)
                editor1.apply()

                val SHARED_PREF_check = "check"
                val KEY_check = "key_check"
                val sp2 = getSharedPreferences(SHARED_PREF_check,
MODE_PRIVATE)
                val editor2 = sp2.edit()
                editor2.putString(KEY_check, check)
                editor2.apply()

                val SHARED_PREF_assign = "assign"
                val KEY_assign = "key_assign"
                val sp3 = getSharedPreferences(SHARED_PREF_assign,
MODE_PRIVATE)
                val editor3 = sp3.edit()
                editor3.putString(KEY_assign, assigned)
                editor3.apply()

                val intent = Intent(this, EditCheckActivity::class.java)
                startActivity(intent)
              }


popupWindow.contentView.findViewById<Button>(R.id.pop_comment).setOnCli
ckListener {
                popupWindow.dismiss()
                val inflater2 = LayoutInflater.from(this)
                val popupView2: View =
inflater2.inflate(R.layout.pop_window_comment, null)
                val focusable2 = true
                val popupWindow2 = PopupWindow(popupView2, 900, 800,
focusable2)
                popupWindow2.showAtLocation(view, Gravity.CENTER, 0, 0)
                val date = start.toString()
                val items1 = date.split("/".toRegex()).toTypedArray()
```

```
                    val day = items1[0]
                    val month = items1[1]
                    val year = items1[2]


popupWindow2.contentView.findViewById<TextView>(R.id.tool_title).text =
"Comments"

popupWindow2.contentView.findViewById<ImageButton>(R.id.btn_menu).visibil
ity = View.GONE

popupWindow2.contentView.findViewById<ImageButton>(R.id.btn_back).visibili
ty = View.GONE

popupWindow2.contentView.findViewById<Button>(R.id.pop_update).visibility =
View.GONE

popupWindow2.contentView.findViewById<Button>(R.id.pop_clear_all).visibility
= View.GONE

                    val db = FirebaseFirestore.getInstance()
                    val df = db.collection("Aircraft revision").document(aircraft +
"_" + check + "_" + day + month + year)
                    df.get().addOnSuccessListener { documentSnapshot ->
                       val comment =
documentSnapshot.getString("Comments")
                       val editComment: TextView =
popupWindow2.contentView.findViewById(R.id.inputComment)
                       editComment.text = comment
                       val pop_update: Button =
popupWindow2.contentView.findViewById(R.id.pop_update)
                       pop_update.setOnClickListener {
                          if (editComment.text.toString().isNotEmpty()) {
                             if (editComment.text.toString() == comment.toString())
{
                                db.collection("Aircraft revision")
                                      .document(aircraft + "_" + check + "_" + day +
month + year).update(mapOf("Comments" to editComment.text.toString(),
"Text" to "Yes"))
                                val builder = AlertDialog.Builder(this)
                                builder.setTitle("Information")
                                builder.setMessage("Succesfully updated")
                                builder.setPositiveButton("Accept") { dialog, which
->
                                   val intent = Intent(this,
CheckListActivity::class.java)
                                   startActivity(intent)
                                }
                                val dialog: AlertDialog = builder.create()
                                dialog.show()
                                }
                            } else {
                                db.collection("Aircraft revision")
                                        .document(aircraft + "_" + check + "_" + day +
month + year).update(mapOf("Comments" to editComment.text.toString(),
"Text" to "No"))
                                val builder = AlertDialog.Builder(this)
                                builder.setTitle("Information")
                                builder.setMessage("Succesfully updated")
                                builder.setPositiveButton("Accept") { dialog, which ->
                                    val intent = Intent(this,
CheckListActivity::class.java)
                                    startActivity(intent)
                                    finish()
                                }
                                val dialog: AlertDialog = builder.create()
                                dialog.show()
                            }
                        }
                        val pop_clear: Button =
popupWindow2.contentView.findViewById(R.id.pop_clear_all)
                        pop_clear.setOnClickListener {
                            editComment.text = ""
                        }
                    }
                }


popupWindow.contentView.findViewById<Button>(R.id.pop_new_damages).set
OnClickListener {
                    val intent = Intent(this, DamageActivity::class.java)
                    startActivity(intent)
                }


popupWindow.contentView.findViewById<Button>(R.id.pop_delete).setOnClick
Listener {
                    popupWindow.dismiss()
                    val date = start.toString()
                    val items1 = date.split("/".toRegex()).toTypedArray()
                    val day = items1[0]
                    val month = items1[1]
                    val year = items1[2]

                    val inflater3 = LayoutInflater.from(this)
                    val popupView3: View =
inflater3.inflate(R.layout.pop_window_pin, null)
                    val focusable3 = true
                    val popupWindow3 = PopupWindow(popupView3, 900, 400,
focusable3)
```

```
                    popupWindow3.showAtLocation(view, Gravity.CENTER, 0, 0)


popupWindow3.contentView.findViewById<TextView>(R.id.tool_title).text =
"WRITE PIN"

popupWindow3.contentView.findViewById<ImageButton>(R.id.btn_menu).visibil
ity = View.GONE

popupWindow3.contentView.findViewById<ImageButton>(R.id.btn_back).visibili
ty = View.GONE

                    val db = FirebaseFirestore.getInstance()
                    val user = FirebaseAuth.getInstance().currentUser
                    val userEmail = user.email
                    val df =
db.collection("Users").document(userEmail.toString())
                    df.get().addOnSuccessListener { documentSnapshot ->
                        val txtpin =
popupWindow3.contentView.findViewById<TextView>(R.id.inputPIN)
                        val delete: Button =
popupWindow3.contentView.findViewById(R.id.pop_pin)
                        delete.setOnClickListener {
                            val builder = AlertDialog.Builder(this)
                            builder.setTitle("CAUTION")
                            builder.setMessage("Are you sure you want to
continue?")
                            builder.setPositiveButton("Accept") { dialog, which ->
                                if (documentSnapshot.getString("PIN") ==
txtpin.text.toString()) {
                                    val df = db.collection("Aircraft
revision").document(aircraft + "_" + check + "_" + day + month + year)
                                    df.get().addOnSuccessListener {
documentSnapshot ->
                                        if (documentSnapshot.getString("ID") == (aircraft
+ "_" + check + "_" + day + month + year)) {
                                            db.collection("Aircraft revision")
                                                .document(aircraft + "_" + check + "_" +
day + month + year).delete()
                                            val intent = Intent(this,
CheckListActivity::class.java)
                                            startActivity(intent)
                                        }
                                    }
                                } else {
                                    val builder1 = AlertDialog.Builder(this)
                                    builder1.setTitle("Error")
                                    builder1.setMessage("Incorrect PIN, please try
again.")
                                    builder1.setPositiveButton("Accept", null)
                                    val dialog1: AlertDialog = builder1.create()
                                    dialog1.show()
                                }
                            }
                            builder.setNegativeButton("Cancel", null)
                            val dialog: AlertDialog = builder.create()
                            dialog.show()
                        }
                    }
                }

                val SHARED_PREF_AIRCRAFT = "aircraft"
                val KEY_AIRCRAFT = "key_aircraft"
                val sp = getSharedPreferences(SHARED_PREF_AIRCRAFT,
MODE_PRIVATE)
                val editor = sp.edit()
                editor.putString(KEY_AIRCRAFT, aircraft)
                editor.apply()

                val SHARED_PREF_START = "start"
                val KEY_START = "key_start"
                val sp1 = getSharedPreferences(SHARED_PREF_START,
MODE_PRIVATE)
                val editor1 = sp1.edit()
                editor1.putString(KEY_START, start)
                editor1.apply()

                val SHARED_PREF_check = "check"
                val KEY_check = "key_check"
                val sp2 = getSharedPreferences(SHARED_PREF_check,
MODE_PRIVATE)
                val editor2 = sp2.edit()
                editor2.putString(KEY_check, check)
                editor2.apply()

              }
            }
          }
        }
      }

      private fun initSortStart() {
        val db: FirebaseFirestore = FirebaseFirestore.getInstance()
        val collection: CollectionReference = db.collection("Aircraft revision")
        val query: Query = collection
        val rvAircrafts =
findViewById<RecyclerView>(R.id.recyclerViewAircraftList)
        query.get().addOnCompleteListener { task ->

          if (task.isSuccessful) {
```

```
                val mData: ArrayList<AircraftModel> = ArrayList()
                for (document in task.result) {
                    val taskItem: AircraftModel =
document.toObject(AircraftModel::class.java)
                    mData.add(taskItem)
                }
                mData.sortBy {it.timestamp_begin}
                mAdapter = AircraftAdapter(mData)
                rvAircrafts.adapter = mAdapter
                mAdapter!!.setOnItemClickListener(this)
                mAdapter!!.setOnItemLongClickListener(this)
                mAdapter!!.notifyDataSetChanged()
            }
        }

    }

    private fun initSortStartDescending() {
        val db: FirebaseFirestore = FirebaseFirestore.getInstance()
        val collection: CollectionReference = db.collection("Aircraft revision")
        val query: Query = collection
        val rvAircrafts =
findViewById<RecyclerView>(R.id.recyclerViewAircraftList)
        query.get().addOnCompleteListener { task ->

            if (task.isSuccessful) {
                val mData: ArrayList<AircraftModel> = ArrayList()
                for (document in task.result) {
                    val taskItem: AircraftModel =
document.toObject(AircraftModel::class.java)
                    mData.add(taskItem)
                }
                mData.sortByDescending {it.timestamp_begin}
                mAdapter = AircraftAdapter(mData)
                rvAircrafts.adapter = mAdapter
                mAdapter!!.setOnItemClickListener(this)
                mAdapter!!.setOnItemLongClickListener(this)
                mAdapter!!.notifyDataSetChanged()
            }
        }

    }

    private fun initSortEnd() {
        val db: FirebaseFirestore = FirebaseFirestore.getInstance()
        val collection: CollectionReference = db.collection("Aircraft revision")
        val query: Query = collection
        val rvAircrafts =
findViewById<RecyclerView>(R.id.recyclerViewAircraftList)
        query.get().addOnCompleteListener { task ->
```

```
            if (task.isSuccessful) {
                val mData: ArrayList<AircraftModel> = ArrayList()
                for (document in task.result) {
                    val taskItem: AircraftModel =
document.toObject(AircraftModel::class.java)
                    mData.add(taskItem)
                }
                mData.sortBy {it.timestamp_end}
                mAdapter = AircraftAdapter(mData)
                rvAircrafts.adapter = mAdapter
                mAdapter!!.setOnItemClickListener(this)
                mAdapter!!.setOnItemLongClickListener(this)
                mAdapter!!.notifyDataSetChanged()
            }
        }

    }

    private fun initSortEndDescending() {
        val db: FirebaseFirestore = FirebaseFirestore.getInstance()
        val collection: CollectionReference = db.collection("Aircraft revision")
        val query: Query = collection
        val rvAircrafts =
findViewById<RecyclerView>(R.id.recyclerViewAircraftList)
        query.get().addOnCompleteListener { task ->

            if (task.isSuccessful) {
                val mData: ArrayList<AircraftModel> = ArrayList()
                for (document in task.result) {
                    val taskItem: AircraftModel =
document.toObject(AircraftModel::class.java)
                    mData.add(taskItem)
                }
                mData.sortByDescending {it.timestamp_begin}
                mAdapter = AircraftAdapter(mData)
                rvAircrafts.adapter = mAdapter
                mAdapter!!.setOnItemClickListener(this)
                mAdapter!!.setOnItemLongClickListener(this)
                mAdapter!!.notifyDataSetChanged()
            }
        }

    }

    private fun initSortCRS() {
        val db: FirebaseFirestore = FirebaseFirestore.getInstance()
        val collection: CollectionReference = db.collection("Aircraft revision")
        val query: Query = collection
        val rvAircrafts =
findViewById<RecyclerView>(R.id.recyclerViewAircraftList)
        query.get().addOnCompleteListener { task ->

            if (task.isSuccessful) {
                val mData: ArrayList<AircraftModel> = ArrayList()
                for (document in task.result) {
                    val taskItem: AircraftModel =
document.toObject(AircraftModel::class.java)
                    mData.add(taskItem)
                }
                mData.sortBy {it.timestamp_crs}
                mAdapter = AircraftAdapter(mData)
                rvAircrafts.adapter = mAdapter
                mAdapter!!.setOnItemClickListener(this)
                mAdapter!!.setOnItemLongClickListener(this)
                mAdapter!!.notifyDataSetChanged()
            }
        }

    }

    private fun initSortCRSDescending() {
        val db: FirebaseFirestore = FirebaseFirestore.getInstance()
        val collection: CollectionReference = db.collection("Aircraft revision")
        val query: Query = collection
        val rvAircrafts =
findViewById<RecyclerView>(R.id.recyclerViewAircraftList)
        query.get().addOnCompleteListener { task ->

            if (task.isSuccessful) {
                val mData: ArrayList<AircraftModel> = ArrayList()
                for (document in task.result) {
                    val taskItem: AircraftModel =
document.toObject(AircraftModel::class.java)
                    mData.add(taskItem)
                }
                mData.sortByDescending {it.timestamp_crs}
                mAdapter = AircraftAdapter(mData)
                rvAircrafts.adapter = mAdapter
                mAdapter!!.setOnItemClickListener(this)
                mAdapter!!.setOnItemLongClickListener(this)
                mAdapter!!.notifyDataSetChanged()
            }
        }

    }

    private fun initSortAircraft() {
        val db: FirebaseFirestore = FirebaseFirestore.getInstance()
        val collection: CollectionReference = db.collection("Aircraft revision")
        val query: Query = collection
        val rvAircrafts =
findViewById<RecyclerView>(R.id.recyclerViewAircraftList)
        query.get().addOnCompleteListener { task ->

            if (task.isSuccessful) {
                val mData: ArrayList<AircraftModel> = ArrayList()
                for (document in task.result) {
                    val taskItem: AircraftModel =
document.toObject(AircraftModel::class.java)
                    mData.add(taskItem)
                }
                mData.sortBy {it.Aircraft}
                mAdapter = AircraftAdapter(mData)
                rvAircrafts.adapter = mAdapter
                mAdapter!!.setOnItemClickListener(this)
                mAdapter!!.setOnItemLongClickListener(this)
                mAdapter!!.notifyDataSetChanged()
            }
        }

    }

    private fun initSortAircraftDescending() {
        val db: FirebaseFirestore = FirebaseFirestore.getInstance()
        val collection: CollectionReference = db.collection("Aircraft revision")
        val query: Query = collection
        val rvAircrafts =
findViewById<RecyclerView>(R.id.recyclerViewAircraftList)
        query.get().addOnCompleteListener { task ->

            if (task.isSuccessful) {
                val mData: ArrayList<AircraftModel> = ArrayList()
                for (document in task.result) {
                    val taskItem: AircraftModel =
document.toObject(AircraftModel::class.java)
                    mData.add(taskItem)
                }
                mData.sortByDescending {it.Aircraft}
                mAdapter = AircraftAdapter(mData)
                rvAircrafts.adapter = mAdapter
                mAdapter!!.setOnItemClickListener(this)
                mAdapter!!.setOnItemLongClickListener(this)
                mAdapter!!.notifyDataSetChanged()
            }
        }

    }

    private fun initSortCheck() {
      val db: FirebaseFirestore = FirebaseFirestore.getInstance()
      val collection: CollectionReference = db.collection("Aircraft revision")
      val query: Query = collection
      val rvAircrafts =
findViewById<RecyclerView>(R.id.recyclerViewAircraftList)
      query.get().addOnCompleteListener { task ->

         if (task.isSuccessful) {
            val mData: ArrayList<AircraftModel> = ArrayList()
            for (document in task.result) {
               val taskItem: AircraftModel =
document.toObject(AircraftModel::class.java)
               mData.add(taskItem)
            }
            mData.sortBy {it.Check}
            mAdapter = AircraftAdapter(mData)
            rvAircrafts.adapter = mAdapter
            mAdapter!!.setOnItemClickListener(this)
            mAdapter!!.setOnItemLongClickListener(this)
            mAdapter!!.notifyDataSetChanged()
         }
      }
   }

   private fun initSortCheckDescending() {
      val db: FirebaseFirestore = FirebaseFirestore.getInstance()
      val collection: CollectionReference = db.collection("Aircraft revision")
      val query: Query = collection
      val rvAircrafts =
findViewById<RecyclerView>(R.id.recyclerViewAircraftList)
      query.get().addOnCompleteListener { task ->

         if (task.isSuccessful) {
            val mData: ArrayList<AircraftModel> = ArrayList()
            for (document in task.result) {
               val taskItem: AircraftModel =
document.toObject(AircraftModel::class.java)
               mData.add(taskItem)
            }
            mData.sortByDescending {it.Check}
            mAdapter = AircraftAdapter(mData)
            rvAircrafts.adapter = mAdapter
            mAdapter!!.setOnItemClickListener(this)
            mAdapter!!.setOnItemLongClickListener(this)
            mAdapter!!.notifyDataSetChanged()
         }
      }

   }
```

```
    private fun initSortStatus() {
        val db: FirebaseFirestore = FirebaseFirestore.getInstance()
        val collection: CollectionReference = db.collection("Aircraft revision")
        val query: Query = collection
        val rvAircrafts =
findViewById<RecyclerView>(R.id.recyclerViewAircraftList)
        query.get().addOnCompleteListener { task ->

            if (task.isSuccessful) {
                val mData: ArrayList<AircraftModel> = ArrayList()
                for (document in task.result) {
                    val taskItem: AircraftModel =
document.toObject(AircraftModel::class.java)
                    mData.add(taskItem)
                }
                mData.sortBy {it.Status}
                mAdapter = AircraftAdapter(mData)
                rvAircrafts.adapter = mAdapter
                mAdapter!!.setOnItemClickListener(this)
                mAdapter!!.setOnItemLongClickListener(this)
                mAdapter!!.notifyDataSetChanged()
            }
        }
    }

    private fun initSortStatusDescending() {
        val db: FirebaseFirestore = FirebaseFirestore.getInstance()
        val collection: CollectionReference = db.collection("Aircraft revision")
        val query: Query = collection
        val rvAircrafts =
findViewById<RecyclerView>(R.id.recyclerViewAircraftList)
        query.get().addOnCompleteListener { task ->

            if (task.isSuccessful) {
                val mData: ArrayList<AircraftModel> = ArrayList()
                for (document in task.result) {
                    val taskItem: AircraftModel =
document.toObject(AircraftModel::class.java)
                    mData.add(taskItem)
                }
                mData.sortByDescending {it.Status}
                mAdapter = AircraftAdapter(mData)
                rvAircrafts.adapter = mAdapter
                mAdapter!!.setOnItemClickListener(this)
                mAdapter!!.setOnItemLongClickListener(this)
                mAdapter!!.notifyDataSetChanged()
            }
        }

    }
    private fun showMenu() {
        val button : ImageButton = findViewById(R.id.btn_menu)
        button.setOnClickListener {
            val user = FirebaseAuth.getInstance().currentUser
            val userEmail = user.email
            val db = FirebaseFirestore.getInstance()
            val df = db.collection("Users").document(userEmail)
            df.get().addOnSuccessListener { documentSnapshot ->
                if (documentSnapshot.getString("Job") == "MRO engineer" ||
                        documentSnapshot.getString("Job") == "CAMO engineer"
                        || documentSnapshot.getString("Job") == "Total access
engineer"
                        || documentSnapshot.getString("Job") == "Administrator") {
                    val inflater = LayoutInflater.from(this)
                    val popupView: View = inflater.inflate(R.layout.menu, null)
                    val focusable = true
                    val popupWindow = PopupWindow(popupView, 500, 1000,
focusable)
                    button.x.toInt()
                    button.y.toInt()
                    popupWindow.showAsDropDown(button)

popupWindow.contentView.findViewById<TextView>(R.id.add_text).text =
"ADD CHECK"

popupWindow.contentView.findViewById<LinearLayout>(R.id.add).setOnClickLi
stener {
                        val SHARED_PREF_START = "start";
                        val sp = getSharedPreferences(SHARED_PREF_START,
MODE_PRIVATE)
                        sp.edit().clear().apply()
                        val intent = Intent(this, AddCheckActivity::class.java)
                        startActivity(intent)
                    }

popupWindow.contentView.findViewById<LinearLayout>(R.id.advancedSearch)
.setOnClickListener {
                        val inflater2 = LayoutInflater.from(this)
                        val popupView2: View =
inflater2.inflate(R.layout.pop_window_search_check, null)
                        val focusable2 = true
                        val popupWindow2 = PopupWindow(popupView2,
ViewGroup.LayoutParams.WRAP_CONTENT,
ViewGroup.LayoutParams.WRAP_CONTENT, focusable2)
                        popupWindow2.showAsDropDown(button)
                        popupWindow.dismiss()

                        val aircraft =
popupWindow2.contentView.findViewById<EditText>(R.id.inputAircraft)
```

```
            aircraft.onFocusChangeListener =
View.OnFocusChangeListener { b, hasFocus ->
                if (!hasFocus) {
                    aircraft.setHint("-")
                    hideKeyboard(b)
                } else aircraft.hint = ""
            }
            val begin =
popupWindow2.contentView.findViewById<EditText>(R.id.inputBegin)
            begin.onFocusChangeListener = View.OnFocusChangeListener
{ b, hasFocus ->
                if (!hasFocus) {
                    begin.setHint("-")
                    hideKeyboard(b)
                } else begin.hint = ""
            }
            val end =
popupWindow2.contentView.findViewById<EditText>(R.id.inputEnd)
            end.onFocusChangeListener = View.OnFocusChangeListener {
b, hasFocus ->
                if (!hasFocus) {
                    end.setHint("-")
                    hideKeyboard(b)
                } else end.hint = ""
            }
            val crs =
popupWindow2.contentView.findViewById<EditText>(R.id.inputCRS)
            crs.onFocusChangeListener = View.OnFocusChangeListener {
b, hasFocus ->
                if (!hasFocus) {
                    crs.setHint("-")
                    hideKeyboard(b)
                } else crs.hint = ""
            }
            val check =
popupWindow2.contentView.findViewById<EditText>(R.id.inputCheck)
            check.onFocusChangeListener = View.OnFocusChangeListener
{ b, hasFocus ->
                if (!hasFocus) {
                    check.setHint("-")
                    hideKeyboard(b)
                } else check.hint = ""
            }
            val status =
popupWindow2.contentView.findViewById<EditText>(R.id.inputStatus)
            status.onFocusChangeListener = View.OnFocusChangeListener
{ b, hasFocus ->
                if (!hasFocus) {
                    status.setHint("-")
                    hideKeyboard(b)
                } else status.hint = ""
            }

            val btn_search : Button =
popupWindow2.contentView.findViewById(R.id.button_advanced)
            btn_search.setOnClickListener {
                val aircraft : EditText =
popupWindow2.contentView.findViewById(R.id.inputAircraft)
                if (aircraft.text.toString().isEmpty()) {
                    initData()
                } else {
                    val db : FirebaseFirestore= FirebaseFirestore.getInstance()
                    val rvAircraft=
findViewById<RecyclerView>(R.id.recyclerViewAircraftList)
                    db.collection("Aircraft revision")
                            .whereEqualTo("Aircraft",
aircraft.text.toString()).get().addOnSuccessListener { documents ->
                                    val mData: ArrayList<AircraftModel> = ArrayList()
                                    for (document in documents) {
                                        val taskItem: AircraftModel =
document.toObject(AircraftModel::class.java)
                                        mData.add(taskItem)
                                    }
                                    mAdapter = AircraftAdapter(mData)
                                    rvAircraft.adapter = mAdapter
                                    mAdapter!!.setOnItemClickListener(this)
                                    mAdapter!!.setOnItemLongClickListener(this)
                                    mAdapter!!.notifyDataSetChanged()
                            }
                }
            }
            val btn_reset : Button =
popupWindow2.contentView.findViewById(R.id.button_reset)
            btn_reset.setOnClickListener {
                initData()
            }
        }


popupWindow.contentView.findViewById<LinearLayout>(R.id.sortBy).setOnClic
kListener {

            val inflater3 = LayoutInflater.from(this)
            val popupView3: View =
inflater3.inflate(R.layout.pop_up_sortby_engineer, null)
            val popupWindow3 = PopupWindow(popupView3,
ViewGroup.LayoutParams.WRAP_CONTENT,
ViewGroup.LayoutParams.WRAP_CONTENT, true)
            popupWindow3.showAsDropDown(button)
```

```
popupWindow3.contentView.findViewById<RadioButton>(R.id.radioAirDescendi
ng).setOnClickListener {
                    initSortAircraft()
                }

popupWindow3.contentView.findViewById<RadioButton>(R.id.radioAirAscendin
g).setOnClickListener {
                    initSortAircraftDescending()
                }

popupWindow3.contentView.findViewById<RadioButton>(R.id.radioStartDesce
nding).setOnClickListener {
                    initSortStart()
                }

popupWindow3.contentView.findViewById<RadioButton>(R.id.radioStartAscend
ing).setOnClickListener {
                    initSortStartDescending()
                }

popupWindow3.contentView.findViewById<RadioButton>(R.id.radioEndDescen
ding).setOnClickListener {
                    initSortEnd()
                }

popupWindow3.contentView.findViewById<RadioButton>(R.id.radioEndAscendi
ng).setOnClickListener {
                    initSortEndDescending()
                }

popupWindow3.contentView.findViewById<RadioButton>(R.id.radioCRSDesce
nding).setOnClickListener {
                    initSortCRS()
                }

popupWindow3.contentView.findViewById<RadioButton>(R.id.radioCRSAscend
ing).setOnClickListener {
                    initSortCRSDescending()
                }

popupWindow3.contentView.findViewById<RadioButton>(R.id.radioCheckDesc
ending).setOnClickListener {
                    initSortCheck()
                }

popupWindow3.contentView.findViewById<RadioButton>(R.id.radioCheckAsce
nding).setOnClickListener {
                    initSortCheckDescending()
                }
```

```
popupWindow3.contentView.findViewById<RadioButton>(R.id.radioStatusDesc
ending).setOnClickListener {
                    initSortStatus()
                }

popupWindow3.contentView.findViewById<RadioButton>(R.id.radioStatusAsce
nding).setOnClickListener {
                    initSortStatusDescending()
                }
                popupWindow.dismiss()
            }

            popupView.setOnClickListener { popupWindow.dismiss() }
            popupView.setOnTouchListener { v, event ->
                popupWindow.dismiss()
                true
            }

        }
        if (documentSnapshot.getString("Job") == "Airplane maintenance
technician") {
            val inflater = LayoutInflater.from(this)
            val popupView: View = inflater.inflate(R.layout.menu, null)
            val focusable = true
            val popupWindow = PopupWindow(popupView, 500, 1000,
focusable)
            button.x.toInt()
            button.y.toInt()
            popupWindow.showAsDropDown(button)


popupWindow.contentView.findViewById<LinearLayout>(R.id.add).visibility =
View.GONE


popupWindow.contentView.findViewById<LinearLayout>(R.id.advancedSearch)
.setOnClickListener {
                val inflater2 = LayoutInflater.from(this)
                val popupView2: View =
inflater2.inflate(R.layout.pop_window_search_check, null)
                val focusable2 = true
                val popupWindow2 = PopupWindow(popupView2,
ViewGroup.LayoutParams.WRAP_CONTENT,
ViewGroup.LayoutParams.WRAP_CONTENT, focusable2)
                popupWindow2.showAsDropDown(button)
                popupWindow.dismiss()

                val aircraft =
popupWindow2.contentView.findViewById<EditText>(R.id.inputAircraft)
                aircraft.onFocusChangeListener =
View.OnFocusChangeListener { b, hasFocus ->
                    if (!hasFocus) {
                        aircraft.setHint("-")
                        hideKeyboard(b)
                    } else aircraft.hint = ""
                }
                val begin =
popupWindow2.contentView.findViewById<EditText>(R.id.inputBegin)
                begin.onFocusChangeListener = View.OnFocusChangeListener
{ b, hasFocus ->
                    if (!hasFocus) {
                        begin.setHint("-")
                        hideKeyboard(b)
                    } else begin.hint = ""
                }
                val end =
popupWindow2.contentView.findViewById<EditText>(R.id.inputEnd)
                end.onFocusChangeListener = View.OnFocusChangeListener {
b, hasFocus ->
                    if (!hasFocus) {
                        end.setHint("-")
                        hideKeyboard(b)
                    } else end.hint = ""
                }
                val crs =
popupWindow2.contentView.findViewById<EditText>(R.id.inputCRS)
                crs.onFocusChangeListener = View.OnFocusChangeListener {
b, hasFocus ->
                    if (!hasFocus) {
                        crs.setHint("-")
                        hideKeyboard(b)
                    } else crs.hint = ""
                }
                val check =
popupWindow2.contentView.findViewById<EditText>(R.id.inputCheck)
                check.onFocusChangeListener = View.OnFocusChangeListener
{ b, hasFocus ->
                    if (!hasFocus) {
                        check.setHint("-")
                        hideKeyboard(b)
                    } else check.hint = ""
                }
                val status =
popupWindow2.contentView.findViewById<EditText>(R.id.inputStatus)
                status.onFocusChangeListener = View.OnFocusChangeListener
{ b, hasFocus ->
                    if (!hasFocus) {
                        status.setHint("-")
                        hideKeyboard(b)
                    } else status.hint = ""
                }

                val btn_search : Button =
popupWindow2.contentView.findViewById(R.id.button_advanced)
                btn_search.setOnClickListener {
                    val aircraft : EditText =
popupWindow2.contentView.findViewById(R.id.inputAircraft)
                    if (aircraft.text.toString().isEmpty()) {
                        initData()
                    } else {
                        val db : FirebaseFirestore= FirebaseFirestore.getInstance()
                        val rvAircraft=
findViewById<RecyclerView>(R.id.recyclerViewAircraftList)
                        db.collection("Aircraft revision")
                                .whereEqualTo("Aircraft",
aircraft.text.toString()).get().addOnSuccessListener { documents ->
                                            val mData: ArrayList<AircraftModel> = ArrayList()
                                            for (document in documents) {
                                                val taskItem: AircraftModel =
document.toObject(AircraftModel::class.java)
                                                mData.add(taskItem)
                                            }
                                            mAdapter = AircraftAdapter(mData)
                                            rvAircraft.adapter = mAdapter
                                            mAdapter!!.setOnItemClickListener(this)
                                            mAdapter!!.setOnItemLongClickListener(this)
                                            mAdapter!!.notifyDataSetChanged()
                                }
                    }
                }
                val btn_reset : Button =
popupWindow2.contentView.findViewById(R.id.button_reset)
                btn_reset.setOnClickListener {
                    initData()
                }
            }


popupWindow.contentView.findViewById<LinearLayout>(R.id.sortBy).setOnClic
kListener {
                val inflater3 = LayoutInflater.from(this)
                val popupView3: View =
inflater3.inflate(R.layout.pop_up_sortby_engineer, null)
                val popupWindow3 = PopupWindow(popupView3,
ViewGroup.LayoutParams.WRAP_CONTENT,
ViewGroup.LayoutParams.WRAP_CONTENT, true)
                popupWindow3.showAsDropDown(button)

popupWindow3.contentView.findViewById<RadioButton>(R.id.radioAirDescendi
ng).setOnClickListener {
```

```
                    initSortAircraft()
                }

popupWindow3.contentView.findViewById<RadioButton>(R.id.radioAirAscendin
g).setOnClickListener {
                    initSortAircraftDescending()
                }

popupWindow3.contentView.findViewById<RadioButton>(R.id.radioStartDesce
nding).setOnClickListener {
                    initSortStart()
                }

popupWindow3.contentView.findViewById<RadioButton>(R.id.radioStartAscend
ing).setOnClickListener {
                    initSortStartDescending()
                }

popupWindow3.contentView.findViewById<RadioButton>(R.id.radioEndDescen
ding).setOnClickListener {
                    initSortEnd()
                }

popupWindow3.contentView.findViewById<RadioButton>(R.id.radioEndAscendi
ng).setOnClickListener {
                    initSortEndDescending()
                }

popupWindow3.contentView.findViewById<RadioButton>(R.id.radioCRSDesce
nding).setOnClickListener {
                    initSortCRS()
                }

popupWindow3.contentView.findViewById<RadioButton>(R.id.radioCRSAscend
ing).setOnClickListener {
                    initSortCRSDescending()
                }

popupWindow3.contentView.findViewById<RadioButton>(R.id.radioCheckDesc
ending).setOnClickListener {
                    initSortCheck()
                }

popupWindow3.contentView.findViewById<RadioButton>(R.id.radioCheckAsce
nding).setOnClickListener {
                    initSortCheckDescending()
                }

popupWindow3.contentView.findViewById<RadioButton>(R.id.radioStatusDesc
ending).setOnClickListener {
```

```
                        initSortStatus()
                    }

popupWindow3.contentView.findViewById<RadioButton>(R.id.radioStatusAsce
nding).setOnClickListener {
                        initSortStatusDescending()
                    }
                    popupWindow.dismiss()
                }

                popupView.setOnClickListener { popupWindow.dismiss() }
                popupView.setOnTouchListener { v, event ->
                    popupWindow.dismiss()
                    true
                }


            }
        }
    }
}

    override fun onItemLongClick(view: View?, position: Int) {
        val db: FirebaseFirestore = FirebaseFirestore.getInstance()
        val collection: CollectionReference = db.collection("Aircraft revision")
        val query: Query = collection
        query.get().addOnCompleteListener { task ->
            if (task.isSuccessful) {
                val mData: ArrayList<AircraftModel> = ArrayList()
                for (document in task.result) {
                    val taskItem: AircraftModel =
document.toObject(AircraftModel::class.java)
                    mData.add(taskItem)
                }
            }
        }
    }

    private fun searchBar() {
        val search : EditText = findViewById(R.id.search)
        val db : FirebaseFirestore= FirebaseFirestore.getInstance()
        val rvAircraft= findViewById<RecyclerView>(R.id.recyclerViewAircraftList)
        db.collection("Aircraft revision")
                .whereEqualTo("Aircraft",
search.text.toString()).get().addOnSuccessListener { documents ->
                    val mData: ArrayList<AircraftModel> = ArrayList()
                    for (document in documents) {
                        val taskItem: AircraftModel =
document.toObject(AircraftModel::class.java)
                        mData.add(taskItem)
                }
                mAdapter = AircraftAdapter(mData)
                rvAircraft.adapter = mAdapter
                mAdapter!!.setOnItemClickListener(this)
                mAdapter!!.setOnItemLongClickListener(this)
                mAdapter!!.notifyDataSetChanged()
            }
    }

    private fun hideKeyboard(view: View) {
        val inputMethodManager =
getSystemService(INPUT_METHOD_SERVICE) as InputMethodManager
        inputMethodManager.hideSoftInputFromWindow(view.windowToken, 0)
    }

}
```

A.5. Crew home activity

```
class CrewHomeActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {

        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_home_crew)

        val menu : ImageButton = findViewById(R.id.btn_menu)
        menu.visibility = View.GONE

        val btnCrew= findViewById<ImageButton>(R.id.btn_crew)
        btnCrew.setOnClickListener { openActivity1() }

        val btnMail = findViewById<ImageButton>(R.id.btn_message)
        btnMail.setOnClickListener {
            val intent = Intent(this, MailActivity::class.java)
            startActivity(intent) }

        val calendar = Calendar.getInstance()
        val date = calendar.time

        val stringYear= DateFormat.format("yy", date) as String
        val stringMonth = DateFormat.format("MM", date) as String
        val stringNumber = DateFormat.format("dd", date) as String
        val stringDay = DateFormat.format("EEEE", date) as String
        val textDay = findViewById<TextView>(R.id.tool_title)
        textDay.text = ("$stringDay , $stringNumber / $stringMonth / $stringYear")

        val back : ImageButton = findViewById(R.id.btn_back)
        back.setBackgroundResource(R.drawable.ic_baseline_logout_24)
        back.setOnClickListener {
            val builder = AlertDialog.Builder(this)
            builder.setTitle("Warning!").setMessage("You sure, that you want to
logout?")
            builder.setPositiveButton(
                    "Yes"
            ) { dialog, id ->
                val i = Intent(this, WelcomeActivity::class.java)
                startActivity(i)
            }
            builder.setNegativeButton(
                    "No"
            ) { dialog, id -> dialog.cancel() }
            val alert = builder.create()
            alert.show()
        }

        val edit : ImageButton = findViewById(R.id.btn_edit_info)
        edit.setOnClickListener {
            val intent = Intent(this, EditUserInformation::class.java)
            startActivity(intent)
        }

        displayEmailJob()

    }

    private fun displayEmailJob() {
        val user = Firebase.auth.currentUser
        val email= user.email
        val emailTextView = findViewById<TextView>(R.id.mailTextView)
        val jobTextView = findViewById<TextView>(R.id.loginAsTextView)
        val nameTextView = findViewById<TextView>(R.id.welcomeTextView)
        emailTextView.text = ("$email")
        val db = FirebaseFirestore.getInstance()
        val df: DocumentReference = db.collection("Users").document(email)
        df.get().addOnSuccessListener { documentSnapshot ->
            val job = documentSnapshot.getString("Job")
            jobTextView.text = ("$job")
            val name = documentSnapshot.getString("Name")
            nameTextView.text= ("$name")
        }

        val ref = FirebaseStorage.getInstance().reference.child("profile/$email")
        val profile : CardView = this.findViewById(R.id.card_profile)
        val imageView = profile.findViewById<ImageView>(R.id.profilePicture)
        imageView.drawable
        ref.downloadUrl.addOnSuccessListener { Uri ->
            val imageURL = Uri.toString()
            Glide.with(this)
                .load(imageURL)
                .into(imageView)
        }.addOnFailureListener {
        }
    }

    private fun openActivity1() {
        val intent = Intent(this, FleetActivity::class.java)
        startActivity(intent)
    }
}
```

A.7. Damage activity

```
class DamageActivity : AppCompatActivity() , MyItemClickListener,
MyItemLongClickListener {

    private var mAdapter: DamageAdapter? = null

    override fun onCreate(savedInstanceState: Bundle?) {

        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_damages)

        val title: TextView = findViewById(R.id.tool_title)
        title.text = "DAMAGES"

        val back = findViewById<View>(R.id.btn_back) as ImageButton
        back.setOnClickListener {
            val intent = Intent(this, CheckListActivity::class.java)
            startActivity(intent)
            finish() }

        initView()
        initData()

        showMenu()

        val btn_search : Button = findViewById(R.id.button_search)
        btn_search.onFocusChangeListener = View.OnFocusChangeListener { a,
hasFocus ->
            if (!hasFocus) {
                btn_search.setHint("Search")
                hideKeyboard(a)
            } else btn_search.hint = ""
        }

        btn_search.setOnClickListener {
            val search : EditText = findViewById(R.id.search)
            if (search.text.toString().isEmpty()) {

            } else {
                searchBar()
            }
        }

        val btn_reset: Button = findViewById(R.id.button_reset)
        btn_reset.setOnClickListener {
            initData()
        }

    }

    private fun searchBar() {
        val SHARED_PREF_AIRCRAFT = "aircraft"
        val KEY_AIRCRAFT = "key_aircraft"
        val sp = getSharedPreferences(SHARED_PREF_AIRCRAFT,
MODE_PRIVATE)
        val aircraft = sp.getString(KEY_AIRCRAFT, null)

        val SHARED_PREF_START = "start"
        val KEY_START = "key_start"
        val sp1 = getSharedPreferences(SHARED_PREF_START,
MODE_PRIVATE)
        val start = sp1.getString(KEY_START, null)

        val SHARED_PREF_check = "check"
        val KEY_check = "key_check"
        val sp2 = getSharedPreferences(SHARED_PREF_check,
MODE_PRIVATE)
        val check = sp2.getString(KEY_check, null)

        val items1 = start?.split("/".toRegex())?.toTypedArray()
        val day = items1?.get(0)
        val month = items1?.get(1)
        val year = items1?.get(2)

        val id = aircraft + "_" + check + "_" + day + month + year
        val search : EditText = findViewById(R.id.search)
        val db : FirebaseFirestore= FirebaseFirestore.getInstance()
        val rvDamage=
findViewById<RecyclerView>(R.id.recyclerViewDamagesList)
        db.collection("Aircraft revision").document(id).collection("New
damages")
                .whereEqualTo("ID",
search.text.toString()).get().addOnSuccessListener { documents ->
                    val mData: ArrayList<DamageModel> = ArrayList()
                    for (document in documents) {
```

```
                    val taskItem: DamageModel =
document.toObject(DamageModel::class.java)
                    mData.add(taskItem)
                }
                mAdapter = DamageAdapter(mData)
                rvDamage.adapter = mAdapter
                mAdapter!!.setOnItemClickListener(this)
                mAdapter!!.setOnItemLongClickListener(this)
                mAdapter!!.notifyDataSetChanged()
            }
    }

    private fun hideKeyboard(view: View) {
        val inputMethodManager =
getSystemService(INPUT_METHOD_SERVICE) as InputMethodManager
        inputMethodManager.hideSoftInputFromWindow(view.windowToken, 0)
    }

    override fun onBackPressed() {
        super.onBackPressed()
        val intent = Intent(this, CheckListActivity::class.java)
        startActivity(intent)
        finish()
    }

    private fun initView() {
        val rvDamages =
findViewById<RecyclerView>(R.id.recyclerViewDamagesList)
        rvDamages.layoutManager = LinearLayoutManager(this)
        rvDamages.itemAnimator = DefaultItemAnimator()
    }

    private fun initData() {

        val SHARED_PREF_AIRCRAFT = "aircraft"
        val KEY_AIRCRAFT = "key_aircraft"
        val sp = getSharedPreferences(SHARED_PREF_AIRCRAFT,
MODE_PRIVATE)
        val aircraft = sp.getString(KEY_AIRCRAFT, null)

        val SHARED_PREF_START = "start"
        val KEY_START = "key_start"
        val sp1 = getSharedPreferences(SHARED_PREF_START,
MODE_PRIVATE)
        val start = sp1.getString(KEY_START, null)

        val SHARED_PREF_check = "check"
        val KEY_check = "key_check"
        val sp2 = getSharedPreferences(SHARED_PREF_check,
MODE_PRIVATE)
```

```
        val check = sp2.getString(KEY_check, null)

        val items1 = start?.split("/".toRegex())?.toTypedArray()
        val day = items1?.get(0)
        val month = items1?.get(1)
        val year = items1?.get(2)

        val id = aircraft + "_" + check + "_" + day + month + year
        val db : FirebaseFirestore= FirebaseFirestore.getInstance()
        val collection: CollectionReference = db.collection("Aircraft
revision").document(id).collection("New damages")
        val query : Query = collection
        val rvDamage=
findViewById<RecyclerView>(R.id.recyclerViewDamagesList)
        query.get().addOnCompleteListener { task ->
            if (task.isSuccessful) {
                val mData: java.util.ArrayList<DamageModel> = java.util.ArrayList()
                for (document in task.result) {
                    val taskItem: DamageModel =
document.toObject(DamageModel::class.java)
                    mData.add(taskItem)
                }
                mAdapter = DamageAdapter(mData)
                rvDamage.adapter = mAdapter
                mAdapter!!.setOnItemClickListener(this)
                mAdapter!!.setOnItemLongClickListener(this)
                mAdapter!!.notifyDataSetChanged()
            }
        }

    }

    private fun initDataCertified() {
        val SHARED_PREF_AIRCRAFT = "aircraft"
        val KEY_AIRCRAFT = "key_aircraft"
        val sp = getSharedPreferences(SHARED_PREF_AIRCRAFT,
MODE_PRIVATE)
        val aircraft = sp.getString(KEY_AIRCRAFT, null)

        val SHARED_PREF_START = "start"
        val KEY_START = "key_start"
        val sp1 = getSharedPreferences(SHARED_PREF_START,
MODE_PRIVATE)
        val start = sp1.getString(KEY_START, null)

        val SHARED_PREF_check = "check"
        val KEY_check = "key_check"
        val sp2 = getSharedPreferences(SHARED_PREF_check,
MODE_PRIVATE)
        val check = sp2.getString(KEY_check, null)
        val items1 = start?.split("/".toRegex())?.toTypedArray()
        val day = items1?.get(0)
        val month = items1?.get(1)
        val year = items1?.get(2)

        val id = aircraft + "_" + check + "_" + day + month + year
        val db : FirebaseFirestore= FirebaseFirestore.getInstance()
        val rvDamage=
findViewById<RecyclerView>(R.id.recyclerViewDamagesList)
        db.collection("Aircraft revision")
            .document(id).collection("New damages")
            .whereEqualTo("Certified", "Yes").get().addOnSuccessListener {
documents ->
                val mData: ArrayList<DamageModel> = ArrayList()
                for (document in documents) {
                    val taskItem: DamageModel =
document.toObject(DamageModel::class.java)
                    mData.add(taskItem)
                }
                mAdapter = DamageAdapter(mData)
                rvDamage.adapter = mAdapter
                mAdapter!!.setOnItemClickListener(this)
                mAdapter!!.setOnItemLongClickListener(this)
                mAdapter!!.notifyDataSetChanged()
            }

    }

    private fun initDataChecked() {
        val SHARED_PREF_AIRCRAFT = "aircraft"
        val KEY_AIRCRAFT = "key_aircraft"
        val sp = getSharedPreferences(SHARED_PREF_AIRCRAFT,
MODE_PRIVATE)
        val aircraft = sp.getString(KEY_AIRCRAFT, null)

        val SHARED_PREF_START = "start"
        val KEY_START = "key_start"
        val sp1 = getSharedPreferences(SHARED_PREF_START,
MODE_PRIVATE)
        val start = sp1.getString(KEY_START, null)

        val SHARED_PREF_check = "check"
        val KEY_check = "key_check"
        val sp2 = getSharedPreferences(SHARED_PREF_check,
MODE_PRIVATE)
        val check = sp2.getString(KEY_check, null)

        val items1 = start?.split("/".toRegex())?.toTypedArray()
        val day = items1?.get(0)
        val month = items1?.get(1)
        val year = items1?.get(2)

        val id = aircraft + "_" + check + "_" + day + month + year
        val db : FirebaseFirestore= FirebaseFirestore.getInstance()
        val rvDamage=
findViewById<RecyclerView>(R.id.recyclerViewDamagesList)
        db.collection("Aircraft revision")
                .document(id).collection("New damages")
                .whereEqualTo("Checked", "Yes").get().addOnSuccessListener {
documents ->
                    val mData: ArrayList<DamageModel> = ArrayList()
                    for (document in documents) {
                        val taskItem: DamageModel =
document.toObject(DamageModel::class.java)
                        mData.add(taskItem)
                    }
                    mAdapter = DamageAdapter(mData)
                    rvDamage.adapter = mAdapter
                    mAdapter!!.setOnItemClickListener(this)
                    mAdapter!!.setOnItemLongClickListener(this)
                    mAdapter!!.notifyDataSetChanged()
                }

    }

    private fun initDataNotChecked() {

        val SHARED_PREF_AIRCRAFT = "aircraft"
        val KEY_AIRCRAFT = "key_aircraft"
        val sp = getSharedPreferences(SHARED_PREF_AIRCRAFT,
MODE_PRIVATE)
        val aircraft = sp.getString(KEY_AIRCRAFT, null)

        val SHARED_PREF_START = "start"
        val KEY_START = "key_start"
        val sp1 = getSharedPreferences(SHARED_PREF_START,
MODE_PRIVATE)
        val start = sp1.getString(KEY_START, null)

        val SHARED_PREF_check = "check"
        val KEY_check = "key_check"
        val sp2 = getSharedPreferences(SHARED_PREF_check,
MODE_PRIVATE)
        val check = sp2.getString(KEY_check, null)

        val items1 = start?.split("/".toRegex())?.toTypedArray()
        val day = items1?.get(0)
        val month = items1?.get(1)
        val year = items1?.get(2)
        val id = aircraft + "_" + check + "_" + day + month + year
        val db : FirebaseFirestore= FirebaseFirestore.getInstance()
        val rvDamage=
findViewById<RecyclerView>(R.id.recyclerViewDamagesList)
        db.collection("Aircraft revision")
                .document(id).collection("New damages")
                .whereEqualTo("Checked", "No").get().addOnSuccessListener {
documents ->
                    val mData: ArrayList<DamageModel> = ArrayList()
                    for (document in documents) {
                        val taskItem: DamageModel =
document.toObject(DamageModel::class.java)
                        mData.add(taskItem)
                    }
                    mAdapter = DamageAdapter(mData)
                    rvDamage.adapter = mAdapter
                    mAdapter!!.setOnItemClickListener(this)
                    mAdapter!!.setOnItemLongClickListener(this)
                    mAdapter!!.notifyDataSetChanged()
                }

    }

    private fun initDataNotCertified() {

        val SHARED_PREF_AIRCRAFT = "aircraft"
        val KEY_AIRCRAFT = "key_aircraft"
        val sp = getSharedPreferences(SHARED_PREF_AIRCRAFT,
MODE_PRIVATE)
        val aircraft = sp.getString(KEY_AIRCRAFT, null)

        val SHARED_PREF_START = "start"
        val KEY_START = "key_start"
        val sp1 = getSharedPreferences(SHARED_PREF_START,
MODE_PRIVATE)
        val start = sp1.getString(KEY_START, null)

        val SHARED_PREF_check = "check"
        val KEY_check = "key_check"
        val sp2 = getSharedPreferences(SHARED_PREF_check,
MODE_PRIVATE)
        val check = sp2.getString(KEY_check, null)

        val items1 = start?.split("/".toRegex())?.toTypedArray()
        val day = items1?.get(0)
        val month = items1?.get(1)
        val year = items1?.get(2)

        val id = aircraft + "_" + check + "_" + day + month + year
        val db : FirebaseFirestore= FirebaseFirestore.getInstance()
        val rvDamage=
findViewById<RecyclerView>(R.id.recyclerViewDamagesList)
        db.collection("Aircraft revision")
            .document(id).collection("New damages")
            .whereEqualTo("Certified", "No").get().addOnSuccessListener {
documents ->
                val mData: ArrayList<DamageModel> = ArrayList()
                for (document in documents) {
                    val taskItem: DamageModel =
document.toObject(DamageModel::class.java)
                    mData.add(taskItem)
                }
                mAdapter = DamageAdapter(mData)
                rvDamage.adapter = mAdapter
                mAdapter!!.setOnItemClickListener(this)
                mAdapter!!.setOnItemLongClickListener(this)
                mAdapter!!.notifyDataSetChanged()
            }
    }

    private fun initDataA() {
        val SHARED_PREF_AIRCRAFT = "aircraft"
        val KEY_AIRCRAFT = "key_aircraft"
        val sp = getSharedPreferences(SHARED_PREF_AIRCRAFT,
MODE_PRIVATE)
        val aircraft = sp.getString(KEY_AIRCRAFT, null)

        val SHARED_PREF_START = "start"
        val KEY_START = "key_start"
        val sp1 = getSharedPreferences(SHARED_PREF_START,
MODE_PRIVATE)
        val start = sp1.getString(KEY_START, null)

        val SHARED_PREF_check = "check"
        val KEY_check = "key_check"
        val sp2 = getSharedPreferences(SHARED_PREF_check,
MODE_PRIVATE)
        val check = sp2.getString(KEY_check, null)

        val items1 = start?.split("/".toRegex())?.toTypedArray()
        val day = items1?.get(0)
        val month = items1?.get(1)
        val year = items1?.get(2)

        val id = aircraft + "_" + check + "_" + day + month + year
        val db : FirebaseFirestore= FirebaseFirestore.getInstance()
        val rvDamage=
findViewById<RecyclerView>(R.id.recyclerViewDamagesList)
        db.collection("Aircraft revision")
```

```
                .document(id).collection("New damages")
                .whereEqualTo("Category", "A").get().addOnSuccessListener {
documents ->
                    val mData: ArrayList<DamageModel> = ArrayList()
                    for (document in documents) {
                        val taskItem: DamageModel =
document.toObject(DamageModel::class.java)
                        mData.add(taskItem)
                    }
                    mAdapter = DamageAdapter(mData)
                    rvDamage.adapter = mAdapter
                    mAdapter!!.setOnItemClickListener(this)
                    mAdapter!!.setOnItemLongClickListener(this)
                    mAdapter!!.notifyDataSetChanged()
                }

    }

    private fun initDataB() {
        val SHARED_PREF_AIRCRAFT = "aircraft"
        val KEY_AIRCRAFT = "key_aircraft"
        val sp = getSharedPreferences(SHARED_PREF_AIRCRAFT,
MODE_PRIVATE)
        val aircraft = sp.getString(KEY_AIRCRAFT, null)

        val SHARED_PREF_START = "start"
        val KEY_START = "key_start"
        val sp1 = getSharedPreferences(SHARED_PREF_START,
MODE_PRIVATE)
        val start = sp1.getString(KEY_START, null)

        val SHARED_PREF_check = "check"
        val KEY_check = "key_check"
        val sp2 = getSharedPreferences(SHARED_PREF_check,
MODE_PRIVATE)
        val check = sp2.getString(KEY_check, null)

        val items1 = start?.split("/".toRegex())?.toTypedArray()
        val day = items1?.get(0)
        val month = items1?.get(1)
        val year = items1?.get(2)

        val id = aircraft + "_" + check + "_" + day + month + year
        val db : FirebaseFirestore= FirebaseFirestore.getInstance()
        val rvDamage=
findViewById<RecyclerView>(R.id.recyclerViewDamagesList)
        db.collection("Aircraft revision")
                .document(id).collection("New damages")
                .whereEqualTo("Category", "B").get().addOnSuccessListener {
documents ->
```

```
                val mData: ArrayList<DamageModel> = ArrayList()
                for (document in documents) {
                    val taskItem: DamageModel =
document.toObject(DamageModel::class.java)
                    mData.add(taskItem)
                }
                mAdapter = DamageAdapter(mData)
                rvDamage.adapter = mAdapter
                mAdapter!!.setOnItemClickListener(this)
                mAdapter!!.setOnItemLongClickListener(this)
                mAdapter!!.notifyDataSetChanged()
            }

    }

    private fun initDataC() {
        val SHARED_PREF_AIRCRAFT = "aircraft"
        val KEY_AIRCRAFT = "key_aircraft"
        val sp = getSharedPreferences(SHARED_PREF_AIRCRAFT,
MODE_PRIVATE)
        val aircraft = sp.getString(KEY_AIRCRAFT, null)

        val SHARED_PREF_START = "start"
        val KEY_START = "key_start"
        val sp1 = getSharedPreferences(SHARED_PREF_START,
MODE_PRIVATE)
        val start = sp1.getString(KEY_START, null)

        val SHARED_PREF_check = "check"
        val KEY_check = "key_check"
        val sp2 = getSharedPreferences(SHARED_PREF_check,
MODE_PRIVATE)
        val check = sp2.getString(KEY_check, null)

        val items1 = start?.split("/".toRegex())?.toTypedArray()
        val day = items1?.get(0)
        val month = items1?.get(1)
        val year = items1?.get(2)

        val id = aircraft + "_" + check + "_" + day + month + year
        val db : FirebaseFirestore= FirebaseFirestore.getInstance()
        val rvDamage=
findViewById<RecyclerView>(R.id.recyclerViewDamagesList)
        db.collection("Aircraft revision")
            .document(id).collection("New damages")
            .whereEqualTo("Category", "C").get().addOnSuccessListener {
documents ->
                val mData: ArrayList<DamageModel> = ArrayList()
                for (document in documents) {
                    val taskItem: DamageModel =
document.toObject(DamageModel::class.java)
                        mData.add(taskItem)
                    }
                    mAdapter = DamageAdapter(mData)
                    rvDamage.adapter = mAdapter
                    mAdapter!!.setOnItemClickListener(this)
                    mAdapter!!.setOnItemLongClickListener(this)
                    mAdapter!!.notifyDataSetChanged()
                }

    }

    private fun initDataPending() {
        val SHARED_PREF_AIRCRAFT = "aircraft"
        val KEY_AIRCRAFT = "key_aircraft"
        val sp = getSharedPreferences(SHARED_PREF_AIRCRAFT,
MODE_PRIVATE)
        val aircraft = sp.getString(KEY_AIRCRAFT, null)

        val SHARED_PREF_START = "start"
        val KEY_START = "key_start"
        val sp1 = getSharedPreferences(SHARED_PREF_START,
MODE_PRIVATE)
        val start = sp1.getString(KEY_START, null)

        val SHARED_PREF_check = "check"
        val KEY_check = "key_check"
        val sp2 = getSharedPreferences(SHARED_PREF_check,
MODE_PRIVATE)
        val check = sp2.getString(KEY_check, null)

        val items1 = start?.split("/".toRegex())?.toTypedArray()
        val day = items1?.get(0)
        val month = items1?.get(1)
        val year = items1?.get(2)

        val id = aircraft + "_" + check + "_" + day + month + year
        val db : FirebaseFirestore= FirebaseFirestore.getInstance()
        val rvDamage=
findViewById<RecyclerView>(R.id.recyclerViewDamagesList)
        db.collection("Aircraft revision")
                .document(id).collection("New damages")
                .whereEqualTo("RB", "Pending").get().addOnSuccessListener {
documents ->
                    val mData: ArrayList<DamageModel> = ArrayList()
                    for (document in documents) {
                        val taskItem: DamageModel =
document.toObject(DamageModel::class.java)
                        mData.add(taskItem)
                    }
                mAdapter = DamageAdapter(mData)
                rvDamage.adapter = mAdapter
                mAdapter!!.setOnItemClickListener(this)
                mAdapter!!.setOnItemLongClickListener(this)
                mAdapter!!.notifyDataSetChanged()
            }
    }

    private fun initDataDone() {
        val SHARED_PREF_AIRCRAFT = "aircraft"
        val KEY_AIRCRAFT = "key_aircraft"
        val sp = getSharedPreferences(SHARED_PREF_AIRCRAFT,
MODE_PRIVATE)
        val aircraft = sp.getString(KEY_AIRCRAFT, null)

        val SHARED_PREF_START = "start"
        val KEY_START = "key_start"
        val sp1 = getSharedPreferences(SHARED_PREF_START,
MODE_PRIVATE)
        val start = sp1.getString(KEY_START, null)

        val SHARED_PREF_check = "check"
        val KEY_check = "key_check"
        val sp2 = getSharedPreferences(SHARED_PREF_check,
MODE_PRIVATE)
        val check = sp2.getString(KEY_check, null)

        val items1 = start?.split("/".toRegex())?.toTypedArray()
        val day = items1?.get(0)
        val month = items1?.get(1)
        val year = items1?.get(2)

        val id = aircraft + "_" + check + "_" + day + month + year
        val db : FirebaseFirestore= FirebaseFirestore.getInstance()
        val rvDamage=
findViewById<RecyclerView>(R.id.recyclerViewDamagesList)
        db.collection("Aircraft revision")
            .document(id).collection("New damages")
            .whereEqualTo("RB", "Done").get().addOnSuccessListener {
documents ->
                val mData: ArrayList<DamageModel> = ArrayList()
                for (document in documents) {
                    val taskItem: DamageModel =
document.toObject(DamageModel::class.java)
                    mData.add(taskItem)
                }
                mAdapter = DamageAdapter(mData)
                rvDamage.adapter = mAdapter
                mAdapter!!.setOnItemClickListener(this)
                mAdapter!!.setOnItemLongClickListener(this)
                mAdapter!!.notifyDataSetChanged()
            }

    }

    private fun initDataSDT() {
        val SHARED_PREF_AIRCRAFT = "aircraft"
        val KEY_AIRCRAFT = "key_aircraft"
        val sp = getSharedPreferences(SHARED_PREF_AIRCRAFT,
MODE_PRIVATE)
        val aircraft = sp.getString(KEY_AIRCRAFT, null)

        val SHARED_PREF_START = "start"
        val KEY_START = "key_start"
        val sp1 = getSharedPreferences(SHARED_PREF_START,
MODE_PRIVATE)
        val start = sp1.getString(KEY_START, null)

        val SHARED_PREF_check = "check"
        val KEY_check = "key_check"
        val sp2 = getSharedPreferences(SHARED_PREF_check,
MODE_PRIVATE)
        val check = sp2.getString(KEY_check, null)

        val items1 = start?.split("/".toRegex())?.toTypedArray()
        val day = items1?.get(0)
        val month = items1?.get(1)
        val year = items1?.get(2)

        val id = aircraft + "_" + check + "_" + day + month + year
        val db : FirebaseFirestore= FirebaseFirestore.getInstance()
        val rvDamage=
findViewById<RecyclerView>(R.id.recyclerViewDamagesList)
        db.collection("Aircraft revision")
            .document(id).collection("New damages")
            .whereEqualTo("RB", "N/A").get().addOnSuccessListener {
documents ->
                val mData: ArrayList<DamageModel> = ArrayList()
                for (document in documents) {
                    val taskItem: DamageModel =
document.toObject(DamageModel::class.java)
                    mData.add(taskItem)
                }
                mAdapter = DamageAdapter(mData)
                rvDamage.adapter = mAdapter
                mAdapter!!.setOnItemClickListener(this)
                mAdapter!!.setOnItemLongClickListener(this)
                mAdapter!!.notifyDataSetChanged()
            }
    }

    private fun initSortClosing() {
        val SHARED_PREF_AIRCRAFT = "aircraft"
        val KEY_AIRCRAFT = "key_aircraft"
        val sp = getSharedPreferences(SHARED_PREF_AIRCRAFT,
MODE_PRIVATE)
        val aircraft = sp.getString(KEY_AIRCRAFT, null)

        val SHARED_PREF_START = "start"
        val KEY_START = "key_start"
        val sp1 = getSharedPreferences(SHARED_PREF_START,
MODE_PRIVATE)
        val start = sp1.getString(KEY_START, null)

        val SHARED_PREF_check = "check"
        val KEY_check = "key_check"
        val sp2 = getSharedPreferences(SHARED_PREF_check,
MODE_PRIVATE)
        val check = sp2.getString(KEY_check, null)

        val items1 = start?.split("/".toRegex())?.toTypedArray()
        val day = items1?.get(0)
        val month = items1?.get(1)
        val year = items1?.get(2)

        val id = aircraft + "_" + check + "_" + day + month + year
        val db : FirebaseFirestore= FirebaseFirestore.getInstance()
        val rvDamage=
findViewById<RecyclerView>(R.id.recyclerViewDamagesList)
        db.collection("Aircraft revision")
            .document(id).collection("New
damages").get().addOnSuccessListener { documents ->
                val mData: ArrayList<DamageModel> = ArrayList()
                for (document in documents) {
                    val taskItem: DamageModel =
document.toObject(DamageModel::class.java)
                    mData.add(taskItem)
                }
                mData.sortBy {it.timestamp_closing}
                mAdapter = DamageAdapter(mData)
                rvDamage.adapter = mAdapter
                mAdapter!!.setOnItemClickListener(this)
                mAdapter!!.setOnItemLongClickListener(this)
                mAdapter!!.notifyDataSetChanged()
            }


    }
```

```
  private fun initSortClosingDescending() {

      val SHARED_PREF_AIRCRAFT = "aircraft"
      val KEY_AIRCRAFT = "key_aircraft"
      val sp = getSharedPreferences(SHARED_PREF_AIRCRAFT,
MODE_PRIVATE)
      val aircraft = sp.getString(KEY_AIRCRAFT, null)

      val SHARED_PREF_START = "start"
      val KEY_START = "key_start"
      val sp1 = getSharedPreferences(SHARED_PREF_START,
MODE_PRIVATE)
      val start = sp1.getString(KEY_START, null)

      val SHARED_PREF_check = "check"
      val KEY_check = "key_check"
      val sp2 = getSharedPreferences(SHARED_PREF_check,
MODE_PRIVATE)
      val check = sp2.getString(KEY_check, null)

      val items1 = start?.split("/".toRegex())?.toTypedArray()
      val day = items1?.get(0)
      val month = items1?.get(1)
      val year = items1?.get(2)

      val id = aircraft + "_" + check + "_" + day + month + year
      val db : FirebaseFirestore= FirebaseFirestore.getInstance()
      val rvDamage=
findViewById<RecyclerView>(R.id.recyclerViewDamagesList)
      db.collection("Aircraft revision")
          .document(id).collection("New
damages").get().addOnSuccessListener { documents ->
              val mData: ArrayList<DamageModel> = ArrayList()
              for (document in documents) {
                  val taskItem: DamageModel =
document.toObject(DamageModel::class.java)
                  mData.add(taskItem)
              }
              mData.sortByDescending {it.timestamp_closing}
              mAdapter = DamageAdapter(mData)
              rvDamage.adapter = mAdapter
              mAdapter!!.setOnItemClickListener(this)
              mAdapter!!.setOnItemLongClickListener(this)
              mAdapter!!.notifyDataSetChanged()
          }
  }

  private fun initSortEntry() {

      val SHARED_PREF_AIRCRAFT = "aircraft"
```

```
        val KEY_AIRCRAFT = "key_aircraft"
        val sp = getSharedPreferences(SHARED_PREF_AIRCRAFT,
MODE_PRIVATE)
        val aircraft = sp.getString(KEY_AIRCRAFT, null)

        val SHARED_PREF_START = "start"
        val KEY_START = "key_start"
        val sp1 = getSharedPreferences(SHARED_PREF_START,
MODE_PRIVATE)
        val start = sp1.getString(KEY_START, null)

        val SHARED_PREF_check = "check"
        val KEY_check = "key_check"
        val sp2 = getSharedPreferences(SHARED_PREF_check,
MODE_PRIVATE)
        val check = sp2.getString(KEY_check, null)

        val items1 = start?.split("/".toRegex())?.toTypedArray()
        val day = items1?.get(0)
        val month = items1?.get(1)
        val year = items1?.get(2)

        val id = aircraft + "_" + check + "_" + day + month + year
        val db : FirebaseFirestore= FirebaseFirestore.getInstance()
        val rvDamage=
findViewById<RecyclerView>(R.id.recyclerViewDamagesList)
        db.collection("Aircraft revision")
            .document(id).collection("New
damages").get().addOnSuccessListener { documents ->
                val mData: ArrayList<DamageModel> = ArrayList()
                for (document in documents) {
                    val taskItem: DamageModel =
document.toObject(DamageModel::class.java)
                    mData.add(taskItem)
                }
                mData.sortBy {it.timestamp_entry}
                mAdapter = DamageAdapter(mData)
                rvDamage.adapter = mAdapter
                mAdapter!!.setOnItemClickListener(this)
                mAdapter!!.setOnItemLongClickListener(this)
                mAdapter!!.notifyDataSetChanged()
            }


    }

    private fun initSortEntryDescending() {

        val SHARED_PREF_AIRCRAFT = "aircraft"
        val KEY_AIRCRAFT = "key_aircraft"
        val sp = getSharedPreferences(SHARED_PREF_AIRCRAFT,
MODE_PRIVATE)
        val aircraft = sp.getString(KEY_AIRCRAFT, null)

        val SHARED_PREF_START = "start"
        val KEY_START = "key_start"
        val sp1 = getSharedPreferences(SHARED_PREF_START,
MODE_PRIVATE)
        val start = sp1.getString(KEY_START, null)

        val SHARED_PREF_check = "check"
        val KEY_check = "key_check"
        val sp2 = getSharedPreferences(SHARED_PREF_check,
MODE_PRIVATE)
        val check = sp2.getString(KEY_check, null)

        val items1 = start?.split("/".toRegex())?.toTypedArray()
        val day = items1?.get(0)
        val month = items1?.get(1)
        val year = items1?.get(2)

        val id = aircraft + "_" + check + "_" + day + month + year
        val db : FirebaseFirestore= FirebaseFirestore.getInstance()
        val rvDamage=
findViewById<RecyclerView>(R.id.recyclerViewDamagesList)
        db.collection("Aircraft revision")
                .document(id).collection("New
damages").get().addOnSuccessListener { documents ->
                    val mData: ArrayList<DamageModel> = ArrayList()
                    for (document in documents) {
                        val taskItem: DamageModel =
document.toObject(DamageModel::class.java)
                        mData.add(taskItem)
                    }
                    mData.sortByDescending {it.timestamp_entry}
                    mAdapter = DamageAdapter(mData)
                    rvDamage.adapter = mAdapter
                    mAdapter!!.setOnItemClickListener(this)
                    mAdapter!!.setOnItemLongClickListener(this)
                    mAdapter!!.notifyDataSetChanged()
                }
    }

    private fun showMenu() {
        val button : ImageButton = findViewById(R.id.btn_menu)
        button.setOnClickListener {
            val user = FirebaseAuth.getInstance().currentUser
            val userEmail = user.email
            val db = FirebaseFirestore.getInstance()
            val df = db.collection("Users").document(userEmail)
```

```
          df.get().addOnSuccessListener { documentSnapshot ->
             if (documentSnapshot.getString("Job") == "MRO engineer" ||
documentSnapshot.getString("Job") == "CAMO engineer"
                    || documentSnapshot.getString("Job") == "Total access
engineer") {
                val inflater = LayoutInflater.from(this)
                val popupView: View = inflater.inflate(R.layout.menu_admin, null)
                val focusable = true
                val popupWindow = PopupWindow(popupView, 500, 1000,
focusable)
                button.x.toInt()
                button.y.toInt()
                popupWindow.showAsDropDown(button)


popupWindow.contentView.findViewById<LinearLayout>(R.id.add).visibility =
GONE


popupWindow.contentView.findViewById<LinearLayout>(R.id.linear_showAll).s
etOnClickListener {

                    val inflater2 = LayoutInflater.from(this)
                    val popupView2: View =
inflater2.inflate(R.layout.menu_damages, null)
                    val popupWindow2 = PopupWindow(popupView2, 500, 1000,
true)

popupWindow2.showAsDropDown(popupWindow.contentView.findViewById<Li
nearLayout>(R.id.linear_showAll))


popupWindow2.contentView.findViewById<LinearLayout>(R.id.linear_all).setOn
ClickListener {
                        initData()
                        mAdapter!!.notifyDataSetChanged()
                    }

popupWindow2.contentView.findViewById<LinearLayout>(R.id.linear_show_cer
tified).setOnClickListener {
                        initDataCertified()
                        mAdapter!!.notifyDataSetChanged()
                    }

popupWindow2.contentView.findViewById<LinearLayout>(R.id.linear_showNot
_certified).setOnClickListener {
                        initDataNotCertified()
                        mAdapter!!.notifyDataSetChanged()
                    }
```

```
popupWindow2.contentView.findViewById<LinearLayout>(R.id.linear_show_cer
tified).setOnClickListener {
                        initDataCertified()
                        mAdapter!!.notifyDataSetChanged()
                    }

popupWindow2.contentView.findViewById<LinearLayout>(R.id.linear_showNot
_certified).setOnClickListener {
                        initDataNotCertified()
                        mAdapter!!.notifyDataSetChanged()
                    }

popupWindow2.contentView.findViewById<LinearLayout>(R.id.linear_show_ch
ecked).setOnClickListener {
                        initDataChecked()
                        mAdapter!!.notifyDataSetChanged()
                    }

popupWindow2.contentView.findViewById<LinearLayout>(R.id.linear_showNot
_checked).setOnClickListener {
                        initDataNotChecked()
                        mAdapter!!.notifyDataSetChanged()
                    }

                }

popupWindow.contentView.findViewById<LinearLayout>(R.id.sortBy).setOnClic
kListener {
                    val inflater3 = LayoutInflater.from(this)
                    val popupView3: View =
inflater3.inflate(R.layout.pop_up_sortby_amt, null)
                    val popupWindow3 = PopupWindow(popupView3, 500,
ViewGroup.LayoutParams.WRAP_CONTENT, true)
                    popupWindow3.showAsDropDown(button)
                    popupWindow.dismiss()

popupWindow3.contentView.findViewById<RadioButton>(R.id.radioCategoryA).
setOnClickListener {
                        initDataA()
                    }

popupWindow3.contentView.findViewById<RadioButton>(R.id.radioCategoryB).
setOnClickListener {
                        initDataB()
                    }

popupWindow3.contentView.findViewById<RadioButton>(R.id.radioCategoryC).
setOnClickListener {
                        initDataC()
```

```
            }

popupWindow3.contentView.findViewById<RadioButton>(R.id.radioSDTDone).
setOnClickListener {
                initDataDone()
            }

popupWindow3.contentView.findViewById<RadioButton>(R.id.radioSDTPendin
g).setOnClickListener {
                initDataPending()
            }

popupWindow3.contentView.findViewById<RadioButton>(R.id.radioSDT).setOn
ClickListener {
                initDataSDT()
            }

popupWindow3.contentView.findViewById<RadioButton>(R.id.radioClosingDes
cending).setOnClickListener {
                initSortClosing()
            }

popupWindow3.contentView.findViewById<RadioButton>(R.id.radioClosingAsc
ending).setOnClickListener {
                initSortClosingDescending()
            }

popupWindow3.contentView.findViewById<RadioButton>(R.id.radioStartDesce
nding).setOnClickListener {
                initSortEntry()
            }

popupWindow3.contentView.findViewById<RadioButton>(R.id.radioStartAscend
ing).setOnClickListener {
                initSortEntryDescending()
            }
        }

popupWindow.contentView.findViewById<LinearLayout>(R.id.advancedSearch)
.setOnClickListener {
            val inflater3 = LayoutInflater.from(this)
            val popupView3: View =
inflater3.inflate(R.layout.pop_window_search_damage, null)
            val popupWindow3 = PopupWindow(popupView3, 520,
ViewGroup.LayoutParams.WRAP_CONTENT, true)
            popupWindow3.showAsDropDown(button)
            popupWindow.dismiss()
        }

        popupView.setOnClickListener { popupWindow.dismiss() }
```

```
                popupView.setOnTouchListener { v, event ->
                    popupWindow.dismiss()
                    true
                }
            }

            if (documentSnapshot.getString("Job") == "Airplane maintenance
technician" ||
                    documentSnapshot.getString("Job") == "Administrator"){

                val inflater = LayoutInflater.from(this)
                val popupView: View = inflater.inflate(R.layout.menu_admin, null)
                val focusable = true
                val popupWindow = PopupWindow(popupView, 500, 1000,
focusable)
                button.x.toInt()
                button.y.toInt()
                popupWindow.showAsDropDown(button)

popupWindow.contentView.findViewById<TextView>(R.id.add_text).text =
"ADD DAMAGE"

popupWindow.contentView.findViewById<LinearLayout>(R.id.add).setOnClickLi
stener {
                    val intent = Intent(this, AddDamageActivity::class.java)
                    startActivity(intent)
                    finish()
                }

popupWindow.contentView.findViewById<LinearLayout>(R.id.linear_showAll).s
etOnClickListener {
                    val inflater2 = LayoutInflater.from(this)
                    val popupView2: View =
inflater2.inflate(R.layout.menu_damages, null)
                    val popupWindow2 = PopupWindow(popupView2, 500, 1000,
true)

popupWindow2.showAsDropDown(popupWindow.contentView.findViewById<Li
nearLayout>(R.id.linear_showAll))

popupWindow2.contentView.findViewById<LinearLayout>(R.id.linear_all).setOn
ClickListener {
                        initData()
                        mAdapter!!.notifyDataSetChanged()
                    }

popupWindow2.contentView.findViewById<LinearLayout>(R.id.linear_show_cer
tified).setOnClickListener {
                        initDataCertified()
                        mAdapter!!.notifyDataSetChanged()
                    }

popupWindow2.contentView.findViewById<LinearLayout>(R.id.linear_showNot
_certified).setOnClickListener {
                        initDataNotCertified()
                        mAdapter!!.notifyDataSetChanged()
                    }

popupWindow2.contentView.findViewById<LinearLayout>(R.id.linear_show_ch
ecked).setOnClickListener {
                        initDataChecked()
                        mAdapter!!.notifyDataSetChanged()
                    }

popupWindow2.contentView.findViewById<LinearLayout>(R.id.linear_showNot
_checked).setOnClickListener {
                        initDataNotChecked()
                        mAdapter!!.notifyDataSetChanged()
                    }

                }

popupWindow.contentView.findViewById<LinearLayout>(R.id.sortBy).setOnClic
kListener {
                    val inflater3 = LayoutInflater.from(this)
                    val popupView3: View =
inflater3.inflate(R.layout.pop_up_sortby_amt, null)
                    val popupWindow3 = PopupWindow(popupView3,
ViewGroup.LayoutParams.WRAP_CONTENT,
ViewGroup.LayoutParams.WRAP_CONTENT, true)
                    popupWindow3.showAsDropDown(button)
                    popupWindow.dismiss()

popupWindow3.contentView.findViewById<RadioButton>(R.id.radioCategoryA).
setOnClickListener {
                        initDataA()
                    }

popupWindow3.contentView.findViewById<RadioButton>(R.id.radioCategoryB).
setOnClickListener {
                        initDataB()
                    }

popupWindow3.contentView.findViewById<RadioButton>(R.id.radioCategoryC).
setOnClickListener {
                        initDataC()
                    }

popupWindow3.contentView.findViewById<RadioButton>(R.id.radioSDTDone).
setOnClickListener {
```

```
                        initDataDone()
                    }

popupWindow3.contentView.findViewById<RadioButton>(R.id.radioSDTPendin
g).setOnClickListener {
                        initDataPending()
                    }

popupWindow3.contentView.findViewById<RadioButton>(R.id.radioSDT).setOn
ClickListener {
                        initDataSDT()
                    }

popupWindow3.contentView.findViewById<RadioButton>(R.id.radioClosingDes
cending).setOnClickListener {
                        initSortClosing()
                    }

popupWindow3.contentView.findViewById<RadioButton>(R.id.radioClosingAsc
ending).setOnClickListener {
                        initSortClosingDescending()
                    }

popupWindow3.contentView.findViewById<RadioButton>(R.id.radioStartDesce
nding).setOnClickListener {
                        initSortEntry()
                    }

popupWindow3.contentView.findViewById<RadioButton>(R.id.radioStartAscend
ing).setOnClickListener {
                        initSortEntryDescending()
                    }
                }

popupWindow.contentView.findViewById<LinearLayout>(R.id.advancedSearch)
.setOnClickListener {
                    val inflater3 = LayoutInflater.from(this)
                    val popupView3: View =
inflater3.inflate(R.layout.pop_window_search_damage, null)
                    val popupWindow3 = PopupWindow(popupView3,
ViewGroup.LayoutParams.WRAP_CONTENT,
ViewGroup.LayoutParams.WRAP_CONTENT, true)
                    popupWindow3.showAsDropDown(button)
                    popupWindow.dismiss()
                }
                popupView.setOnClickListener { popupWindow.dismiss() }
                popupView.setOnTouchListener { v, event ->
                    popupWindow.dismiss()
                    true
                }
                }
            }
        }
    }

    override fun onItemClick(view: View?, position: Int) {
        val SHARED_PREF_AIRCRAFT = "aircraft"
        val KEY_AIRCRAFT = "key_aircraft"
        val sp = getSharedPreferences(SHARED_PREF_AIRCRAFT,
MODE_PRIVATE)
        val aircraft = sp.getString(KEY_AIRCRAFT, null)

        val SHARED_PREF_START = "start"
        val KEY_START = "key_start"
        val sp1 = getSharedPreferences(SHARED_PREF_START,
MODE_PRIVATE)
        val start = sp1.getString(KEY_START, null)

        val SHARED_PREF_check = "check"
        val KEY_check = "key_check"
        val sp2 = getSharedPreferences(SHARED_PREF_check,
MODE_PRIVATE)
        val check = sp2.getString(KEY_check, null)

        val items1 = start?.split("/".toRegex())?.toTypedArray()
        val day = items1?.get(0)
        val month = items1?.get(1)
        val year = items1?.get(2)
        val id = aircraft + "_" + check + "_" + day + month + year
        val db : FirebaseFirestore= FirebaseFirestore.getInstance()
        val collection: CollectionReference = db.collection("Aircraft
revision").document(id).collection("New damages")
        val query : Query = collection
        query.get().addOnCompleteListener { task ->
            if (task.isSuccessful) {
                val mData: ArrayList<DamageModel> = ArrayList()
                for (document in task.result) {
                    val taskItem: DamageModel =
document.toObject(DamageModel::class.java)
                    mData.add(taskItem)
                }
                val bean: DamageModel = mData[position - 1]
                val damage_id = bean.ID
                val category = bean.Category

                val user = FirebaseAuth.getInstance().currentUser
                val userEmail = user.email
                val db = FirebaseFirestore.getInstance()
                val df = db.collection("Users").document(userEmail)
                df.get().addOnSuccessListener { documentSnapshot ->
                if (documentSnapshot.getString("Job") == "MRO engineer"
                    || documentSnapshot.getString("Job") == "CAMO engineer"
                    || documentSnapshot.getString("Job") == "Total access
engineer") {
                    val inflater = LayoutInflater.from(this)
                    val popupView: View =
inflater.inflate(R.layout.pop_window_mro_damages, null)
                    val focusable = true
                    val popID : TextView = popupView.findViewById(R.id.tool_title)
                    popID.isAllCaps=false
                    popID.text = ("Damage  $damage_id")
                    val popupWindow = PopupWindow(popupView, 900,
ViewGroup.LayoutParams.WRAP_CONTENT, focusable)
                    popupWindow.showAtLocation(view, Gravity.CENTER, 0, 0)


popupWindow.contentView.findViewById<ImageButton>(R.id.btn_menu).visibilit
y = GONE

popupWindow.contentView.findViewById<ImageButton>(R.id.btn_back).visibilit
y = GONE


popupWindow.contentView.findViewById<Button>(R.id.pop_details).setOnClick
Listener {
                        val intent = Intent(this, DetailsDamageActivity::class.java)
                        startActivity(intent)
                        finish()
                    }


popupWindow.contentView.findViewById<Button>(R.id.pop_comment).setOnCli
ckListener {
                        popupWindow.dismiss()
                        val inflater2 = LayoutInflater.from(this)
                        val popupView2: View =
inflater2.inflate(R.layout.pop_window_comment, null)
                        val focusable2 = true
                        val popupWindow2 = PopupWindow(popupView2, 900, 800,
focusable2)
                        popupWindow2.showAtLocation(view, Gravity.CENTER, 0, 0)

popupWindow2.contentView.findViewById<TextView>(R.id.tool_title).text =
"Comments"

popupWindow2.contentView.findViewById<ImageButton>(R.id.btn_menu).visibil
ity = View.GONE

popupWindow2.contentView.findViewById<ImageButton>(R.id.btn_back).visibili
```

```
ty = View.GONE

popupWindow2.contentView.findViewById<Button>(R.id.pop_update).visibility =
View.GONE

popupWindow2.contentView.findViewById<Button>(R.id.pop_clear_all).visibility
= View.GONE
                      val date = start.toString()
                      val items1 = date.split("/".toRegex()).toTypedArray()
                      val day = items1[0]
                      val month = items1[1]
                      val year = items1[2]

                      val db = FirebaseFirestore.getInstance()
                      val df = db.collection("Aircraft revision")
                            .document(aircraft + "_" + check + "_" + day + month +
year)
                            .collection("New
damages").document(damage_id.toString())
                      df.get().addOnSuccessListener { documentSnapshot ->
                         val comment =
documentSnapshot.getString("Comments")
                         val editComment : TextView =
popupWindow2.contentView.findViewById(R.id.inputComment)
                         editComment.text = comment
                      }

popupWindow2.contentView.findViewById<Button>(R.id.pop_update).visibility =
GONE

popupWindow2.contentView.findViewById<Button>(R.id.pop_clear_all).visibility
= GONE
                   }


popupWindow.contentView.findViewById<Button>(R.id.pop_damage_document
s).setOnClickListener {

                      val SHARED_PREF_plane = "plane"
                      val KEY_plane = "key_plane"
                      val sp4 = getSharedPreferences(SHARED_PREF_plane,
MODE_PRIVATE)
                      val editor = sp4.edit()
                      editor.putString(KEY_plane, id)
                      editor.apply()

                      val SHARED_PREF_DAMAGE = "DAMAGE"
                      val KEY_DAMAGE = "key_DAMAGE"
                      val sp5 = getSharedPreferences(SHARED_PREF_DAMAGE,
MODE_PRIVATE)
```

```
                        val editor2 = sp5.edit()
                        editor2.putString(KEY_DAMAGE, damage_id)
                        editor2.apply()

                        val intent = Intent(this,
DocumentsDamageActivity::class.java)
                        startActivity(intent)
                    }

                }

                if (documentSnapshot.getString("Job") == "Airplane
maintenance technician") {

                    val inflater = LayoutInflater.from(this)
                    val popupView: View =
inflater.inflate(R.layout.pop_window_amt_damages, null)
                    val focusable = true
                    val popID : TextView = popupView.findViewById(R.id.tool_title)
                    popID.isAllCaps=false
                    popID.text = ("Damage  $damage_id")
                    val popupWindow = PopupWindow(popupView, 900,
ViewGroup.LayoutParams.WRAP_CONTENT, focusable)
                    popupWindow.showAtLocation(view, Gravity.CENTER, 0, 0)


popupWindow.contentView.findViewById<ImageButton>(R.id.btn_menu).visibilit
y = GONE

popupWindow.contentView.findViewById<ImageButton>(R.id.btn_back).visibilit
y = GONE


popupWindow.contentView.findViewById<Button>(R.id.pop_details).setOnClick
Listener {
                        val intent = Intent(this, DetailsDamageActivity::class.java)
                        startActivity(intent)
                        finish()
                    }


popupWindow.contentView.findViewById<Button>(R.id.pop_edit_damage).setO
nClickListener {
                        val intent = Intent(this, EditDamageActivity::class.java)
                        startActivity(intent)
                        finish()
                    }


popupWindow.contentView.findViewById<Button>(R.id.pop_sdt_rb).setOnClick
```

```
Listener {
                            if (category.toString() == "A") {
                                val builder = AlertDialog.Builder(this)
                                builder.setTitle("Error")
                                builder.setMessage("SDT RB only available for damage
category B and C.")
                                builder.setNegativeButton("Accept", null)
                                builder.create().show()
                            }
                            if (category.toString() == "B" || category.toString() == "C") {
                                val intent = Intent(this, sdtRbActivity::class.java)
                                startActivity(intent)
                                finish()
                            }

                        }


popupWindow.contentView.findViewById<Button>(R.id.pop_comment).setOnCli
ckListener {
                            popupWindow.dismiss()
                            val inflater2 = LayoutInflater.from(this)
                            val popupView2: View =
inflater2.inflate(R.layout.pop_window_comment, null)
                            val focusable2 = true
                            val popupWindow2 = PopupWindow(popupView2, 900, 900,
focusable2)
                            popupWindow2.showAtLocation(view, Gravity.CENTER, 0, 0)
                            val date = start.toString()
                            val items1 = date.split("/".toRegex()).toTypedArray()
                            val day = items1[0]
                            val month = items1[1]
                            val year = items1[2]

                            val db = FirebaseFirestore.getInstance()
                            val df = db.collection("Aircraft revision")
                                    .document(aircraft + "_" + check + "_" + day + month +
year)
                                    .collection("New
damages").document(damage_id.toString())
                            df.get().addOnSuccessListener { documentSnapshot ->
                                val comment =
documentSnapshot.getString("Comments")
                                val editComment : TextView =
popupWindow2.contentView.findViewById(R.id.inputComment)
                                editComment.text = comment
                            }

popupWindow2.contentView.findViewById<Button>(R.id.pop_update).visibility =
GONE

popupWindow2.contentView.findViewById<Button>(R.id.pop_clear_all).visibility
= GONE
                    }


popupWindow.contentView.findViewById<Button>(R.id.pop_damage_document
s).setOnClickListener {

                       val SHARED_PREF_plane = "plane"
                       val KEY_plane = "key_plane"
                       val sp4 = getSharedPreferences(SHARED_PREF_plane,
MODE_PRIVATE)
                       val editor = sp4.edit()
                       editor.putString(KEY_plane, id)
                       editor.apply()

                       val SHARED_PREF_DAMAGE = "DAMAGE"
                       val KEY_DAMAGE = "key_DAMAGE"
                       val sp5 = getSharedPreferences(SHARED_PREF_DAMAGE,
MODE_PRIVATE)
                       val editor2 = sp5.edit()
                       editor2.putString(KEY_DAMAGE, damage_id)
                       editor2.apply()

                       val intent = Intent(this,
DocumentsDamageActivity::class.java)
                       startActivity(intent)
                    }

                }

                if (documentSnapshot.getString("Job") == "Administrator") {
                    val inflater = LayoutInflater.from(this)
                    val popupView: View =
inflater.inflate(R.layout.pop_window_admin_damages, null)
                    val focusable = true
                    val popID : TextView = popupView.findViewById(R.id.tool_title)
                    popID.isAllCaps=false
                    popID.text = ("Damage  $damage_id")
                    val popupWindow = PopupWindow(popupView, 900,
ViewGroup.LayoutParams.WRAP_CONTENT, focusable)
                    popupWindow.showAtLocation(view, Gravity.CENTER, 0, 0)


popupWindow.contentView.findViewById<ImageButton>(R.id.btn_menu).visibilit
y = GONE

popupWindow.contentView.findViewById<ImageButton>(R.id.btn_back).visibilit
```

```
y = GONE


popupWindow.contentView.findViewById<Button>(R.id.pop_details).setOnClick
Listener {
                        val intent = Intent(this, DetailsDamageActivity::class.java)
                        startActivity(intent)
                        finish()
                    }


popupWindow.contentView.findViewById<Button>(R.id.pop_delete).setOnClick
Listener {

                        popupWindow.dismiss()

                        val inflater3 = LayoutInflater.from(this)
                        val popupView3: View =
inflater3.inflate(R.layout.pop_window_pin, null)
                        val focusable3 = true
                        val popupWindow3 = PopupWindow(popupView3, 900,
ViewGroup.LayoutParams.WRAP_CONTENT, focusable3)
                        popupWindow3.showAtLocation(view, Gravity.CENTER, 0, 0)


popupWindow3.contentView.findViewById<TextView>(R.id.tool_title).text =
"Write PIN"

popupWindow3.contentView.findViewById<ImageButton>(R.id.btn_menu).visibil
ity = GONE

popupWindow3.contentView.findViewById<ImageButton>(R.id.btn_back).visibili
ty = GONE

                        df.get().addOnSuccessListener { documentSnapshot ->
                            val txtpin =
popupWindow3.contentView.findViewById<TextView>(R.id.inputPIN)
                            val delete: Button =
popupWindow3.contentView.findViewById(R.id.pop_pin)
                            delete.setOnClickListener {
                                val builder = AlertDialog.Builder(this)
                                builder.setTitle("CAUTION")
                                builder.setMessage("Are you sure you want to
continue?")
                                builder.setPositiveButton("Accept") { dialog, which ->
                                    if (documentSnapshot.getString("PIN") ==
txtpin.text.toString()) {
                                        db.collection("Aircraft revision").document(aircraft
+ "_" + check + "_" + day + month + year)
                                            .collection("New
damages").document("$damage_id").delete()
                                        db.collection("Fleet").document(aircraft.toString())
                                               .collection("Damage
map").document("$damage_id").delete()
                                        popupWindow3.dismiss()
                                        startActivity(intent)
                                    } else {
                                        val builder1 = AlertDialog.Builder(this)
                                        builder1.setTitle("Error")
                                        builder1.setMessage("Incorrect PIN, please try
again.")
                                        builder1.setPositiveButton("Accept", null)
                                        val dialog1: AlertDialog = builder1.create()
                                        dialog1.show()
                                        popupWindow3.dismiss()
                                    }
                                }
                                builder.setNegativeButton("Cancel", null)
                                val dialog: AlertDialog = builder.create()
                                dialog.show()
                            }
                        }
                    }


popupWindow.contentView.findViewById<Button>(R.id.pop_edit_damage).setO
nClickListener {
                        val intent = Intent(this, EditDamageActivity::class.java)
                        startActivity(intent)
                        finish()
                    }


popupWindow.contentView.findViewById<Button>(R.id.pop_sdt_rb).setOnClick
Listener {

                        if (category.toString() == "A") {
                            val builder = AlertDialog.Builder(this)
                            builder.setTitle("Error")
                            builder.setMessage("SDT RB only available for damage
category B and C.")
                            builder.setNegativeButton("Accept", null)
                            builder.create().show()
                        }
                        if (category.toString() == "B" || category.toString() == "C") {
                            val intent = Intent(this, sdtRbActivity::class.java)
                            startActivity(intent)
                            finish()
                        }
```

```
            }


popupWindow.contentView.findViewById<Button>(R.id.pop_damage_document
s).setOnClickListener {

                val SHARED_PREF_plane = "plane"
                val KEY_plane = "key_plane"
                val sp4 = getSharedPreferences(SHARED_PREF_plane,
MODE_PRIVATE)
                val editor = sp4.edit()
                editor.putString(KEY_plane, id)
                editor.apply()

                val SHARED_PREF_DAMAGE = "DAMAGE"
                val KEY_DAMAGE = "key_DAMAGE"
                val sp5 = getSharedPreferences(SHARED_PREF_DAMAGE,
MODE_PRIVATE)
                val editor2 = sp5.edit()
                editor2.putString(KEY_DAMAGE, damage_id)
                editor2.apply()

                val intent = Intent(this,
DocumentsDamageActivity::class.java)
                startActivity(intent)
            }


popupWindow.contentView.findViewById<Button>(R.id.pop_comment).setOnCli
ckListener {
                popupWindow.dismiss()
                val inflater2 = LayoutInflater.from(this)
                val popupView2: View =
inflater2.inflate(R.layout.pop_window_comment, null)
                val focusable2 = true
                val popupWindow2 = PopupWindow(popupView2, 900, 900,
focusable2)
                popupWindow2.showAtLocation(view, Gravity.CENTER, 0, 0)
                val date = start.toString()
                val items1 = date.split("/".toRegex()).toTypedArray()
                val day = items1[0]
                val month = items1[1]
                val year = items1[2]

                val db = FirebaseFirestore.getInstance()
                val df = db.collection("Aircraft revision")
                    .document(aircraft + "_" + check + "_" + day + month +
year)
                    .collection("New
damages").document(damage_id.toString())
```

```
                        df.get().addOnSuccessListener { documentSnapshot ->
                            val comment =
documentSnapshot.getString("Comments")
                            val editComment : TextView =
popupWindow2.contentView.findViewById(R.id.inputComment)
                            editComment.text = comment
                        }

popupWindow2.contentView.findViewById<Button>(R.id.pop_update).visibility =
GONE

popupWindow2.contentView.findViewById<Button>(R.id.pop_clear_all).visibility
= GONE
                    }

                }
            }

            val SHARED_PREF_DAMAGE = "DAMAGE"
            val KEY_DAMAGE= "key_DAMAGE"
            val sp = getSharedPreferences(SHARED_PREF_DAMAGE,
MODE_PRIVATE)
            val editor = sp.edit()
            editor.putString(KEY_DAMAGE, damage_id)
            editor.apply()
        }
    }
}

override fun onItemLongClick(view: View?, postion: Int) {
    TODO("Not yet implemented")
}
```

A.8. Damage map activity

```
class DamageMapActivity : AppCompatActivity() , MyItemClickListener,
MyItemLongClickListener {

    private var mAdapter: DamageAdapter? = null

    override fun onCreate(savedInstanceState: Bundle?) {

        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_damages)

        val title: TextView = findViewById(R.id.tool_title)
        title.text = "DAMAGE MAP"

        val back : ImageButton = findViewById(R.id.btn_back)
        back.setOnClickListener{
            openActivity1()
        }

        initView()
        initData()

        showMenu()
    }

    private fun openActivity1() {

        val user = FirebaseAuth.getInstance().currentUser
        val userEmail = user.email
        val db = FirebaseFirestore.getInstance()
        val df = db.collection("Users").document(userEmail)
        df.get().addOnSuccessListener { documentSnapshot ->

            if (documentSnapshot.getString("Job") == "MRO engineer" ||
                    documentSnapshot.getString("Job") == "CAMO engineer"
                    || documentSnapshot.getString("Job") == "Total access
engineer") {
                val intent = Intent(this, EngineerHomeActivity::class.java)
                startActivity(intent)
                finish()
            }

            if (documentSnapshot.getString("Job") == "Administrator") {
                val intent = Intent(this, AdminHomeActivity::class.java)
                startActivity(intent)
                finish()
            }

            if (documentSnapshot.getString("Job") == "Airplane maintenance
technician") {
                val intent = Intent(this, AMTHomeActivity::class.java)
                startActivity(intent)
                finish()
            }

            if (documentSnapshot.getString("Job") == "Cabin crew") {
                val intent = Intent(this, CrewHomeActivity::class.java)
                startActivity(intent)
                finish()
            }
        }
    }

    private fun initView() {
        val rvDamages =
findViewById<RecyclerView>(R.id.recyclerViewDamagesList)
        rvDamages.layoutManager = LinearLayoutManager(this)
        rvDamages.itemAnimator = DefaultItemAnimator()
    }

    private fun initData() {
        val SHARED_PREF_AIRCRAFT = "aircraft"
        val KEY_AIRCRAFT = "key_aircraft"
        val sp = getSharedPreferences(SHARED_PREF_AIRCRAFT,
MODE_PRIVATE)
        val aircraft = sp.getString(KEY_AIRCRAFT, null)

        val db : FirebaseFirestore= FirebaseFirestore.getInstance()
        val collection: CollectionReference =
db.collection("Fleet").document("$aircraft")
                .collection("Damage map")
        val query : Query = collection
        val rvDamage=
findViewById<RecyclerView>(R.id.recyclerViewDamagesList)
        query.get().addOnCompleteListener { task ->
            if (task.isSuccessful) {
                val mData: java.util.ArrayList<DamageModel> = java.util.ArrayList()
                for (document in task.result) {
                    val taskItem: DamageModel =
document.toObject(DamageModel::class.java)
                    mData.add(taskItem)
                }
                mAdapter = DamageAdapter(mData)
                rvDamage.adapter = mAdapter
                mAdapter!!.setOnItemClickListener(this)
                mAdapter!!.setOnItemLongClickListener(this)
                mAdapter!!.notifyDataSetChanged()
            }
        }

    }


    private fun showMenu() {
        val button : ImageButton = findViewById(R.id.btn_menu)
        button.setOnClickListener {
            val user = FirebaseAuth.getInstance().currentUser
            val userEmail = user.email
            val db = FirebaseFirestore.getInstance()
            val df = db.collection("Users").document(userEmail)
            df.get().addOnSuccessListener { documentSnapshot ->

            }
        }
    }
```

```
 override fun onItemClick(view: View?, position: Int) {
     val SHARED_PREF_AIRCRAFT = "aircraft"
     val KEY_AIRCRAFT = "key_aircraft"
     val sp = getSharedPreferences(SHARED_PREF_AIRCRAFT,
MODE_PRIVATE)
     val aircraft = sp.getString(KEY_AIRCRAFT, null)

     val SHARED_PREF_START = "start"
     val KEY_START = "key_start"
     val sp1 = getSharedPreferences(SHARED_PREF_START,
MODE_PRIVATE)
     val start = sp1.getString(KEY_START, null)

     val SHARED_PREF_check = "check"
     val KEY_check = "key_check"
     val sp2 = getSharedPreferences(SHARED_PREF_check,
MODE_PRIVATE)
     val check = sp2.getString(KEY_check, null)

     val items1 = start?.split("/".toRegex())?.toTypedArray()
     val day = items1?.get(0)
     val month = items1?.get(1)
     val year = items1?.get(2)
     val id = aircraft + "_" + check + "_" + day + month + year
     val db : FirebaseFirestore= FirebaseFirestore.getInstance()
     val collection: CollectionReference =
db.collection("Fleet").document("$aircraft")
         .collection("Damage map")
     val query : Query = collection
     query.get().addOnCompleteListener { task ->
       if (task.isSuccessful) {
         val mData: ArrayList<DamageModel> = ArrayList()
         for (document in task.result) {
           val taskItem: DamageModel =
document.toObject(DamageModel::class.java)
           mData.add(taskItem)
         }
         val bean: DamageModel = mData[position - 1]
         val damage_id = bean.ID
         val category = bean.Category

         val user = FirebaseAuth.getInstance().currentUser
         val userEmail = user.email
         val db = FirebaseFirestore.getInstance()
         val df = db.collection("Users").document(userEmail)
         df.get().addOnSuccessListener { documentSnapshot ->

         }
```

```
                val SHARED_PREF_DAMAGE = "DAMAGE"
                val KEY_DAMAGE= "key_DAMAGE"
                val sp = getSharedPreferences(SHARED_PREF_DAMAGE,
MODE_PRIVATE)
                val editor = sp.edit()
                editor.putString(KEY_DAMAGE, damage_id)
                editor.apply()
            }
        }
    }

    override fun onItemLongClick(view: View?, postion: Int) {
        TODO("Not yet implemented")
    }
}
```

A.9. Details damage activity

```
class DetailsDamageActivity : AppCompatActivity() {

    private var mAdapter: AircraftAdapter? = null

    override fun onCreate(savedInstanceState: Bundle?) {

        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_damage_details)

        val title: TextView = findViewById(R.id.tool_title)
        title.text = "Details"

        val button : ImageButton = findViewById(R.id.btn_menu)
        button.visibility = View.GONE

        val dimen = resources.getStringArray(R.array.dimen_array)
        val spinner: Spinner = findViewById(R.id.spinner_dimen)
        val adapter = ArrayAdapter(this, R.layout.spinner_item_selected2, dimen)
        adapter.setDropDownViewResource(R.layout.spinner_dropdown_item)
        spinner.adapter = adapter
        spinner.avoidDropdownFocus()
        spinner.onItemSelectedListener = object :
AdapterView.OnItemSelectedListener {
            override fun onItemSelected(
                parent: AdapterView<*>,
                view: View?,
                position: Int,
                id: Long
            ) {
                if (parent.getItemAtPosition(position) == "-") {
            } else {
                val SHARED_PREF_DIMEN = "DIMEN";
                val KEY_DIMEN = "key_DIMEN";
                val mSpnValue: String = spinner.selectedItem.toString()
                val myPrefs: SharedPreferences =
getSharedPreferences(SHARED_PREF_DIMEN, MODE_PRIVATE)
                val prefsEditor = myPrefs.edit()
                prefsEditor.putString(KEY_DIMEN, mSpnValue)
                prefsEditor.apply()
            }
        }

        override fun onNothingSelected(parent: AdapterView<*>?) {}
    }

    val type = resources.getStringArray(R.array.type_array)
    val spinner_type: Spinner = findViewById(R.id.spinner_Type)
    val adapter_type = ArrayAdapter(this, R.layout.spinner_item_selected,
type)

adapter_type.setDropDownViewResource(R.layout.spinner_dropdown_item)
    spinner_type.adapter = adapter_type
    spinner_type.avoidDropdownFocus()
    spinner_type.onItemSelectedListener = object :
AdapterView.OnItemSelectedListener {
        override fun onItemSelected(
                parent: AdapterView<*>,
                view: View?,
                position: Int,
                id: Long
        ) {
            if (parent.getItemAtPosition(position) == "-") {
            }
        }

        override fun onNothingSelected(parent: AdapterView<*>?) {}
    }


    val cat = resources.getStringArray(R.array.cat_array)
    val spinner1: Spinner = findViewById(R.id.spinner_cat)
    val adapter1 = ArrayAdapter(this, R.layout.spinner_item_selected, cat)
    adapter1.setDropDownViewResource(R.layout.spinner_dropdown_item)
    spinner1.adapter = adapter1
    spinner1.avoidDropdownFocus()
    spinner1.onItemSelectedListener = object :
AdapterView.OnItemSelectedListener {
        override fun onItemSelected(
                parent: AdapterView<*>,
                view: View?,
                position: Int,
                id: Long
        ) {

        }

        override fun onNothingSelected(parent: AdapterView<*>?) {}
    }

    val btn_back : ImageButton = findViewById(R.id.btn_back)
    btn_back.setOnClickListener {
        val intent = Intent(this, DamageActivity::class.java)
        startActivity(intent)
    }

    displayInfo()

    val calendar = Calendar.getInstance()
    val edit = findViewById<View>(R.id.input_entry) as EditText
    val yy = calendar.get(Calendar.YEAR)
    val mm = calendar.get(Calendar.MONTH)
    val dd = calendar.get(Calendar.DAY_OF_MONTH)
    val datePicker = DatePickerDialog(
            this@DetailsDamageActivity,
            { _, year, monthOfYear, dayOfMonth ->
                val date =
                        (dayOfMonth.toString() + "/" + (monthOfYear + 1).toString() +
"/" + year.toString())
                edit.setText(date)
            },
            yy,
            mm,
            dd
    )
    datePicker.setButton(DialogInterface.BUTTON_NEUTRAL, "CLEAR") { _,
which ->
        if (which == DialogInterface.BUTTON_NEUTRAL) {
            edit.setText("-")
        }
    }

    edit.setOnClickListener {
        datePicker.hide()
    }

    val edit2 = findViewById<View>(R.id.input_closing) as EditText
    val calendar2 = Calendar.getInstance()
    val yy2 = calendar2.get(Calendar.YEAR)
    val mm2 = calendar2.get(Calendar.MONTH)
    val dd2 = calendar2.get(Calendar.DAY_OF_MONTH)
        val datePicker2 = DatePickerDialog(
            this@DetailsDamageActivity,
            { _, year, monthOfYear, dayOfMonth ->
                val date2 =
                    (dayOfMonth.toString() + "/" + (monthOfYear + 1).toString() +
"/" + year.toString())
                edit2.setText(date2)
            },
            yy2,
            mm2,
            dd2
        )
        datePicker2.setButton(DialogInterface.BUTTON_NEUTRAL, "CLEAR") {
_, which ->
            if (which == DialogInterface.BUTTON_NEUTRAL) {
                edit2.setText("-")
            }
        }

        edit2.setOnClickListener() {
            datePicker2.hide()
        }

        changeButton()

        buttonEdit()

    }

    override fun onBackPressed() {
        super.onBackPressed()
        val intent = Intent(this, DamageActivity::class.java)
        startActivity(intent)
    }

    private fun changeButton() {
        val btn_add : Button = findViewById(R.id.btn_Add)
        val btn_map : Button = findViewById(R.id.pop_damage_nomap)
        val btn_edit : Button = findViewById(R.id.pop_damage_map)
        val user = FirebaseAuth.getInstance().currentUser
        val userEmail = user.email
        val db = FirebaseFirestore.getInstance()
        val df3 = db.collection("Users").document(userEmail)
        df3.get().addOnSuccessListener { documentSnapshot ->
            if (documentSnapshot.getString("Job") == "MRO engineer") {
                btn_add.setText("Certify")
            }
            if (documentSnapshot.getString("Job") == "CAMO engineer") {
                btn_add.setText("Check")
            }
            if (documentSnapshot.getString("Job") == "Airplane maintenance
technician") {
                btn_edit.setText("Edit")
                btn_add.visibility = View.GONE
                btn_map.visibility = View.GONE
            }
            if (documentSnapshot.getString("Job") == "Total access engineer"
                || documentSnapshot.getString("Job") == "Administrator") {
                btn_add.setText("Certify/check")
            }
        }
    }

    private fun edit() {
        val edit : Button = findViewById(R.id.pop_damage_map)
        edit.setOnClickListener {
            val intent = Intent(this, EditDamageActivity::class.java)
            startActivity(intent)
            finish()
        }
    }

    private fun buttonEdit() {
        val user = FirebaseAuth.getInstance().currentUser
        val userEmail = user.email
        val db = FirebaseFirestore.getInstance()
        val df = db.collection("Users").document(userEmail)
        df.get().addOnSuccessListener { documentSnapshot ->
            if (documentSnapshot.getString("Job") == "Airplane maintenance
technician") {
                edit()
            } else {
                add()
                map()
            }
        }
    }

    private fun add() {

        val SHARED_PREF_DAMAGE = "DAMAGE"
        val KEY_DAMAGE = "key_DAMAGE"
        val sp0 = getSharedPreferences(SHARED_PREF_DAMAGE,
MODE_PRIVATE)
        val damage = sp0.getString(KEY_DAMAGE, null)

        val SHARED_PREF_AIRCRAFT = "aircraft";
        val KEY_AIRCRAFT = "key_aircraft";
        val sp = getSharedPreferences(SHARED_PREF_AIRCRAFT,
MODE_PRIVATE)
```

```
      val aircraft = sp.getString(KEY_AIRCRAFT, null)

      val SHARED_PREF_START = "start"
      val KEY_START = "key_start"
      val sp1 = getSharedPreferences(SHARED_PREF_START,
MODE_PRIVATE)
      val start = sp1.getString(KEY_START, null)

      val SHARED_PREF_check = "check"
      val KEY_check = "key_check"
      val sp2 = getSharedPreferences(SHARED_PREF_check,
MODE_PRIVATE)
      val check = sp2.getString(KEY_check, null)

      val items1 = start?.split("/".toRegex())?.toTypedArray()
      val day = items1?.get(0)
      val month = items1?.get(1)
      val year = items1?.get(2)

      val id = aircraft + "_" + check + "_" + day + month + year
      val db2 = FirebaseFirestore.getInstance()
      val df2: DocumentReference = db2.collection("Aircraft
revision").document(id)
            .collection("New damages").document(damage.toString())
      val btn_add : Button = findViewById(R.id.btn_Add)
      btn_add.setOnClickListener {
         df2.get().addOnSuccessListener { documentSnapshot ->
            if (documentSnapshot.getString("Certified") == "Yes"
                  && documentSnapshot.getString("Checked") == "Yes") {
               val builder = AlertDialog.Builder(this)
               builder.setTitle("Information")
               builder.setMessage("This action has been done previously.")
               builder.setPositiveButton("Accept", null)
               val dialog: AlertDialog = builder.create()
               dialog.show()
            } else {
               val db = FirebaseFirestore.getInstance()
               val user = FirebaseAuth.getInstance().currentUser
               val userEmail = user.email
               val df= db.collection("Users").document(userEmail.toString())
               df.get().addOnSuccessListener { documentSnapshot ->
                  if (documentSnapshot.getString("Job") == "MRO engineer") {
                        val SHARED_PREF_DAMAGE = "DAMAGE"
                        val KEY_DAMAGE = "key_DAMAGE"
                        val sp =
getSharedPreferences(SHARED_PREF_DAMAGE, MODE_PRIVATE)
                        val damage = sp.getString(KEY_DAMAGE, null)


                     val inflater3 = LayoutInflater.from(this)
```

```
                    val popupView3: View =
inflater3.inflate(R.layout.pop_window_pin, null)
                    val focusable3 = true
                    val popupWindow3 = PopupWindow(popupView3, 900, 400,
focusable3)
                    popupWindow3.showAtLocation(popupView3,
Gravity.CENTER, 0, 0)

popupWindow3.contentView.findViewById<TextView>(R.id.tool_title).text =
"WRITE PIN"

popupWindow3.contentView.findViewById<ImageButton>(R.id.btn_menu).visibil
ity = View.GONE

popupWindow3.contentView.findViewById<ImageButton>(R.id.btn_back).visibili
ty = View.GONE

                       df.get().addOnSuccessListener { documentSnapshot ->
                          val txtpin =
popupWindow3.contentView.findViewById<TextView>(R.id.inputPIN)
                          val pin: Button =
popupWindow3.contentView.findViewById(R.id.pop_pin)
                          pin.setText("Certify")
                          pin.setOnClickListener {
                             if (documentSnapshot.getString("PIN") ==
txtpin.text.toString()) {
                                val builder1 = AlertDialog.Builder(this)
                                builder1.setTitle("Information")
                                builder1.setMessage("Damage report
successfully certified.")
                                builder1.setPositiveButton("Accept", null)
                                val dialog1: AlertDialog = builder1.create()
                                dialog1.show()
                                db.collection("Fleet")
                                      .document(aircraft.toString())
                                      .collection("Damage map")

.document(damage.toString()).update("Certified", "Yes",
                                            "CertifyBY", userEmail.toString())
                                db.collection("Aircraft revision")
                                      .document(aircraft + "_" + check + "_" + day +
month + year)
                                      .collection("New
damages").document(damage.toString()).update("Certified",
"Yes","CertifyBY", userEmail.toString())
                                popupWindow3.dismiss()
                                val intent = Intent(this, DamageActivity::class.java)
                                startActivity(intent)
                                finish()
                             } else {
```

```
                                        val builder1 = AlertDialog.Builder(this)
                                        builder1.setTitle("Error")
                                        builder1.setMessage("Incorrect PIN, please try
again.")
                                        builder1.setPositiveButton("Accept", null)
                                        val dialog1: AlertDialog = builder1.create()
                                        dialog1.show()
                                        popupWindow3.dismiss()

                                    }
                                }
                            }
                      }
                      if (documentSnapshot.getString("Job") == "CAMO engineer") {


                          val inflater3 = LayoutInflater.from(this)
                          val popupView3: View =
inflater3.inflate(R.layout.pop_window_pin, null)
                          val focusable3 = true
                          val popupWindow3 = PopupWindow(popupView3, 900, 400,
focusable3)
                          popupWindow3.showAtLocation(popupView3,
Gravity.CENTER, 0, 0)

popupWindow3.contentView.findViewById<TextView>(R.id.tool_title).text =
"WRITE PIN"

popupWindow3.contentView.findViewById<ImageButton>(R.id.btn_menu).visibil
ity = View.GONE

popupWindow3.contentView.findViewById<ImageButton>(R.id.btn_back).visibili
ty = View.GONE
                              df.get().addOnSuccessListener { documentSnapshot ->
                                  val txtpin =
popupWindow3.contentView.findViewById<TextView>(R.id.inputPIN)
                                  val pin: Button =
popupWindow3.contentView.findViewById(R.id.pop_pin)
                                  pin.setText("Check")
                                  pin.setOnClickListener {
                                      if (documentSnapshot.getString("PIN") ==
txtpin.text.toString()) {
                                          val builder1 = AlertDialog.Builder(this)
                                          builder1.setTitle("Information")
                                          builder1.setMessage("Damage report
successfully checked.")
                                          builder1.setPositiveButton("Accept", null)
                                          val dialog1: AlertDialog = builder1.create()
                                          dialog1.show()
                                          db.collection("Fleet")
                                        .document(aircraft.toString())
                                        .collection("Damage map")

.document(damage.toString()).update("Checked", "Yes","CheckedBY",
userEmail.toString())
                                    db.collection("Aircraft revision")
                                        .document(aircraft + "_" + check + "_" + day +
month + year)
                                        .collection("New
damages").document(damage.toString()).update("Checked", "Yes",
"CheckedBY", userEmail.toString())
                                    popupWindow3.dismiss()
                                    val intent = Intent(this, DamageActivity::class.java)
                                    startActivity(intent)
                                    finish()

                                } else {
                                    val builder1 = AlertDialog.Builder(this)
                                    builder1.setTitle("Error")
                                    builder1.setMessage("Incorrect PIN, please try
again.")
                                    builder1.setPositiveButton("Accept", null)
                                    val dialog1: AlertDialog = builder1.create()
                                    dialog1.show()
                                    popupWindow3.dismiss()
                                }
                            }
                        }
                    }

                    if (documentSnapshot.getString("Job") == "Total access
engineer" ||
                            documentSnapshot.getString("Job") == "Administrator") {
                        val SHARED_PREF_DAMAGE = "DAMAGE"
                        val KEY_DAMAGE = "key_DAMAGE"
                        val sp = getSharedPreferences(SHARED_PREF_DAMAGE,
MODE_PRIVATE)
                        val damage = sp.getString(KEY_DAMAGE, null)

                        df.get().addOnSuccessListener { documentSnapshot ->

                            val builder1 = AlertDialog.Builder(this)
                            builder1.setTitle("Select an option")
                            builder1.setMessage("Do you want to check or certify
the damage? Please choose an option below.")
                            builder1.setPositiveButton("Certify") { dialog, which ->

                                val inflater3 = LayoutInflater.from(this)
                                val popupView3: View =
inflater3.inflate(R.layout.pop_window_pin, null)
```

```
                            val focusable3 = true
                            val popupWindow3 = PopupWindow(popupView3, 900,
400, focusable3)
                            popupWindow3.showAtLocation(popupView3,
Gravity.CENTER, 0, 0)

popupWindow3.contentView.findViewById<TextView>(R.id.tool_title).text =
"WRITE PIN"

popupWindow3.contentView.findViewById<ImageButton>(R.id.btn_menu).visibil
ity = View.GONE

popupWindow3.contentView.findViewById<ImageButton>(R.id.btn_back).visibili
ty = View.GONE
                            val txtpin =
popupWindow3.contentView.findViewById<TextView>(R.id.inputPIN)
                            val pin: Button =
popupWindow3.contentView.findViewById(R.id.pop_pin)
                            pin.setText("Confirm")
                            pin.setOnClickListener {
                                if (documentSnapshot.getString("PIN") ==
txtpin.text.toString()) {
                                    val builder1 = AlertDialog.Builder(this)
                                    builder1.setTitle("Information")
                                    builder1.setMessage("Damage report
successfully certified.")
                                    builder1.setPositiveButton("Accept", null)
                                    val dialog1: AlertDialog = builder1.create()
                                    dialog1.show()
                                    db.collection("Fleet")
                                        .document(aircraft.toString())
                                        .collection("Damage map")

.document(damage.toString()).update("Certified", "Yes",
                                            "CertifyBY", userEmail.toString())
                                    db.collection("Aircraft revision")
                                        .document(aircraft + "_" + check + "_" + day +
month + year)
                                        .collection("New
damages").document(damage.toString()).update("Certified", "Yes",
"CertifyBY", userEmail.toString())
                                    popupWindow3.dismiss()
                                } else {
                                    val builder1 = AlertDialog.Builder(this)
                                    builder1.setTitle("Error")
                                    builder1.setMessage("Incorrect PIN, please try
again.")
                                    builder1.setPositiveButton("Accept", null)
                                    val dialog1: AlertDialog = builder1.create()
                                    dialog1.show()
                                    popupWindow3.dismiss()
                                }
                            }
                        }
                        builder1.setNegativeButton("Check") { dialog, which ->
                            val inflater3 = LayoutInflater.from(this)
                            val popupView3: View =
inflater3.inflate(R.layout.pop_window_pin, null)
                            val focusable3 = true
                            val popupWindow3 = PopupWindow(popupView3, 900,
400, focusable3)
                            popupWindow3.showAtLocation(popupView3,
Gravity.CENTER, 0, 0)
                            val txtpin =
popupWindow3.contentView.findViewById<TextView>(R.id.inputPIN)
                            val pin: Button =
popupWindow3.contentView.findViewById(R.id.pop_pin)
                            pin.setText("Confirm")
                            pin.setOnClickListener {
                                if (documentSnapshot.getString("PIN") ==
txtpin.text.toString()) {
                                    val builder1 = AlertDialog.Builder(this)
                                    builder1.setTitle("Information")
                                    builder1.setMessage("Damage report successfully
checked.")
                                    builder1.setPositiveButton("Accept", null)
                                    val dialog1: AlertDialog = builder1.create()
                                    dialog1.show()
                                    db.collection("Fleet")
                                            .document(aircraft.toString())
                                            .collection("Damage map")

.document(damage.toString()).update("Checked", "Yes", "CheckedBY",
userEmail.toString())
                                    db.collection("Aircraft revision")
                                            .document(aircraft + "_" + check + "_" + day +
month + year)
                                            .collection("New
damages").document(damage.toString()).update("Checked", "Yes",
"CheckedBY", userEmail.toString())
                                    popupWindow3.dismiss()
                                } else {
                                    val builder1 = AlertDialog.Builder(this)
                                    builder1.setTitle("Error")
                                    builder1.setMessage("Incorrect PIN, please try
again.")
                                    builder1.setPositiveButton("Accept", null)
                                    val dialog1: AlertDialog = builder1.create()
                                    dialog1.show()
                                    popupWindow3.dismiss()
                        }
                      }

                    }
                    val dialog1: AlertDialog = builder1.create()
                    dialog1.show()
                  }
                }
              }
            }
          }
        }
      }

      private fun map() {
        val db = FirebaseFirestore.getInstance()
        val user = FirebaseAuth.getInstance().currentUser
        val userEmail = user.email
        val df= db.collection("Users").document(userEmail.toString())
        df.get().addOnSuccessListener { documentSnapshot ->

          val db : FirebaseFirestore= FirebaseFirestore.getInstance()
          val spinner = findViewById<View>(R.id.spinner_cat) as Spinner
          val cat = spinner.selectedItem.toString()
          val spinner2: Spinner = findViewById(R.id.spinner_dimen)
          val dimen = spinner2.selectedItem.toString()
          val spinner3 = findViewById<View>(R.id.spinner_Type) as Spinner
          val type = spinner3.selectedItem.toString()
          val ata = findViewById<EditText>(R.id.inputATA)
          val location = findViewById<TextView>(R.id.inputLocation)
          val dimen1: TextView = findViewById(R.id.input_dimen_1)
          val dimen2: TextView = findViewById(R.id.input_dimen_2)
          val dimen3: TextView = findViewById(R.id.input_dimen_3)
          val fh: EditText = findViewById(R.id.input_FH)
          val fc: EditText = findViewById(R.id.input_FC)
          val days: EditText = findViewById(R.id.input_Days)
          val fh2: EditText = findViewById(R.id.input_FH_repair)
          val fc2: EditText = findViewById(R.id.input_FC_repair)
          val days2: EditText = findViewById(R.id.input_Days_repair)
          val entry: EditText = findViewById(R.id.input_entry)
          val closing: EditText = findViewById(R.id.input_closing)
          val pn: EditText = findViewById(R.id.input_PN)
          val sn: EditText = findViewById(R.id.input_SN)
          val ref: EditText = findViewById(R.id.input_ref)
          val comment: EditText = findViewById(R.id.inputComment)
          val regis : EditText = findViewById(R.id.inputAircraft)

          val include:Button = findViewById(R.id.pop_damage_map)
          val no_map = findViewById<Button>(R.id.pop_damage_nomap)
```

```
        no_map.setOnClickListener {
            val inflater3 = LayoutInflater.from(this)
            val popupView3: View = inflater3.inflate(R.layout.pop_window_pin,
null)
            val focusable3 = true
            val popupWindow3 = PopupWindow(popupView3, 900, 400,
focusable3)
            popupWindow3.showAtLocation(popupView3, Gravity.CENTER, 0, 0)

            val txtpin =
popupWindow3.contentView.findViewById<TextView>(R.id.inputPIN)
            val pin: Button =
popupWindow3.contentView.findViewById(R.id.pop_pin)

            val SHARED_PREF_AIRCRAFT = "aircraft"
            val KEY_AIRCRAFT = "key_aircraft"
            val sp = getSharedPreferences(SHARED_PREF_AIRCRAFT,
MODE_PRIVATE)
            val aircraft = sp.getString(KEY_AIRCRAFT, null)

            val SHARED_PREF_DAMAGE = "DAMAGE"
            val KEY_DAMAGE= "key_DAMAGE"
            val sp3 = getSharedPreferences(SHARED_PREF_DAMAGE,
MODE_PRIVATE)
            val damage_id = sp3.getString(KEY_DAMAGE,null)

            pin.setOnClickListener {
                if (documentSnapshot.getString("PIN") == txtpin.text.toString()) {
                    val builder = AlertDialog.Builder(this)
                    builder.setTitle("Information")
                    builder.setMessage("Removed from damage map")
                    builder.setPositiveButton("Accept",null)
                    val dialog: AlertDialog = builder.create()
                    dialog.show()
                    db.collection("Fleet")
                            .document(aircraft.toString())
                            .collection("Damage
map").document(damage_id.toString()).delete()
                    popupWindow3.dismiss()

                }
                else {
                    val builder = AlertDialog.Builder(this)
                    builder.setTitle("Error")
                    builder.setMessage("Incorrect pin. Try again.")
                    builder.setPositiveButton("Accept",null)
                    val dialog: AlertDialog = builder.create()
                    dialog.show()
                    popupWindow3.dismiss()
                }
            }

        }

        include.setOnClickListener {
            val inflater3 = LayoutInflater.from(this)
            val popupView3: View = inflater3.inflate(R.layout.pop_window_pin,
null)
            val focusable3 = true
            val popupWindow3 = PopupWindow(popupView3, 900, 400,
focusable3)
            popupWindow3.showAtLocation(popupView3, Gravity.CENTER, 0, 0)

            val txtpin =
popupWindow3.contentView.findViewById<TextView>(R.id.inputPIN)
            val pin: Button =
popupWindow3.contentView.findViewById(R.id.pop_pin)

            val SHARED_PREF_AIRCRAFT = "aircraft"
            val KEY_AIRCRAFT = "key_aircraft"
            val sp = getSharedPreferences(SHARED_PREF_AIRCRAFT,
MODE_PRIVATE)
            val aircraft = sp.getString(KEY_AIRCRAFT, null)

            val SHARED_PREF_START = "start"
            val KEY_START = "key_start"
            val sp1 = getSharedPreferences(SHARED_PREF_START,
MODE_PRIVATE)
            val start = sp1.getString(KEY_START, null)

            val SHARED_PREF_check = "check"
            val KEY_check = "key_check"
            val sp2 = getSharedPreferences(SHARED_PREF_check,
MODE_PRIVATE)
            val check = sp2.getString(KEY_check, null)

            val items1 = start?.split("/".toRegex())?.toTypedArray()
            val day = items1?.get(0)
            val month = items1?.get(1)
            val year = items1?.get(2)

            val SHARED_PREF_DAMAGE = "DAMAGE"
            val KEY_DAMAGE= "key_DAMAGE"
            val sp3 = getSharedPreferences(SHARED_PREF_DAMAGE,
MODE_PRIVATE)
            val damage_id = sp3.getString(KEY_DAMAGE,null)

            pin.setOnClickListener {
                if (documentSnapshot.getString("PIN") == txtpin.text.toString()) {
```

```
            val builder = AlertDialog.Builder(this)
            builder.setTitle("Information")
            builder.setMessage("Added to damage map")
            builder.setPositiveButton("Accept",null)
            val dialog: AlertDialog = builder.create()
            dialog.show()
            val cc : TextView = findViewById(R.id.certify_by)
            val cc2 : TextView = findViewById(R.id.check_by)
            val revisionInfo = hashMapOf(
                "Aircraft" to regis.text.toString(),
                "Inspection FH" to fh.text.toString(),
                "Inspection FC" to fc.text.toString(),
                "Inspection days" to days.text.toString(),
                "Repair FH" to fh2.text.toString(),
                "Repair FC" to fc2.text.toString(),
                "Repair days" to days2.text.toString(),
                "Type" to type,
                "Dim" to dimen,
                "Dim1" to dimen1.text.toString(),
                "Dim2" to dimen2.text.toString(),
                "Dim3" to dimen3.text.toString(),
                "References" to ref.text.toString(),
                "Entry" to entry.text.toString(),
                "Closing" to closing.text.toString(),
                "Description" to type+"_"+location.text.toString(),
                "Comments" to comment.text.toString(),
                "PN" to pn.text.toString(),
                "SN" to sn.text.toString(),
                "Category" to cat,
                "ATA" to ata.text.toString(),
                "Location" to location.text.toString(),
                "Dimension" to dimen1.text.toString() + "x"
                    + dimen2.text.toString() + "x"
                    + dimen3.text.toString() + dimen,
                "ID" to damage_id,
                "CheckedBY" to "-",
                "CertifyBY" to "-")
            db.collection("Aircraft revision")
                .document(aircraft + "_" + check + "_" + day + month +
year)
                .collection("New
damages").document(damage_id.toString()).update("Map" , "Yes",
                    "CertifyBY", cc.text.toString(), "CheckBY" ,
cc2.text.toString())
            db.collection("Fleet")
                .document(aircraft.toString())
                .collection("Damage
map").document(damage_id.toString()).set(revisionInfo)
            db.collection("Fleet")
                .document(aircraft.toString())
```

```
                                .collection("Damage
map").document(damage_id.toString()).update("CertifyBY", cc.text.toString() ,
                                        "CheckBY" , cc2.text.toString())
                        popupWindow3.dismiss()
                    }
                    else {
                        val builder = AlertDialog.Builder(this)
                        builder.setTitle("Error")
                        builder.setMessage("Incorrect pin. Try again.")
                        builder.setPositiveButton("Accept",null)
                        val dialog: AlertDialog = builder.create()
                        dialog.show()
                        popupWindow3.dismiss()
                    }
                }
            }
        }
    }

    private fun Spinner.avoidDropdownFocus() {
        try {
            val isAppCompat = this is
androidx.appcompat.widget.AppCompatSpinner
            val spinnerClass = if (isAppCompat)
androidx.appcompat.widget.AppCompatSpinner::class.java else
Spinner::class.java
            val popupWindowClass = if (isAppCompat)
androidx.appcompat.widget.ListPopupWindow::class.java else
android.widget.ListPopupWindow::class.java

            val listPopup = spinnerClass
                    .getDeclaredField("mPopup")
                    .apply { isAccessible = true }
                    .get(this)
            if (popupWindowClass.isInstance(listPopup)) {
                val popup = popupWindowClass
                        .getDeclaredField("mPopup")
                        .apply { isAccessible = true }
                        .get(listPopup)
                if (popup is PopupWindow) {
                    popup.isFocusable = false
                }
            }
        } catch (e: Exception) {
            e.printStackTrace()
        }
    }

    private fun displayInfo()  {
```

```
    val SHARED_PREF_DAMAGE = "DAMAGE"
    val KEY_DAMAGE= "key_DAMAGE"
    val sp0= getSharedPreferences(SHARED_PREF_DAMAGE,
MODE_PRIVATE)
    val damage = sp0.getString(KEY_DAMAGE, null)

    val SHARED_PREF_AIRCRAFT = "aircraft";
    val KEY_AIRCRAFT = "key_aircraft";
    val sp = getSharedPreferences(SHARED_PREF_AIRCRAFT,
MODE_PRIVATE)
    val aircraft = sp.getString(KEY_AIRCRAFT, null)

    val SHARED_PREF_START = "start"
    val KEY_START = "key_start"
    val sp1 = getSharedPreferences(SHARED_PREF_START,
MODE_PRIVATE)
    val start = sp1.getString(KEY_START, null)

    val SHARED_PREF_check = "check"
    val KEY_check = "key_check"
    val sp2 = getSharedPreferences(SHARED_PREF_check,
MODE_PRIVATE)
    val check = sp2.getString(KEY_check, null)

    val items1 = start?.split("/".toRegex())?.toTypedArray()
    val day = items1?.get(0)
    val month = items1?.get(1)
    val year = items1?.get(2)

    val id = aircraft+"_"+check+"_"+day+month+year
    val db = FirebaseFirestore.getInstance()
    val df: DocumentReference = db.collection("Aircraft
revision").document(id)
        .collection("New damages").document(damage.toString())

    val ref =
FirebaseStorage.getInstance().reference.child("damage/$damage")
    val imageView : ImageView = this.findViewById(R.id.imageView2)
    imageView.drawable
    ref.downloadUrl.addOnSuccessListener {Uri->
      val imageURL = Uri.toString()
      Glide.with(this)
          .load(imageURL)
          .into(imageView)
    }

    df.get().addOnSuccessListener { documentSnapshot ->

      val cc : TextView = findViewById(R.id.certify_by)
      val certify = documentSnapshot.getString("CertifyBY").toString()
```

```
            cc.text = (certify)

            val cc1 : TextView = findViewById(R.id.check_by)
            val check_by = documentSnapshot.getString("CheckedBY").toString()
            cc1.text = (check_by)

            val txtplane = findViewById<TextView>(R.id.inputAircraft)
            val plane = documentSnapshot.getString("Aircraft").toString()
            txtplane.text = plane
            txtplane.isFocusable=false
            txtplane.setBackgroundResource(R.drawable.custom_input)

            val spinner = findViewById<View>(R.id.spinner_cat) as Spinner
            if (documentSnapshot.getString("Category") == "-") {
                for (i in 0 until spinner.adapter.count) {
                    if (spinner.adapter.getItem(i).toString().contains(aircraft.toString()))
{
                        spinner.setSelection(0)
                    }
                }
            } else {
                val value = documentSnapshot.getString("Category")
                setSpinText(spinner, value)
            }
            spinner.isEnabled=false

            val spinner3 = findViewById<View>(R.id.spinner_Type) as Spinner
            if (documentSnapshot.getString("Type") == "-") {
                for (i in 0 until spinner3.adapter.count) {
                    if
(spinner3.adapter.getItem(i).toString().contains(aircraft.toString())) {
                        spinner3.setSelection(0)
                    }
                }
            } else {
                val value = documentSnapshot.getString("Type")
                setSpinText(spinner3, value)
            }
            spinner3.isEnabled=false


            val spinner2: Spinner = findViewById(R.id.spinner_dimen)
            if (documentSnapshot.getString("Dim") == "-") {
                for (i in 0 until spinner2.adapter.count) {
                    if
(spinner2.adapter.getItem(i).toString().contains(aircraft.toString())) {
                        spinner2.setSelection(0)
                    }
                }
            } else {
```

```
    val value = documentSnapshot.getString("Dim")
    setSpinText(spinner2, value)
}
spinner2.isEnabled=false

val txtata = findViewById<TextView>(R.id.inputATA)
val ata = documentSnapshot.getString("ATA").toString()
txtata.text = ata
txtata.isFocusable=false
txtata.setBackgroundResource(R.drawable.custom_input)

val txtlocation = findViewById<TextView>(R.id.inputLocation)
val location = documentSnapshot.getString("Location").toString()
txtlocation.text = location
txtlocation.isFocusable=false
txtlocation.setBackgroundResource(R.drawable.custom_input)

val txtdimen1: TextView = findViewById(R.id.input_dimen_1)
val dimen1 = documentSnapshot.getString("Dim1").toString()
txtdimen1.text = dimen1
txtdimen1.isFocusable=false
txtdimen1.setBackgroundResource(R.drawable.custom_input)

val txtdimen2: TextView = findViewById(R.id.input_dimen_2)
val dimen2 = documentSnapshot.getString("Dim2").toString()
txtdimen2.text = dimen2
txtdimen2.isFocusable=false
txtdimen2.setBackgroundResource(R.drawable.custom_input)

val txtdimen3: TextView = findViewById(R.id.input_dimen_3)
val dimen3 = documentSnapshot.getString("Dim3").toString()
txtdimen3.text = dimen3
txtdimen3.isFocusable=false
txtdimen3.setBackgroundResource(R.drawable.custom_input)

val txtfh: TextView = findViewById(R.id.input_FH)
val fh = documentSnapshot.getString("Inspection FH").toString()
txtfh.text = fh
txtfh.isFocusable=false
txtfh.setBackgroundResource(R.drawable.custom_input)

val txtfc: TextView = findViewById(R.id.input_FC)
val fc = documentSnapshot.getString("Inspection FC").toString()
txtfc.text = fc
txtfc.isFocusable=false
txtfc.setBackgroundResource(R.drawable.custom_input)

val txtdays: TextView= findViewById(R.id.input_Days)
val days = documentSnapshot.getString("Inspection days").toString()
txtdays.text = days
txtdays.isFocusable=false
txtdays.setBackgroundResource(R.drawable.custom_input)

val txtfh2: TextView = findViewById(R.id.input_FH_repair)
val fh2 = documentSnapshot.getString("Repair FH").toString()
txtfh2.text = fh2
txtfh2.isFocusable=false
txtfh2.setBackgroundResource(R.drawable.custom_input)

val txtfc2: TextView = findViewById(R.id.input_FC_repair)
val fc2 = documentSnapshot.getString("Repair FC").toString()
txtfc2.text = fc2
txtfc2.isFocusable=false
txtfc2.setBackgroundResource(R.drawable.custom_input)

val txtdays2: TextView= findViewById(R.id.input_Days_repair)
val days2 = documentSnapshot.getString("Repair days").toString()
txtdays2.text = days2
txtdays2.isFocusable=false
txtdays2.setBackgroundResource(R.drawable.custom_input)

val txtentry: TextView = findViewById(R.id.input_entry)
val entry = documentSnapshot.getString("Entry").toString()
txtentry.text = entry

val txtclosing: TextView = findViewById(R.id.input_closing)
val closing = documentSnapshot.getString("Closing").toString()
txtclosing.text = closing

val txtpn: TextView = findViewById(R.id.input_PN)
val pn = documentSnapshot.getString("PN").toString()
txtpn.text = pn
txtpn.isFocusable=false
txtpn.setBackgroundResource(R.drawable.custom_input)

val txtsn: TextView = findViewById(R.id.input_SN)
val sn = documentSnapshot.getString("SN").toString()
txtsn.text = sn
txtsn.isFocusable=false
txtsn.setBackgroundResource(R.drawable.custom_input)

val txtref: TextView = findViewById(R.id.input_ref)
val ref = documentSnapshot.getString("References").toString()
txtref.text = ref
txtref.isFocusable=false

val txtcomment: TextView= findViewById(R.id.inputComment)
val comment = documentSnapshot.getString("Comments").toString()
txtcomment.text = comment
txtcomment.isFocusable=false
```

```
            txtcomment.setBackgroundResource(R.drawable.custom_input)

        }

    }

    private fun setSpinText(spin: Spinner, text: String?) {
        for (i in 0 until spin.adapter.count) {
            if (spin.adapter.getItem(i).toString().contains(text.toString())) {
                spin.setSelection(i)
            }
        }
    }

}
```

A.10. Documents activity

```
class DocumentsActivity : AppCompatActivity() , MyItemClickListener,
MyItemLongClickListener {

    private var mAdapter: DocumentAdapter? = null

    var textViewStatus: TextView? = null
    var progressBar: ProgressBar? = null
    var fileName : String? = null

    var mStorageReference: StorageReference? = null

    override fun onCreate(savedInstanceState: Bundle?) {

        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_documents)

        val text: TextView = findViewById(R.id.tool_title)
        text.text = "Aircraft documentation"

        mStorageReference = FirebaseStorage.getInstance().reference

        textViewStatus = findViewById<View>(R.id.textViewStatus) as TextView
        progressBar = findViewById<View>(R.id.progressbar) as ProgressBar

        showMenu()

        val back = findViewById<View>(R.id.btn_back) as ImageButton
        back.setOnClickListener {
            super.onBackPressed()
            finish()
        }
        val btn_search : Button = findViewById(R.id.button_search)
        btn_search.onFocusChangeListener = View.OnFocusChangeListener { a,
hasFocus ->
            if (!hasFocus) {
                btn_search.setHint("Search")
                hideKeyboard(a)
            } else btn_search.hint = ""
        }

        btn_search.setOnClickListener {
            val search : EditText = findViewById(R.id.search)
            if (search.text.toString().isEmpty()) {

            } else {
                searchBar()
            }
        }

        val btn_reset: Button = findViewById(R.id.button_reset)
        btn_reset.setOnClickListener {
            initData()
        }

        initView()
        initData()

    }

    private fun searchBar() {
        val SHARED_PREF_AIRCRAFT = "aircraft"
        val KEY_AIRCRAFT = "key_aircraft"
        val sp = getSharedPreferences(SHARED_PREF_AIRCRAFT,
MODE_PRIVATE)
        val aircraft = sp.getString(KEY_AIRCRAFT, null)

        val search : EditText = findViewById(R.id.search)
        val db : FirebaseFirestore= FirebaseFirestore.getInstance()
        val rvDocuments =
findViewById<RecyclerView>(R.id.recyclerViewDocumentList)
        db.collection("Fleet").document("$aircraft")
                .collection("Documentation")
                .whereEqualTo("Name",
search.text.toString()).get().addOnSuccessListener { documents ->
                    val mData: ArrayList<DocumentModel> = ArrayList()
                    for (document in documents) {
                        val taskItem: DocumentModel =
document.toObject(DocumentModel::class.java)
                        mData.add(taskItem)
                    }
                mAdapter = DocumentAdapter(mData)
                rvDocuments.adapter = mAdapter
                mAdapter!!.setOnItemClickListener(this)
                mAdapter!!.setOnItemLongClickListener(this)
                mAdapter!!.notifyDataSetChanged()
            }
    }

    private fun hideKeyboard(view: View) {
        val inputMethodManager =
getSystemService(INPUT_METHOD_SERVICE) as InputMethodManager
        inputMethodManager.hideSoftInputFromWindow(view.windowToken, 0)
    }

    private fun initView() {
        val rvDocuments =
findViewById<RecyclerView>(R.id.recyclerViewDocumentList)
        rvDocuments.layoutManager = LinearLayoutManager(this)
        rvDocuments.itemAnimator = DefaultItemAnimator()
    }

    private fun initData() {

        val SHARED_PREF_AIRCRAFT = "aircraft"
        val KEY_AIRCRAFT = "key_aircraft"
        val sp = getSharedPreferences(SHARED_PREF_AIRCRAFT,
MODE_PRIVATE)
        val aircraft = sp.getString(KEY_AIRCRAFT, null)

        val db: FirebaseFirestore = FirebaseFirestore.getInstance()
        val collection: CollectionReference = db.collection("Fleet")
            .document("$aircraft").collection("Documentation")
        val query: Query = collection
        val rvDocuments =
findViewById<RecyclerView>(R.id.recyclerViewDocumentList)
        query.get().addOnCompleteListener { task ->

            if (task.isSuccessful) {
                val mData: ArrayList<DocumentModel> = ArrayList()
                for (document in task.result) {
                    val taskItem: DocumentModel =
document.toObject(DocumentModel::class.java)
                    mData.add(taskItem)
                }
                mAdapter = DocumentAdapter(mData)
                rvDocuments.adapter = mAdapter
                mAdapter!!.setOnItemClickListener(this)
                mAdapter!!.setOnItemLongClickListener(this)
                mAdapter!!.notifyDataSetChanged()
            }
        }

    }

    private fun getFile() {
            val intent = Intent()
            intent.type = "application/pdf"
            intent.action = Intent.ACTION_GET_CONTENT
            startActivityForResult(Intent.createChooser(intent, "Select Picture"),
PICK_PDF_CODE)
    }

    override fun onActivityResult(requestCode: Int, resultCode: Int, data:
Intent?) {
        super.onActivityResult(requestCode, resultCode, data)

        if (requestCode == PICK_PDF_CODE && resultCode == RESULT_OK &&
data != null && data.data != null) {
            val documentFile = DocumentFile.fromSingleUri(this, data.data!!)
            fileName = documentFile!!.name
            if (data.data != null) {
                uploadFile(data.data)
                mAdapter!!.notifyDataSetChanged()

            } else {
                Toast.makeText(this, "No file chosen",
Toast.LENGTH_SHORT).show()
            }
        }
    }

    private fun uploadFile(data: Uri?) {

        val SHARED_PREF_AIRCRAFT = "aircraft"
        val KEY_AIRCRAFT = "key_aircraft"
        val sp = getSharedPreferences(SHARED_PREF_AIRCRAFT,
MODE_PRIVATE)
        val aircraft = sp.getString(KEY_AIRCRAFT, null)

        val sRef =
mStorageReference!!.child("documentation/fleet/$aircraft/$fileName")
        progressBar!!.visibility = View.VISIBLE

        sRef.putFile(data!!)
                .addOnSuccessListener { taskSnapshot ->
                    progressBar!!.visibility = View.GONE
                    textViewStatus!!.text = "File Uploaded Successfully"
                    val user = FirebaseAuth.getInstance().currentUser
                    val userEmail = user.email
                    val db = FirebaseFirestore.getInstance()
                val df = db.collection("Users").document(userEmail)
                df.get().addOnSuccessListener { documentSnapshot ->
                    val fullName = documentSnapshot.getString("Name")
                    val df2 = db.collection("Fleet").document("$aircraft")
                            .collection("Documentation")
                            .document("$fileName")
                    val revisionInfo = hashMapOf(
                        "User" to fullName.toString(),
                        "Email" to userEmail.toString(),
                        "Name" to "$fileName",
                    )
                    df2.set(revisionInfo)
                    startActivity(intent)
                    finish()
                }
            }
            .addOnFailureListener { exception -> Toast.makeText(
                applicationContext,
                exception.message,
                Toast.LENGTH_LONG
            ).show() }
            .addOnProgressListener { taskSnapshot ->
                val progress = 100 * taskSnapshot.bytesTransferred /
taskSnapshot.totalByteCount
                textViewStatus!!.text = "$progress % Uploading..."

            }

    }

    private fun showMenu() {

        val button : ImageButton = findViewById(R.id.btn_menu)

        button.setOnClickListener {

            val inflater = LayoutInflater.from(this)
            val popupView: View = inflater.inflate(R.layout.menu, null)
            val focusable = true
            val popupWindow = PopupWindow(popupView, 500, 1000, focusable)
            button.x.toInt()
            button.y.toInt()
            popupWindow.showAsDropDown(button)

popupWindow.contentView.findViewById<LinearLayout>(R.id.sortBy).visibility =
View.GONE

popupWindow.contentView.findViewById<LinearLayout>(R.id.advancedSearch)
.visibility = View.GONE
            popupWindow.contentView.findViewById<TextView>(R.id.add_text).text
```

```
= "ADD DOCUMENT"

popupWindow.contentView.findViewById<LinearLayout>(R.id.add).setOnClickLi
stener {
                getFile()
                popupWindow.dismiss()
            }

            popupView.setOnClickListener { popupWindow.dismiss() }
            popupView.setOnTouchListener { v, event ->
                popupWindow.dismiss()
                true
            }
        }
    }

    companion object {
        const val PICK_PDF_CODE = 2342
    }

    override fun onItemClick(view: View?, position: Int) {

        val SHARED_PREF_AIRCRAFT = "aircraft"
        val KEY_AIRCRAFT = "key_aircraft"
        val sp = getSharedPreferences(SHARED_PREF_AIRCRAFT,
MODE_PRIVATE)
        val aircraft = sp.getString(KEY_AIRCRAFT, null)

        val db: FirebaseFirestore = FirebaseFirestore.getInstance()
        val collection: CollectionReference = db.collection("Fleet")
                .document("$aircraft").collection("Documentation")
        val query: Query = collection
        query.get().addOnCompleteListener { task ->
            if (task.isSuccessful) {
                val mData: ArrayList<DocumentModel> = ArrayList()
                for (document in task.result) {
                    val taskItem: DocumentModel =
document.toObject(DocumentModel::class.java)
                    mData.add(taskItem)
                }

                val bean: DocumentModel = mData[position - 1]
                val file = bean.Name

                val inflater = LayoutInflater.from(this)
                val popupView: View =
inflater.inflate(R.layout.pop_window_download, null)
                val focusable = true
                val popupWindow = PopupWindow(popupView, 900, 200, focusable)
                popupWindow.showAtLocation(view, Gravity.CENTER, 0, 0)
popupWindow.contentView.findViewById<Button>(R.id.pop_delete_document).
setOnClickListener {
                popupWindow.dismiss()

                val inflater3 = LayoutInflater.from(this)
                val popupView3: View = inflater3.inflate(R.layout.pop_window_pin,
null)
                val focusable3 = true
                val popupWindow3 = PopupWindow(popupView3, 400, 400,
focusable3)
                popupWindow3.showAtLocation(view, Gravity.CENTER, 0, 0)


popupWindow3.contentView.findViewById<TextView>(R.id.tool_title).text =
"Write PIN"

popupWindow3.contentView.findViewById<ImageButton>(R.id.btn_back).visibili
ty = View.GONE

popupWindow3.contentView.findViewById<ImageButton>(R.id.btn_menu).visibil
ity = View.GONE

                val user = FirebaseAuth.getInstance().currentUser
                val userEmail = user.email
                val df = db.collection("Users").document(userEmail)
                df.get().addOnSuccessListener { documentSnapshot ->
                    val txtpin =
popupWindow3.contentView.findViewById<TextView>(R.id.inputPIN)
                    val delete: Button =
popupWindow3.contentView.findViewById(R.id.pop_pin)
                    delete.setOnClickListener {
                        val builder = AlertDialog.Builder(this)
                        builder.setTitle("CAUTION")
                        builder.setMessage("Are you sure you want to continue?")
                        builder.setPositiveButton("Accept") { dialog, which ->
                            if (documentSnapshot.getString("PIN") ==
txtpin.text.toString()) {
                                db.collection("Fleet").document("$aircraft")
                                    .collection("Documentation")
                                    .document("$file").delete()
                                FirebaseStorage.getInstance()
                                    .getReferenceFromUrl("gs://airdocs-
1414.appspot.com/documentation/fleet/$aircraft/$fileName").delete()
                                popupWindow3.dismiss()
                                startActivity(intent)
                            } else {
                                val builder1 = AlertDialog.Builder(this)
                                builder1.setTitle("Error")
                                builder1.setMessage("Incorrect PIN, please try
again.")
                                builder1.setPositiveButton("Accept", null)
                                val dialog1: AlertDialog = builder1.create()
                                dialog1.show()
                                popupWindow3.dismiss()
                            }
                        }
                        builder.setNegativeButton("Cancel", null)
                        val dialog: AlertDialog = builder.create()
                        dialog.show()
                    }
                }
            }


popupWindow.contentView.findViewById<Button>(R.id.pop_download).setOnCl
ickListener {

                val storageReference = FirebaseStorage.getInstance()
                    .getReferenceFromUrl("gs://airdocs-
1414.appspot.com/documentation/fleet/$aircraft/$fileName")

                storageReference.downloadUrl.addOnSuccessListener { uri ->

                    val request =
DownloadManager.Request(Uri.parse(uri.toString()))

                    request.setTitle("$fileName")

request.setNotificationVisibility(DownloadManager.Request.VISIBILITY_VISIBL
E_NOTIFY_COMPLETED)
                    val downloadManager =
                        getSystemService(DOWNLOAD_SERVICE) as
DownloadManager
                    downloadManager.enqueue(request)

                    Toast.makeText(
                        this,
                        uri.toString(), Toast.LENGTH_LONG
                    ).show();

                }.addOnFailureListener {
                    val builder = AlertDialog.Builder(this)
                    builder.setTitle("Information")
                    builder.setMessage("Failed on retrieving selected file")
                    builder.setPositiveButton("Accept") { dialog, which ->
                    }
                    val dialog: AlertDialog = builder.create()
                    dialog.show()
                }
            }
        }
    }
}

override fun onItemLongClick(view: View?, postion: Int) {
    TODO("Not yet implemented")
}
}
```

A.11. Damage documents activity

```
class DocumentsDamageActivity : AppCompatActivity() , MyItemClickListener,
MyItemLongClickListener {

    private var mAdapter: DocumentDamagesAdapter? = null

    var textViewStatus: TextView? = null
    var progressBar: ProgressBar? = null
    var fileName : String? = null

    var mStorageReference: StorageReference? = null

    override fun onCreate(savedInstanceState: Bundle?) {

        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_documents)

        val text: TextView = findViewById(R.id.tool_title)
        text.text = "Damage documentation"

        mStorageReference = FirebaseStorage.getInstance().reference

        textViewStatus = findViewById<View>(R.id.textViewStatus) as TextView
        progressBar = findViewById<View>(R.id.progressbar) as ProgressBar

        showMenu()

        val back = findViewById<View>(R.id.btn_back) as ImageButton
        back.setOnClickListener {
            super.onBackPressed()
        }

        val btn_search : Button = findViewById(R.id.button_search)
        btn_search.onFocusChangeListener = View.OnFocusChangeListener { a,
hasFocus ->
            if (!hasFocus) {
                btn_search.setHint("Search")
            hideKeyboard(a)
         } else btn_search.hint = ""
      }

      btn_search.setOnClickListener {
         val search : EditText = findViewById(R.id.search)
         if (search.text.toString().isEmpty()) {

         } else {
            searchBar()
         }
      }

      val btn_reset: Button = findViewById(R.id.button_reset)
      btn_reset.setOnClickListener {
         initData()
      }

      initView()
      initData()

   }

   private fun searchBar() {

      val SHARED_PREF_AIRCRAFT = "aircraft"
      val KEY_AIRCRAFT = "key_aircraft"
      val sp = getSharedPreferences(SHARED_PREF_AIRCRAFT,
MODE_PRIVATE)
      val aircraft = sp.getString(KEY_AIRCRAFT, null)

      val search : EditText = findViewById(R.id.search)
      val db : FirebaseFirestore= FirebaseFirestore.getInstance()
      val rvDocuments =
findViewById<RecyclerView>(R.id.recyclerViewDocumentList)
      db.collection("Fleet").document("$aircraft")
            .collection("Documentation")
            .whereEqualTo("Name",
search.text.toString()).get().addOnSuccessListener { documents ->
               val mData: ArrayList<DocumentDamagesModel> = ArrayList()
               for (document in documents) {
                  val taskItem: DocumentDamagesModel =
document.toObject(DocumentDamagesModel::class.java)
                  mData.add(taskItem)
               }
               mAdapter = DocumentDamagesAdapter(mData)
               rvDocuments.adapter = mAdapter
               mAdapter!!.setOnItemClickListener(this)
               mAdapter!!.setOnItemLongClickListener(this)
               mAdapter!!.notifyDataSetChanged()
            }
    }

    private fun hideKeyboard(view: View) {
        val inputMethodManager =
getSystemService(INPUT_METHOD_SERVICE) as InputMethodManager
        inputMethodManager.hideSoftInputFromWindow(view.windowToken, 0)
    }

    private fun initView() {
        val rvDocuments =
findViewById<RecyclerView>(R.id.recyclerViewDocumentList)
        rvDocuments.layoutManager = LinearLayoutManager(this)
        rvDocuments.itemAnimator = DefaultItemAnimator()
    }

    private fun initData() {

        val SHARED_PREF_plane = "plane"
        val KEY_plane = "key_plane"
        val sp = getSharedPreferences(SHARED_PREF_plane,
MODE_PRIVATE)
        val id = sp.getString(KEY_plane, null)

        val SHARED_PREF_DAMAGE = "DAMAGE"
        val KEY_DAMAGE = "key_DAMAGE"
        val sp2 = getSharedPreferences(SHARED_PREF_DAMAGE,
MODE_PRIVATE)
        val damage_id = sp2.getString(KEY_DAMAGE, null)

        val db: FirebaseFirestore = FirebaseFirestore.getInstance()
        val collection: CollectionReference = db.collection("Aircraft
revision").document("$id")
            .collection("New damages").document("$damage_id")
            .collection("Documentation")

        val query: Query = collection
        val rvDocuments =
findViewById<RecyclerView>(R.id.recyclerViewDocumentList)
        query.get().addOnCompleteListener { task ->

            if (task.isSuccessful) {
                val mData: ArrayList<DocumentDamagesModel> = ArrayList()
                for (document in task.result) {
                    val taskItem: DocumentDamagesModel =
document.toObject(DocumentDamagesModel::class.java)
                    mData.add(taskItem)
                }
                mAdapter = DocumentDamagesAdapter(mData)
                rvDocuments.adapter = mAdapter
            mAdapter!!.setOnItemClickListener(this)
            mAdapter!!.setOnItemLongClickListener(this)
            mAdapter!!.notifyDataSetChanged()
         }
      }

   }

   private fun getFile() {
         val intent = Intent()
         intent.type = "application/pdf"
         intent.action = Intent.ACTION_GET_CONTENT
         startActivityForResult(Intent.createChooser(intent, "Select Picture"),
PICK_PDF_CODE)
   }

   override fun onActivityResult(requestCode: Int, resultCode: Int, data:
Intent?) {
      super.onActivityResult(requestCode, resultCode, data)

      if (requestCode == PICK_PDF_CODE && resultCode == RESULT_OK &&
data != null && data.data != null) {
         val documentFile = DocumentFile.fromSingleUri(this, data.data!!)
         fileName = documentFile!!.name
         if (data.data != null) {
            uploadFile(data.data)
         } else {
            Toast.makeText(this, "No file chosen",
Toast.LENGTH_SHORT).show()
         }
      }
   }

   private fun uploadFile(data: Uri?) {

      val SHARED_PREF_plane = "plane"
      val KEY_plane = "key_plane"
      val sp = getSharedPreferences(SHARED_PREF_plane,
MODE_PRIVATE)
      val id = sp.getString(KEY_plane, null)

      val SHARED_PREF_DAMAGE = "DAMAGE"
      val KEY_DAMAGE = "key_DAMAGE"
      val sp2 = getSharedPreferences(SHARED_PREF_DAMAGE,
MODE_PRIVATE)
      val damage_id = sp2.getString(KEY_DAMAGE, null)

      val sRef =
mStorageReference!!.child("documentation/damages/$id/$damage_id/$fileN
ame")
```

```
        progressBar!!.visibility = View.VISIBLE

        sRef.putFile(data!!)
                .addOnSuccessListener { taskSnapshot ->
                    progressBar!!.visibility = View.GONE
                    textViewStatus!!.text = "File Uploaded Successfully"
                    val user = FirebaseAuth.getInstance().currentUser
                    val userEmail = user.email
                    val db = FirebaseFirestore.getInstance()
                    val df = db.collection("Users").document(userEmail)
                    df.get().addOnSuccessListener { documentSnapshot ->
                        val fullName = documentSnapshot.getString("Name")
                        val df2 = db.collection("Aircraft revision").document("$id")
                                .collection("New damages").document("$damage_id")
                                .collection("Documentation").document("$fileName")
                        val revisionInfo = hashMapOf(
                            "User" to fullName.toString(),
                            "Email" to userEmail.toString(),
                            "Name" to "$fileName",
                        )
                        df2.set(revisionInfo)
                        startActivity(intent)
                    }
                }
                .addOnFailureListener { exception -> Toast.makeText(
                    applicationContext,
                    exception.message,
                    Toast.LENGTH_LONG
                ).show() }
                .addOnProgressListener { taskSnapshot ->
                    val progress = 100 * taskSnapshot.bytesTransferred /
taskSnapshot.totalByteCount
                    textViewStatus!!.text = "$progress % Uploading..."

                }

    }

    private fun showMenu() {

        val button : ImageButton = findViewById(R.id.btn_menu)

        button.setOnClickListener {

            val inflater = LayoutInflater.from(this)
            val popupView: View = inflater.inflate(R.layout.menu, null)
            val focusable = true
            val popupWindow = PopupWindow(popupView, 500, 1000, focusable)
            button.x.toInt()
            button.y.toInt()
```

```
         popupWindow.showAsDropDown(button)

popupWindow.contentView.findViewById<LinearLayout>(R.id.sortBy).visibility =
View.GONE

popupWindow.contentView.findViewById<LinearLayout>(R.id.advancedSearch)
.visibility = View.GONE
         popupWindow.contentView.findViewById<TextView>(R.id.add_text).text
= "ADD DOCUMENT"

popupWindow.contentView.findViewById<LinearLayout>(R.id.add).setOnClickLi
stener {
            getFile()
            popupWindow.dismiss()
         }

         popupView.setOnClickListener { popupWindow.dismiss() }
         popupView.setOnTouchListener { v, event ->
            popupWindow.dismiss()
            true
         }
      }
   }

   companion object {
      const val PICK_PDF_CODE = 2342
   }

   override fun onItemClick(view: View?, position: Int) {

      val SHARED_PREF_plane = "plane"
      val KEY_plane = "key_plane"
      val sp = getSharedPreferences(SHARED_PREF_plane,
MODE_PRIVATE)
      val id = sp.getString(KEY_plane, null)

      val SHARED_PREF_DAMAGE = "DAMAGE"
      val KEY_DAMAGE = "key_DAMAGE"
      val sp2 = getSharedPreferences(SHARED_PREF_DAMAGE,
MODE_PRIVATE)
      val damage_id = sp2.getString(KEY_DAMAGE, null)

      val db: FirebaseFirestore = FirebaseFirestore.getInstance()
      val collection: CollectionReference = db.collection("Aircraft
revision").document("$id")
            .collection("New damages").document("$damage_id")
            .collection("Documentation")
      val query: Query = collection
      query.get().addOnCompleteListener { task ->
         if (task.isSuccessful) {
            val mData: ArrayList<DocumentDamagesModel> = ArrayList()
            for (document in task.result) {
               val taskItem: DocumentDamagesModel =
document.toObject(DocumentDamagesModel::class.java)
               mData.add(taskItem)
            }

            val bean: DocumentDamagesModel = mData[position - 1]
            val file = bean.Name

            val inflater = LayoutInflater.from(this)
            val popupView: View =
inflater.inflate(R.layout.pop_window_download, null)
            val focusable = true
            val popupWindow = PopupWindow(popupView, 900, 200, focusable)
            popupWindow.showAtLocation(view, Gravity.CENTER, 0, 0)


popupWindow.contentView.findViewById<Button>(R.id.pop_delete_document).
setOnClickListener {
               popupWindow.dismiss()

               val inflater3 = LayoutInflater.from(this)
               val popupView3: View = inflater3.inflate(R.layout.pop_window_pin,
null)
               val focusable3 = true
               val popupWindow3 = PopupWindow(popupView3, 400, 400,
focusable3)
               popupWindow3.showAtLocation(view, Gravity.CENTER, 0, 0)


popupWindow3.contentView.findViewById<TextView>(R.id.tool_title).text =
"Write PIN"

popupWindow3.contentView.findViewById<ImageButton>(R.id.btn_back).visibili
ty = View.GONE

popupWindow3.contentView.findViewById<ImageButton>(R.id.btn_menu).visibil
ity = View.GONE

               val user = FirebaseAuth.getInstance().currentUser
               val userEmail = user.email
               val df = db.collection("Users").document(userEmail)
               df.get().addOnSuccessListener { documentSnapshot ->
                  val txtpin =
popupWindow3.contentView.findViewById<TextView>(R.id.inputPIN)
                  val delete: Button =
popupWindow3.contentView.findViewById(R.id.pop_pin)
                  delete.setOnClickListener {
                     val builder = AlertDialog.Builder(this)
```

```
                    builder.setTitle("CAUTION")
                    builder.setMessage("Are you sure you want to continue?")
                    builder.setPositiveButton("Accept") { dialog, which ->
                        if (documentSnapshot.getString("PIN") ==
txtpin.text.toString()) {
                            db.collection("Aircraft revision").document("$id")
                                .collection("New
damages").document("$damage_id")

.collection("Documentation").document("$file").delete()
                            FirebaseStorage.getInstance()
                                .getReferenceFromUrl("gs://airdocs-
1414.appspot.com/documentation/damages/$id/$damage_id/$file").delete()
                            popupWindow3.dismiss()
                            startActivity(intent)
                        } else {
                            val builder1 = AlertDialog.Builder(this)
                            builder1.setTitle("Error")
                            builder1.setMessage("Incorrect PIN, please try
again.")
                            builder1.setPositiveButton("Accept", null)
                            val dialog1: AlertDialog = builder1.create()
                            dialog1.show()
                            popupWindow3.dismiss()
                        }
                    }
                    builder.setNegativeButton("Cancel", null)
                    val dialog: AlertDialog = builder.create()
                    dialog.show()
                }
            }
        }


popupWindow.contentView.findViewById<Button>(R.id.pop_download).setOnCl
ickListener {

                val storageReference = FirebaseStorage.getInstance()
                    .getReferenceFromUrl("gs://airdocs-
1414.appspot.com/documentation/damages/$id/$damage_id/$file")

                storageReference.downloadUrl.addOnSuccessListener { uri ->

                    val request =
DownloadManager.Request(Uri.parse(uri.toString()))

                    request.setTitle("$file")

request.setNotificationVisibility(DownloadManager.Request.VISIBILITY_VISIBL
E_NOTIFY_COMPLETED)
```

```
                        val downloadManager =
                            getSystemService(DOWNLOAD_SERVICE) as
DownloadManager
                        downloadManager.enqueue(request)

                        Toast.makeText(
                            this,
                            uri.toString(), Toast.LENGTH_LONG
                        ).show();

                    }.addOnFailureListener {
                        val builder = AlertDialog.Builder(this)
                        builder.setTitle("Information")
                        builder.setMessage("Failed on retrieving selected file")
                        builder.setPositiveButton("Accept") { dialog, which ->
                        }
                        val dialog: AlertDialog = builder.create()
                        dialog.show()
                    }
                }

            }
        }
    }

    override fun onItemLongClick(view: View?, postion: Int) {
        TODO("Not yet implemented")
    }
```

A.12. Edit check activity

```
class EditCheckActivity : AppCompatActivity() {

        var timestamp_end : String? = null
        var timestamp_crs : String? = null

        override fun onCreate(savedInstanceState: Bundle?) {

            super.onCreate(savedInstanceState)
            setContentView(R.layout.activity_addcheck)

            val title : TextView = findViewById(R.id.tool_title)
            title.text = "Edit check"

            val btn_menu : ImageButton = findViewById(R.id.btn_menu)
            btn_menu.visibility = View.GONE

            val aircraft = findViewById<EditText>(R.id.inputAircraft)
            aircraft.onFocusChangeListener = View.OnFocusChangeListener { b,
hasFocus ->
                if (!hasFocus) {
                    aircraft.setHint(R.string.NoHint)
                    hideKeyboard(b)
                } else aircraft.hint = ""
            }

            val assign = findViewById<EditText>(R.id.inputAssign)
            assign.onFocusChangeListener = View.OnFocusChangeListener { a,
hasFocus ->
                if (!hasFocus) {
                    assign.setHint(R.string.NoHint)
                    hideKeyboard(a)
                } else assign.hint = ""
            }

            val revision = resources.getStringArray(R.array.revision_array)
            val spinner: Spinner = findViewById(R.id.spinner_check)
            val adapter = ArrayAdapter(this, R.layout.spinner_item_selected,
revision)
            adapter.setDropDownViewResource(R.layout.spinner_dropdown_item)
            spinner.adapter = adapter
            spinner.avoidDropdownFocus()
            spinner.onItemSelectedListener = object :
AdapterView.OnItemSelectedListener {
                override fun onItemSelected(
                        parent: AdapterView<*>,
                        view: View?,
                        position: Int,
                        id: Long
                ) {
                    if (parent.getItemAtPosition(position) == "-") {
                    } else {
                        val SHARED_PREF_CHECK= "CHECK";
                        val KEY_CHECK= "key_CHECK";
                        val mSpnValue: String = spinner.selectedItem.toString()
                        val myPrefs: SharedPreferences =
getSharedPreferences(SHARED_PREF_CHECK, MODE_PRIVATE)
                        val prefsEditor = myPrefs.edit()
                        prefsEditor.putString(KEY_CHECK, mSpnValue)
                        prefsEditor.apply()
                    }
                }

                override fun onNothingSelected(parent: AdapterView<*>?) {}
            }

            val status = resources.getStringArray(R.array.status_array)
            val spinner2: Spinner = findViewById(R.id.spinner_status)
            val adapter2 = ArrayAdapter(this, R.layout.spinner_item_selected,
status)
            adapter2.setDropDownViewResource(R.layout.spinner_dropdown_item)
            spinner2.adapter = adapter2
            spinner.avoidDropdownFocus()
            spinner2.onItemSelectedListener = object :
AdapterView.OnItemSelectedListener {
                override fun onItemSelected(
                        parent: AdapterView<*>,
                        view: View?,
                        position: Int,
                        id: Long
                ) {
                    if (parent.getItemAtPosition(position) == "-") {
                    } else {
                        val SHARED_PREF_STATUS = "STATUS";
                        val KEY_STATUS= "key_STATUS";
                        val mSpnValue: String = spinner2.selectedItem.toString()
                        val myPrefs: SharedPreferences =
getSharedPreferences(SHARED_PREF_STATUS, MODE_PRIVATE)
                        val prefsEditor = myPrefs.edit()
                        prefsEditor.putString(KEY_STATUS, mSpnValue)
                        prefsEditor.apply()
                    }
                }

                override fun onNothingSelected(parent: AdapterView<*>?) {}
            }

            val calendar = Calendar.getInstance()
            val edit = findViewById<View>(R.id.inputBegin) as EditText
            val yy = calendar.get(Calendar.YEAR)
            val mm = calendar.get(Calendar.MONTH)
            val dd = calendar.get(Calendar.DAY_OF_MONTH)
            val datePicker = DatePickerDialog(
                    this,
                    { _, year, monthOfYear, dayOfMonth ->
                        val date =
                                (dayOfMonth.toString() + "/" + (monthOfYear + 1).toString()
+ "/" + year.toString())
                        edit.setText(date)
                    },
                    yy,
                    mm,
                    dd
            )
            datePicker.setButton(DialogInterface.BUTTON_NEUTRAL, "CLEAR") {
_, which ->
                if (which == DialogInterface.BUTTON_NEUTRAL) {
                    edit.setText("-")
                }
            }

            edit.setOnClickListener() {
                datePicker.show()
            }

            val calendar2 = Calendar.getInstance()
            val edit2 = findViewById<View>(R.id.inputEnd) as EditText
            val yy2 = calendar2.get(Calendar.YEAR)
            val mm2 = calendar2.get(Calendar.MONTH)
            val dd2 = calendar2.get(Calendar.DAY_OF_MONTH)
            val datePicker2 = DatePickerDialog(
                    this,
                    { _, year, monthOfYear, dayOfMonth ->
                        val date2 =
                                (dayOfMonth.toString() + "/" + (monthOfYear + 1).toString()
+ "/" + year.toString())
                        edit2.setText(date2)
                    },
                    yy2,
                    mm2,
                    dd2
            )
            datePicker2.setButton(DialogInterface.BUTTON_NEUTRAL, "CLEAR")
{ _, which ->
                if (which == DialogInterface.BUTTON_NEUTRAL) {
                    edit2.setText("-")
                }
            }

            edit2.setOnClickListener() {
                datePicker2.show()
            }

            val calendar3 = Calendar.getInstance()
            val edit3 = findViewById<View>(R.id.inputCRS) as EditText
            val yy3 = calendar3.get(Calendar.YEAR)
            val mm3 = calendar3.get(Calendar.MONTH)
            val dd3 = calendar3.get(Calendar.DAY_OF_MONTH)
            val datePicker3 = DatePickerDialog(
                    this,
                    DatePickerDialog.OnDateSetListener { _, year, monthOfYear,
dayOfMonth ->
                        val date3 =
                                (dayOfMonth.toString() + "/" + (monthOfYear + 1).toString()
+ "/" + year.toString())
                        edit3.setText(date3)
                    },
                    yy3,
                    mm3,
                dd3
            )
            datePicker3.setButton(DialogInterface.BUTTON_NEUTRAL, "CLEAR")
{ _, which ->
                if (which == DialogInterface.BUTTON_NEUTRAL) {
                    edit3.setText("-")
                }
            }

            edit3.setOnClickListener() {
                datePicker3.show()
            }

            val add: Button = findViewById(R.id.btn_Add)
            add.setOnClickListener {
                updateData()
            }

            assign.setOnClickListener {
                saveEditText()
                val i = Intent(this, AddUserToEditCheck::class.java)
                startActivity(i)
                finish()
            }

            val pref2 = getSharedPreferences("USER", MODE_PRIVATE)
            val name = pref2.getString("user", "")
            if(name  == null) { }
            else{
                assign.setText(name)
            }

            val remove_plane : ImageButton = findViewById(R.id.ivCancel)
            remove_plane.visibility = View.GONE

            val remove_assign : ImageButton = findViewById(R.id.ivCancel2)
            remove_assign.setOnClickListener {
                assign.setText("-")
            }

            val settings = getSharedPreferences(PREFS_NAME, 0)
            val firstStart = settings.getBoolean("firstStart", true)

            if (firstStart) {
                displayInfo()
                val editor = settings.edit()
                editor.putBoolean("firstStart", false)
                editor.apply()
            } else {
                displayEditText()
            }

            val back = findViewById<View>(R.id.btn_back) as ImageButton
            back.setOnClickListener {
                pref2.edit().remove("user").apply()
                deletePreference()
                val intent = Intent(this, CheckListActivity::class.java)
                startActivity(intent)
                finish()
            }

        }

    private fun displayEditText() {

        val SHARED_PREF_START = "start";
        val KEY_START = "key_start";
        val sp = getSharedPreferences(SHARED_PREF_START,
MODE_PRIVATE)
        val start = sp.getString(KEY_START, null)
        val editText1 = findViewById<TextView>(R.id.inputBegin)
        if (start != null) {
            editText1.text = "$start"
        }

        val SHARED_PREF_END = "END";
        val KEY_END = "END";
        val sp1 = getSharedPreferences(SHARED_PREF_END,
MODE_PRIVATE)
        val END = sp1.getString(KEY_END, null)
        val editText2 = findViewById<TextView>(R.id.inputEnd)
        if (END  != null) {
            editText2.text = "$END"
        }

        val SHARED_PREF_CRS = "CRS";
        val KEY_CRS = "CRS";
        val sp3 = getSharedPreferences(SHARED_PREF_CRS,
MODE_PRIVATE)
        val CRS = sp3.getString(KEY_CRS, null)
        val editText3 = findViewById<TextView>(R.id.inputCRS)
        if (CRS  != null) {
            editText3.text = "$CRS"
        }

        val SHARED_PREF_COM = "COM";
        val KEY_COM = "COM";
        val editText4 = findViewById<TextView>(R.id.inputComment)
        val sp4 = getSharedPreferences(SHARED_PREF_COM,
MODE_PRIVATE)
```

```
        val COM = sp4.getString(KEY_COM, null)
        if (COM  != null) {
            editText4.text = "$COM"
        }

        val SHARED_PREF_STATUS = "STATUS";
        val KEY_STATUS= "key_STATUS";
        val myPrefs = getSharedPreferences(SHARED_PREF_STATUS,
MODE_PRIVATE)
        val myString = myPrefs.getString(KEY_STATUS, null)
        val mSpinner = findViewById<Spinner>(R.id.spinner_status)
        setSpinText(mSpinner, myString)

        val SHARED_PREF_CHECK = "CHECK";
        val KEY_CHECK= "key_CHECK";
        val myPrefs1 = getSharedPreferences(SHARED_PREF_CHECK,
MODE_PRIVATE)
        val myString1 = myPrefs1.getString(KEY_CHECK, null)
        val mSpinner1 = findViewById<Spinner>(R.id.spinner_check)
        setSpinText(mSpinner1, myString1)

        val SHARED_PREF_AIRCRAFT = "aircraft";
        val KEY_AIRCRAFT = "key_aircraft";
        val sp5 = getSharedPreferences(SHARED_PREF_AIRCRAFT,
MODE_PRIVATE)
        val aircraft = sp5.getString(KEY_AIRCRAFT, null)
        val text = findViewById<TextView>(R.id.inputAircraft)
        if (aircraft != null) {
            text.text = "$aircraft"
        }

    }

    private fun saveEditText() {

        val SHARED_PREF_START = "start";
        val KEY_START = "key_start";
        val editText = findViewById<TextView>(R.id.inputBegin)
        val start: String = editText.text.toString()
        val sp = getSharedPreferences(SHARED_PREF_START,
MODE_PRIVATE)
        val editor = sp.edit()
        editor.putString(KEY_START, start)
        editor.apply()

        val SHARED_PREF_END = "END";
        val KEY_END = "END";
        val editText1 = findViewById<TextView>(R.id.inputEnd)
        val end : String = editText1.text.toString()
        val sp1 = getSharedPreferences(SHARED_PREF_END,
MODE_PRIVATE)
        val editor1 = sp1.edit()
        editor1.putString(KEY_END, end)
        editor1.apply()

        val SHARED_PREF_CRS = "CRS";
        val KEY_CRS = "CRS";
        val editText2 = findViewById<TextView>(R.id.inputCRS)
        val crs : String = editText2.text.toString()
        val sp2 = getSharedPreferences(SHARED_PREF_CRS,
MODE_PRIVATE)
        val editor2 = sp2.edit()
        editor2.putString(KEY_CRS, crs)
        editor2.apply()

        val SHARED_PREF_COM = "COM";
        val KEY_COM = "COM";
        val editText3 = findViewById<TextView>(R.id.inputComment)
        val com : String = editText3.text.toString()
        val sp3 = getSharedPreferences(SHARED_PREF_COM,
MODE_PRIVATE)
        val editor3 = sp3.edit()
        editor3.putString(KEY_COM, com)
        editor3.apply()

        val SHARED_PREF_AIRCRAFT = "aircraft"
        val KEY_AIRCRAFT = "key_aircraft"
        val editText4: TextView = findViewById(R.id.inputAircraft)
        val aircraft : String = editText4.text.toString()
        val sp4 = getSharedPreferences(SHARED_PREF_AIRCRAFT,
MODE_PRIVATE)
        val editor4 = sp4.edit()
        editor4.putString(KEY_AIRCRAFT, aircraft)
        editor4.apply()

    }

    private fun displayInfo() {

            val SHARED_PREF_AIRCRAFT = "aircraft";
            val KEY_AIRCRAFT = "key_aircraft";
            val sp = getSharedPreferences(SHARED_PREF_AIRCRAFT,
MODE_PRIVATE)
            val aircraft = sp.getString(KEY_AIRCRAFT, null)
            val text = findViewById<TextView>(R.id.inputAircraft)
            if (aircraft != null) {
                text.text = "$aircraft"
            }

            val SHARED_PREF_START = "start"
            val KEY_START = "key_start"
            val sp1 = getSharedPreferences(SHARED_PREF_START,
MODE_PRIVATE)
            val start = sp1.getString(KEY_START, null)
            val textBegin = findViewById<TextView>(R.id.inputBegin)
            if (start !=null)
            {
                textBegin.text =  "$start"
            }

            val SHARED_PREF_check = "check"
            val KEY_check = "key_check"
            val sp2 = getSharedPreferences(SHARED_PREF_check,
MODE_PRIVATE)
            val check = sp2.getString(KEY_check, null)
            val textCheck = findViewById<Spinner>(R.id.spinner_check)
            if (check!=null)
            {
                setSpinText(textCheck, "$check")
            }

            val SHARED_PREF_assign = "assign"
            val KEY_assign = "key_assign"
            val sp3 = getSharedPreferences(SHARED_PREF_assign,
MODE_PRIVATE)
            val assign = sp3.getString(KEY_assign, null)
            val textAssign = findViewById<TextView>(R.id.inputAssign)
            if (assign !=null)
            {
                textAssign.text =  "$assign"
            }

            val items1 = start?.split("/".toRegex())?.toTypedArray()
            val day = items1?.get(0)
            val month = items1?.get(1)
            val year = items1?.get(2)

            val id = aircraft+"_"+check+"_"+day+month+year
            val db = FirebaseFirestore.getInstance()
            val df: DocumentReference = db.collection("Aircraft
revision").document(id)
            df.get().addOnSuccessListener { documentSnapshot ->

                if (documentSnapshot.getString("Status") == "-") {
                   val spinner2: Spinner = findViewById(R.id.spinner_status)
                   for (i in 0 until spinner2.adapter.count) {
                      if (spinner2.adapter.getItem(i).toString().contains(text.toString()))
{
                    spinner2.setSelection(0)
                      }
```

```
            }
        } else {
            val spinner2: Spinner = findViewById(R.id.spinner_status)
            val value = documentSnapshot.getString("Status")
            setSpinText(spinner2, value)
        }

        val textEnding: TextView = findViewById(R.id.inputEnd)
        val end = documentSnapshot.getString("End").toString()
        textEnding.text = end

        val textCrs: TextView = findViewById(R.id.inputCRS)
        val crs = documentSnapshot.getString("CRS").toString()
        textCrs.text = crs

      }
}

private fun setSpinText(spin: Spinner, text: String?) {
      for (i in 0 until spin.adapter.count) {
        if (spin.adapter.getItem(i).toString().contains(text.toString())) {
            spin.setSelection(i)
        }
      }
}

   private fun Spinner.avoidDropdownFocus() {
      try {
        val isAppCompat = this is
androidx.appcompat.widget.AppCompatSpinner
        val spinnerClass = if (isAppCompat)
androidx.appcompat.widget.AppCompatSpinner::class.java else
Spinner::class.java
        val popupWindowClass = if (isAppCompat)
androidx.appcompat.widget.ListPopupWindow::class.java else
android.widget.ListPopupWindow::class.java

        val listPopup = spinnerClass
              .getDeclaredField("mPopup")
              .apply { isAccessible = true }
              .get(this)
        if (popupWindowClass.isInstance(listPopup)) {
            val popup = popupWindowClass
                  .getDeclaredField("mPopup")
                  .apply { isAccessible = true }
                  .get(listPopup)
            if (popup is PopupWindow) {
                popup.isFocusable = false
            }
        }
        } catch (e: Exception) {
            e.printStackTrace()
        }
    }

    override fun onBackPressed() {
        super.onBackPressed()
        val intent = Intent(this, CheckListActivity::class.java)
        startActivity(intent)
    }

    private fun hideKeyboard(view: View) {
        val inputMethodManager =
getSystemService(INPUT_METHOD_SERVICE) as InputMethodManager
        inputMethodManager.hideSoftInputFromWindow(view.windowToken, 0)
    }

    private fun updateData() {

        val SHARED_PREF_AIRCRAFT = "aircraft";
        val KEY_AIRCRAFT = "key_aircraft";
        val sp = getSharedPreferences(SHARED_PREF_AIRCRAFT,
MODE_PRIVATE)
        val aircraft_key = sp.getString(KEY_AIRCRAFT, null)

        val SHARED_PREF_START = "start"
        val KEY_START = "key_start"
        val sp1 = getSharedPreferences(SHARED_PREF_START,
MODE_PRIVATE)
        val start_key = sp1.getString(KEY_START, null)

        val SHARED_PREF_check = "check"
        val KEY_check = "key_check"
        val sp2 = getSharedPreferences(SHARED_PREF_check,
MODE_PRIVATE)
        val check_key = sp2.getString(KEY_check, null)

        val items1 = start_key?.split("/".toRegex())?.toTypedArray()
        val day = items1?.get(0)
        val month = items1?.get(1)
        val year = items1?.get(2)


        val aircraft: EditText = findViewById(R.id.inputAircraft)
        val begin: EditText = findViewById(R.id.inputBegin)
        val end = findViewById<EditText>(R.id.inputEnd)
        val crs: EditText = findViewById(R.id.inputCRS)
        val assigned: EditText = findViewById(R.id.inputAssign)

        val spinner = findViewById<View>(R.id.spinner_check) as Spinner
        val spinner2 = findViewById<View>(R.id.spinner_status) as Spinner
        val check = spinner.selectedItem.toString()
        val status = spinner2.selectedItem.toString()
        val comment : EditText = findViewById(R.id.inputComment)

        val id = aircraft_key+"_"+check_key+"_"+day+month+year

        val time = "$month-$day-$year"
        val formatter: DateFormat = SimpleDateFormat("MM-dd-yyyy")
        val date_convert = formatter.parse(time) as Date
        val output = date_convert.time / 1000L
        val str = output.toString()
        val timestamp_begin = str.toLong() * 1000

        if (end.text.toString().isEmpty()){
            timestamp_end = "0"
        } else {
            val date_end = end.text.toString()
            val items2 = date_end.split("/".toRegex()).toTypedArray()
            val day2 = items2[0]
            val month2 = items2[1]
            val year2 = items2[2]
            val time2 = "$month2-$day2-$year2"
            val date_convert_end = formatter.parse(time2) as Date
            val output_end = date_convert_end.time / 1000L
            val str_end = output_end.toString()
            timestamp_end = ((str_end.toLong() * 1000).toString())
        }

        if (crs.text.toString().isEmpty()){
            timestamp_crs = "0"
        } else {
            val date_crs = crs.text.toString()
            val items3 = date_crs.split("/".toRegex()).toTypedArray()
            val day3 = items3[0]
            val month3 = items3[1]
            val year3 = items3[2]
            val time3 = "$month3-$day3-$year3"
            val date_convert_crs = formatter.parse(time3) as Date
            val output_crs = date_convert_crs.time / 1000L
            val str_crs = output_crs.toString()
            timestamp_crs = ((str_crs.toLong() * 1000).toString())
        }

        val timestamp_current = Calendar.getInstance().timeInMillis

        val db = FirebaseFirestore.getInstance()
        val user = FirebaseAuth.getInstance().currentUser
        val userEmail = user.email
        val df2= db.collection("Users").document(userEmail.toString())
```

```
        df2.get().addOnSuccessListener { documentSnapshot ->
            val added = documentSnapshot.getString("Name")
            if (aircraft.text.toString() != aircraft_key.toString()||
                    begin.text.toString() != start_key.toString()||
                    check != check_key) {
                db.collection("Aircraft revision").document(id).delete()
                val revisionInfo = hashMapOf("Aircraft" to aircraft.text.toString(),
                        "Aircraft" to aircraft.text.toString(),
                        "Check" to check,
                        "Start" to begin.text.toString(),
                        "End" to end.text.toString(),
                        "CRS" to crs.text.toString(),
                        "Status" to status,
                        "Assigned" to assigned.text.toString(),
                        "ID" to aircraft.text.toString() + "_" + check + "_" + day +
month + year,
                        "Comments" to comment.text.toString(),
                        "Text" to "Yes",
                        "Added" to added.toString(),
                        "timestamp_begin" to timestamp_begin.toString(),
                        "timestamp_end" to timestamp_end,
                        "timestamp_crs" to timestamp_crs,
                        "timestamp_current" to timestamp_current.toString())
                db.collection("Aircraft revision")
                        .document(aircraft.text.toString() + "_" + check + "_" + day +
month + year)
                        .set(revisionInfo)
                showSuccess()

            } else {
                val revisionInfo = hashMapOf(
                        "Aircraft" to aircraft.text.toString(),
                        "Check" to check,
                        "Start" to begin.text.toString(),
                        "End" to end.text.toString(),
                        "CRS" to crs.text.toString(),
                        "Status" to status,
                        "Assigned" to assigned.text.toString(),
                        "ID" to aircraft.text.toString() + "_" + check + "_" + day +
month + year,
                        "Comments" to comment.text.toString(),
                        "Text" to "Yes",
                        "Added" to added.toString(),
                        "timestamp_begin" to timestamp_begin.toString(),
                        "timestamp_end" to timestamp_end,
                        "timestamp_crs" to timestamp_crs,
                        "timestamp_current" to timestamp_current.toString())
                db.collection("Aircraft revision").document(id).set(revisionInfo)
                showSuccess()
            }
```

```
         }


      }

      private fun deletePreference() {

         val settings = getSharedPreferences(PREFS_NAME, 0)
         settings.edit().clear().apply()

         val SHARED_PREF_START = "start";
         val sp = getSharedPreferences(SHARED_PREF_START,
MODE_PRIVATE)
         sp.edit().clear().apply()

         val SHARED_PREF_END = "END";
         val sp1 = getSharedPreferences(SHARED_PREF_END,
MODE_PRIVATE)
         sp1.edit().clear().apply()

         val SHARED_PREF_CRS = "CRS";
         val sp2 = getSharedPreferences(SHARED_PREF_CRS,
MODE_PRIVATE)
         sp2.edit().clear().apply()

         val SHARED_PREF_check = "CHECK"
         val sp3 = getSharedPreferences(SHARED_PREF_check,
MODE_PRIVATE)
         sp3.edit().clear().apply()

         val SHARED_PREF_status = "STATUS"
         val sp4 = getSharedPreferences(SHARED_PREF_status,
MODE_PRIVATE)
         sp4.edit().clear().apply()

         val SHARED_PREF_COM = "COM"
         val sp5 = getSharedPreferences(SHARED_PREF_COM,
MODE_PRIVATE)
         sp5.edit().clear().apply()

      }

      private fun showSuccess() {
         val builder = AlertDialog.Builder(this)
         builder.setTitle("Information")
         builder.setMessage("Check edited successfully.")
         builder.setPositiveButton("Accept") { dialog, which ->
            val i = Intent(this, CheckListActivity::class.java)
            startActivity(i)
            finish()
```

```
        }
        val dialog: AlertDialog = builder.create()
        dialog.show()
      }

   }
```

A.13. Edit damage activity

```
class EditDamageActivity : AppCompatActivity() {

   private var mAdapter: AircraftAdapter? = null
   private val PICK_IMAGE = 100
   private var imageUri: Uri? = null
   private var uriFilePath: Uri? = null
   private val GALLERY = 1
   private  var CAMERA:Int = 2


   override fun onCreate(savedInstanceState: Bundle?) {

      super.onCreate(savedInstanceState)
      setContentView(R.layout.activity_add_damages)

      val title: TextView = findViewById(R.id.tool_title)
      title.text = "Edit damage activity"

      val button : ImageButton = findViewById(R.id.btn_menu)
      button.visibility = View.GONE

      val dimen = resources.getStringArray(R.array.dimen_array)
      val spinner: Spinner = findViewById(R.id.spinner_dimen)
      val adapter = ArrayAdapter(this, R.layout.spinner_item_selected2, dimen)
      adapter.setDropDownViewResource(R.layout.spinner_dropdown_item)
      spinner.adapter = adapter
      spinner.avoidDropdownFocus()
      spinner.onItemSelectedListener = object :
AdapterView.OnItemSelectedListener {
         override fun onItemSelected(
               parent: AdapterView<*>,
               view: View?,
               position: Int,
               id: Long
         ) {
            if (parent.getItemAtPosition(position) == "-") {
            } else {
               val SHARED_PREF_DIMEN = "DIMEN";
               val KEY_DIMEN = "key_DIMEN";
```

```
                val mSpnValue: String = spinner.selectedItem.toString()
                val myPrefs: SharedPreferences =
getSharedPreferences(SHARED_PREF_DIMEN, MODE_PRIVATE)
                val prefsEditor = myPrefs.edit()
                prefsEditor.putString(KEY_DIMEN, mSpnValue)
                prefsEditor.apply()
            }
        }

        override fun onNothingSelected(parent: AdapterView<*>?) {}
    }

    val type = resources.getStringArray(R.array.type_array)
    val spinner_type: Spinner = findViewById(R.id.spinner_Type)
    val adapter_type = ArrayAdapter(this, R.layout.spinner_item_selected,
type)

adapter_type.setDropDownViewResource(R.layout.spinner_dropdown_item)
    spinner_type.adapter = adapter_type
    spinner_type.avoidDropdownFocus()
    spinner_type.onItemSelectedListener = object :
AdapterView.OnItemSelectedListener {
        override fun onItemSelected(
                parent: AdapterView<*>,
                view: View?,
                position: Int,
                id: Long
        ) {
            if (parent.getItemAtPosition(position) == "-") {
            }
        }

        override fun onNothingSelected(parent: AdapterView<*>?) {}
    }

    val btn_back : ImageButton = findViewById(R.id.btn_back)
    btn_back.setOnClickListener {
        val intent = Intent(this, DamageActivity::class.java)
        startActivity(intent)
        finish()
    }

    val calendar = Calendar.getInstance()
    val edit = findViewById<View>(R.id.input_entry) as EditText
    val yy = calendar.get(Calendar.YEAR)
    val mm = calendar.get(Calendar.MONTH)
    val dd = calendar.get(Calendar.DAY_OF_MONTH)
    val datePicker = DatePickerDialog(
            this@EditDamageActivity,
            { _, year, monthOfYear, dayOfMonth ->
```

```
                val date =
                        (dayOfMonth.toString() + "/" + (monthOfYear + 1).toString() +
"/" + year.toString())
                edit.setText(date)
            },
            yy,
            mm,
            dd
        )
        datePicker.setButton(DialogInterface.BUTTON_NEUTRAL, "CLEAR") { _,
which ->
            if (which == DialogInterface.BUTTON_NEUTRAL) {
                edit.setText("-")
            }
        }

        edit.setOnClickListener {
            datePicker.show()
        }

        val edit2 = findViewById<View>(R.id.input_closing) as EditText
        val calendar2 = Calendar.getInstance()
        val yy2 = calendar2.get(Calendar.YEAR)
        val mm2 = calendar2.get(Calendar.MONTH)
        val dd2 = calendar2.get(Calendar.DAY_OF_MONTH)
        val datePicker2 = DatePickerDialog(
            this@EditDamageActivity,
            { _, year, monthOfYear, dayOfMonth ->
                val date2 =
                        (dayOfMonth.toString() + "/" + (monthOfYear + 1).toString() +
"/" + year.toString())
                edit2.setText(date2)
            },
            yy2,
            mm2,
            dd2
        )
        datePicker2.setButton(DialogInterface.BUTTON_NEUTRAL, "CLEAR") {
_, which ->
            if (which == DialogInterface.BUTTON_NEUTRAL) {
                edit2.setText("-")
            }
        }

        edit2.setOnClickListener() {
            datePicker2.show()
        }

        val fh: EditText = findViewById(R.id.input_FH)
        val fc: EditText = findViewById(R.id.input_FC)
```

```
    val days: EditText = findViewById(R.id.input_Days)
    val fh2: EditText = findViewById(R.id.input_FH_repair)
    val fc2: EditText = findViewById(R.id.input_FC_repair)
    val days2: EditText = findViewById(R.id.input_Days_repair)

    val cat = resources.getStringArray(R.array.cat_array)
    val spinner1: Spinner = findViewById(R.id.spinner_cat)
    val adapter1 = ArrayAdapter(this, R.layout.spinner_item_selected, cat)
    adapter1.setDropDownViewResource(R.layout.spinner_dropdown_item)
    spinner1.adapter = adapter1
    spinner1.avoidDropdownFocus()
    spinner1.onItemSelectedListener = object :
AdapterView.OnItemSelectedListener {
        override fun onItemSelected(
            parent: AdapterView<*>,
            view: View?,
            position: Int,
            id: Long
        ) {
          if (parent.getItemAtPosition(position).toString() == "B"
              || parent.getItemAtPosition(position).toString() == "C") {
            edit2.setBackgroundResource(R.color.black_overlay)
            edit2.setOnClickListener {
              datePicker2.hide()
            }
            if (parent.getItemAtPosition(position).toString() == "B") {

              fh.isEnabled = true
              fh.setBackgroundResource(R.drawable.custom_input)
              fc.isEnabled = true
              fc.setBackgroundResource(R.drawable.custom_input)
              days.isEnabled = true
              days.setBackgroundResource(R.drawable.custom_input)

              fh2.isEnabled = false
              fh2.setBackgroundResource(R.color.black_overlay)
              fc2.isEnabled = false
              fc2.setBackgroundResource(R.color.black_overlay)
              days2.isEnabled = false
              days2.setBackgroundResource(R.color.black_overlay)
            }

            if (parent.getItemAtPosition(position).toString() == "C") {

              fh.isEnabled = true
              fh.setBackgroundResource(R.drawable.custom_input)
              fc.isEnabled = true
              fc.setBackgroundResource(R.drawable.custom_input)
              days.isEnabled = true
              days.setBackgroundResource(R.drawable.custom_input)
```

```
                fh2.isEnabled = true
                fh2.setBackgroundResource(R.drawable.custom_input)
                fc2.isEnabled = true
                fc2.setBackgroundResource(R.drawable.custom_input)
                days2.isEnabled = true
                days2.setBackgroundResource(R.drawable.custom_input)
            }

        }

        if (parent.getItemAtPosition(position).toString() == "A") {
            edit2.setBackgroundResource(R.drawable.custom_input)
            edit2.setOnClickListener {
                datePicker2.show()
            }

            fh.isEnabled = false
            fh.setBackgroundResource(R.color.black_overlay)
            fc.isEnabled = false
            fc.setBackgroundResource(R.color.black_overlay)
            days.isEnabled = false
            days.setBackgroundResource(R.color.black_overlay)

            fh2.isEnabled = false
            fh2.setBackgroundResource(R.color.black_overlay)
            fc2.isEnabled = false
            fc2.setBackgroundResource(R.color.black_overlay)
            days2.isEnabled = false
            days2.setBackgroundResource(R.color.black_overlay)


        }

    }

    override fun onNothingSelected(parent: AdapterView<*>?) {}
}

changeButton()

displayInfo()

confirm()

val upload = findViewById<View>(R.id.button_upload) as Button
upload.setOnClickListener {
    openGallery()
}
```

```
        val imageView: ImageView = findViewById(R.id.imageView2)
        val button2: Button = findViewById(R.id.button_clear)
        button2.setOnClickListener {
            imageView.setImageDrawable(null)
        }

    }

    private fun openGallery() {
        val gallery = Intent(Intent.ACTION_PICK,
MediaStore.Images.Media.INTERNAL_CONTENT_URI)
        startActivityForResult(gallery, PICK_IMAGE)
    }

    override fun onActivityResult(requestCode: Int, resultCode: Int, data:
Intent?) {
        val imageView = findViewById<ImageView>(R.id.imageView2)
        super.onActivityResult(requestCode, resultCode, data)
        if (resultCode == RESULT_OK && requestCode == PICK_IMAGE) {
            imageUri = data?.data
            imageView!!.setImageURI(imageUri)
        }
        else if (requestCode == CAMERA) {
            imageUri = data?.data
            imageView!!.setImageURI(imageUri)
        }
    }

    private fun changeButton() {
        val btn_add : Button = findViewById(R.id.btn_Add)
        val user = FirebaseAuth.getInstance().currentUser
        val userEmail = user.email
        val db = FirebaseFirestore.getInstance()
        val df3 = db.collection("Users").document(userEmail)
        df3.get().addOnSuccessListener { documentSnapshot ->
            if (documentSnapshot.getString("Job") == "MRO engineer") {
                btn_add.setText("Certify")
            }
            if (documentSnapshot.getString("Job") == "CAMO engineer") {
                btn_add.setText("Check")
            }
            if (documentSnapshot.getString("Job") == "Airplane maintenance
technician") {
                btn_add.setText("Edit")
            }
            if (documentSnapshot.getString("Job") == "Total access engineer"
                    || documentSnapshot.getString("Job") == "Administrator") {
                btn_add.setText("Edit")
            }
        }
    }

    private fun confirm() {
        val confirm: Button = findViewById(R.id.btn_Add)
        confirm.setOnClickListener {
            updateData()
            showSuccess()
        }
    }

    private fun showSuccess() {
        val builder = AlertDialog.Builder(this)
        builder.setTitle("Information")
        builder.setMessage("Damage report edited successfully.")
        builder.setPositiveButton("Accept") { dialog, which ->
            val intent = Intent(this, DamageActivity::class.java)
            startActivity(intent)
            finish()
        }
        val dialog: AlertDialog = builder.create()
        dialog.show()
    }

    private fun hasNullOrEmptyDrawable(iv: ImageView?): Boolean {
        val drawable = iv!!.drawable
        val bitmapDrawable = if (drawable is BitmapDrawable) drawable else null

        return bitmapDrawable == null || bitmapDrawable.bitmap == null
    }

    private fun updateData() {

        val SHARED_PREF_DAMAGE = "DAMAGE"
        val KEY_DAMAGE= "key_DAMAGE"
        val sp0= getSharedPreferences(SHARED_PREF_DAMAGE,
MODE_PRIVATE)
        val damage = sp0.getString(KEY_DAMAGE, null)

        val SHARED_PREF_AIRCRAFT = "aircraft";
        val KEY_AIRCRAFT = "key_aircraft";
        val sp = getSharedPreferences(SHARED_PREF_AIRCRAFT,
MODE_PRIVATE)
        val aircraft = sp.getString(KEY_AIRCRAFT, null)

        val SHARED_PREF_START = "start"
        val KEY_START = "key_start"
        val sp1 = getSharedPreferences(SHARED_PREF_START,
MODE_PRIVATE)
        val start = sp1.getString(KEY_START, null)
        val SHARED_PREF_check = "check"
        val KEY_check = "key_check"
        val sp2 = getSharedPreferences(SHARED_PREF_check,
MODE_PRIVATE)
        val check = sp2.getString(KEY_check, null)

        val items1 = start?.split("/".toRegex())?.toTypedArray()
        val day = items1?.get(0)
        val month = items1?.get(1)
        val year = items1?.get(2)

        val id = aircraft+"_"+check+"_"+day+month+year
        val db = FirebaseFirestore.getInstance()
        val df: DocumentReference = db.collection("Aircraft
revision").document(id)
                .collection("New damages").document(damage.toString())

        val fileRef =
FirebaseStorage.getInstance().reference.child("damage/$damage")
        val imageView = findViewById<ImageView>(R.id.imageView2)

        if (hasNullOrEmptyDrawable(imageView)) {

        } else {
            val bitmap = (imageView.drawable as BitmapDrawable).bitmap
            imageView.isDrawingCacheEnabled = true
            imageView.buildDrawingCache()
            val baos = ByteArrayOutputStream()
            bitmap.compress(Bitmap.CompressFormat.JPEG, 100, baos)
            val data = baos.toByteArray()
            fileRef.putBytes(data)
        }

        df.get().addOnSuccessListener { documentSnapshot ->

            val map = documentSnapshot.getString("Map")
            val certify = documentSnapshot.getString("Certified")
            val checked = documentSnapshot.getString("Checked")
            val certifyBy = documentSnapshot.getString("CertifyBY")
            val checkedBy = documentSnapshot.getString("CheckedBY")
            val rb = documentSnapshot.getString("RB")

            val spinner = findViewById<View>(R.id.spinner_cat) as Spinner
            val cat = spinner.selectedItem.toString()
            val spinner2: Spinner = findViewById(R.id.spinner_dimen)
            val dimen = spinner2.selectedItem.toString()
            val spinner3 = findViewById<View>(R.id.spinner_Type) as Spinner
            val type = spinner3.selectedItem.toString()
            val ata = findViewById<EditText>(R.id.inputATA)
            val location = findViewById<TextView>(R.id.inputLocation)
            val dimen1: TextView = findViewById(R.id.input_dimen_1)
            val dimen2: TextView = findViewById(R.id.input_dimen_2)
            val dimen3: TextView = findViewById(R.id.input_dimen_3)
            val fh: EditText = findViewById(R.id.input_FH)
            val fc: EditText = findViewById(R.id.input_FC)
            val days: EditText = findViewById(R.id.input_Days)
            val fh2: EditText = findViewById(R.id.input_FH_repair)
            val fc2: EditText = findViewById(R.id.input_FC_repair)
            val days2: EditText = findViewById(R.id.input_Days_repair)
            val entry: EditText = findViewById(R.id.input_entry)
            val closing: EditText = findViewById(R.id.input_closing)
            val pn: EditText = findViewById(R.id.input_PN)
            val sn: EditText = findViewById(R.id.input_SN)
            val ref: EditText = findViewById(R.id.input_ref)
            val comment: EditText = findViewById(R.id.inputComment)
            val regis : EditText = findViewById(R.id.inputAircraft)

            if (comment.text.toString().isEmpty()) {

                val revisionInfo = hashMapOf(
                      "Aircraft" to regis.text.toString(),
                      "Inspection FH" to fh.text.toString(),
                      "Inspection FC" to fc.text.toString(),
                      "Inspection days" to days.text.toString(),
                      "Repair FH" to fh2.text.toString(),
                      "Repair FC" to fc2.text.toString(),
                      "Repair days" to days2.text.toString(),
                      "Type" to type,
                      "Dim" to dimen,
                      "Dim1" to dimen1.text.toString(),
                      "Dim2" to dimen2.text.toString(),
                      "Dim3" to dimen3.text.toString(),
                      "References" to ref.text.toString(),
                      "Entry" to entry.text.toString(),
                      "Closing" to closing.text.toString(),
                      "Description" to type+"_"+location.text.toString(),
                      "Comments" to comment.text.toString(),
                      "Commentary" to "No",
                      "PN" to pn.text.toString(),
                      "SN" to sn.text.toString(),
                      "Category" to cat,
                      "ATA" to ata.text.toString(),
                      "Location" to location.text.toString(),
                      "Dimension" to dimen1.text.toString() + "x"
                            + dimen2.text.toString() + "x"
                            + dimen3.text.toString() + dimen,
                      "ID" to damage.toString(),
                      "RB" to rb.toString(),
                      "Certified" to certify.toString(),
                      "Checked" to checked.toString(),
                      "Map" to map.toString(),
                      "CertifyBY" to certifyBy.toString(),
                      "CheckedBY" to checkedBy.toString())

                db.collection("Aircraft revision").document(id)
                      .collection("New
      damages").document(damage.toString()).set(revisionInfo)

            }

            if (comment.text.toString().isNotEmpty()) {

                val revisionInfo = hashMapOf(
                      "Aircraft" to regis.text.toString(),
                      "Inspection FH" to fh.text.toString(),
                      "Inspection FC" to fc.text.toString(),
                      "Inspection days" to days.text.toString(),
                      "Repair FH" to fh2.text.toString(),
                      "Repair FC" to fc2.text.toString(),
                      "Repair days" to days2.text.toString(),
                      "Type" to type,
                      "Dim" to dimen,
                      "Dim1" to dimen1.text.toString(),
                      "Dim2" to dimen2.text.toString(),
                      "Dim3" to dimen3.text.toString(),
                      "References" to ref.text.toString(),
                      "Entry" to entry.text.toString(),
                      "Closing" to closing.text.toString(),
                      "Description" to type+"_"+location.text.toString(),
                      "Comments" to comment.text.toString(),
                      "Commentary" to "Yes",
                      "PN" to pn.text.toString(),
                      "SN" to sn.text.toString(),
                      "Category" to cat,
                      "ATA" to ata.text.toString(),
                      "Location" to location.text.toString(),
                      "Dimension" to dimen1.text.toString() + "x"
                            + dimen2.text.toString() + "x"
                            + dimen3.text.toString() + dimen,
                      "ID" to damage,
                      "RB" to "No",
                      "Certified" to certify.toString(),
                      "Checked" to checked.toString(),
                      "Map" to map.toString(),
                      "CertifyBY" to certifyBy.toString(),
                      "CheckedBY" to checkedBy.toString())

                db.collection("Aircraft revision").document(id)
                      .collection("New
      damages").document(damage.toString()).set(revisionInfo)
```

```
            }
        }
    }

    private fun Spinner.avoidDropdownFocus() {
        try {
            val isAppCompat = this is
androidx.appcompat.widget.AppCompatSpinner
            val spinnerClass = if (isAppCompat)
androidx.appcompat.widget.AppCompatSpinner::class.java else
Spinner::class.java
            val popupWindowClass = if (isAppCompat)
androidx.appcompat.widget.ListPopupWindow::class.java else
android.widget.ListPopupWindow::class.java

            val listPopup = spinnerClass
                    .getDeclaredField("mPopup")
                    .apply { isAccessible = true }
                    .get(this)
            if (popupWindowClass.isInstance(listPopup)) {
                val popup = popupWindowClass
                        .getDeclaredField("mPopup")
                        .apply { isAccessible = true }
                        .get(listPopup)
                if (popup is PopupWindow) {
                    popup.isFocusable = false
                }
            }
        } catch (e: Exception) {
            e.printStackTrace()
        }
    }

    private fun displayInfo()  {

        val SHARED_PREF_DAMAGE = "DAMAGE"
        val KEY_DAMAGE= "key_DAMAGE"
        val sp0= getSharedPreferences(SHARED_PREF_DAMAGE,
MODE_PRIVATE)
        val damage = sp0.getString(KEY_DAMAGE, null)

        val SHARED_PREF_AIRCRAFT = "aircraft";
        val KEY_AIRCRAFT = "key_aircraft";
        val sp = getSharedPreferences(SHARED_PREF_AIRCRAFT,
MODE_PRIVATE)
        val aircraft = sp.getString(KEY_AIRCRAFT, null)

        val SHARED_PREF_START = "start"
        val KEY_START = "key_start"
        val sp1 = getSharedPreferences(SHARED_PREF_START,
MODE_PRIVATE)
        val start = sp1.getString(KEY_START, null)

        val SHARED_PREF_check = "check"
        val KEY_check = "key_check"
        val sp2 = getSharedPreferences(SHARED_PREF_check,
MODE_PRIVATE)
        val check = sp2.getString(KEY_check, null)

        val items1 = start?.split("/".toRegex())?.toTypedArray()
        val day = items1?.get(0)
        val month = items1?.get(1)
        val year = items1?.get(2)

        val id = aircraft+"_"+check+"_"+day+month+year
        val db = FirebaseFirestore.getInstance()
        val df: DocumentReference = db.collection("Aircraft
revision").document(id)
                .collection("New damages").document(damage.toString())

        val ref =
FirebaseStorage.getInstance().reference.child("damage/$damage")
        val imageView : ImageView = this.findViewById(R.id.imageView2)
        imageView.drawable
        ref.downloadUrl.addOnSuccessListener {Uri->
            val imageURL = Uri.toString()
            Glide.with(this)
                    .load(imageURL)
                    .into(imageView)
        }

        df.get().addOnSuccessListener { documentSnapshot ->

            val txtplane = findViewById<TextView>(R.id.inputAircraft)
            val plane = documentSnapshot.getString("Aircraft").toString()
            txtplane.text = plane
            txtplane.isFocusable=false
            txtplane.setBackgroundResource(R.drawable.custom_input)

            val spinner = findViewById<View>(R.id.spinner_cat) as Spinner
            if (documentSnapshot.getString("Category") == "-") {
                for (i in 0 until spinner.adapter.count) {
                    if (spinner.adapter.getItem(i).toString().contains(aircraft.toString()))
{
                        spinner.setSelection(0)
                    }
                }
            } else {
                val value = documentSnapshot.getString("Category")
                setSpinText(spinner, value)
            }

            val spinner3 = findViewById<View>(R.id.spinner_Type) as Spinner
            if (documentSnapshot.getString("Type") == "-") {
                for (i in 0 until spinner3.adapter.count) {
                    if
(spinner3.adapter.getItem(i).toString().contains(aircraft.toString())) {
                        spinner3.setSelection(0)
                    }
                }
            } else {
                val value = documentSnapshot.getString("Type")
                setSpinText(spinner3, value)
            }

            val spinner2: Spinner = findViewById(R.id.spinner_dimen)
            if (documentSnapshot.getString("Dim") == "-") {
                for (i in 0 until spinner2.adapter.count) {
                    if
(spinner2.adapter.getItem(i).toString().contains(aircraft.toString())) {
                        spinner2.setSelection(0)
                    }
                }
            } else {
                val value = documentSnapshot.getString("Dim")
                setSpinText(spinner2, value)
            }

            val txtata = findViewById<TextView>(R.id.inputATA)
            val ata = documentSnapshot.getString("ATA").toString()
            txtata.text = ata

            val txtlocation = findViewById<TextView>(R.id.inputLocation)
            val location = documentSnapshot.getString("Location").toString()
            txtlocation.text = location

            val txtdimen1: TextView = findViewById(R.id.input_dimen_1)
            val dimen1 = documentSnapshot.getString("Dim1").toString()
            txtdimen1.text = dimen1

            val txtdimen2: TextView = findViewById(R.id.input_dimen_2)
            val dimen2 = documentSnapshot.getString("Dim2").toString()
            txtdimen2.text = dimen2


            val txtdimen3: TextView = findViewById(R.id.input_dimen_3)
            val dimen3 = documentSnapshot.getString("Dim3").toString()
            txtdimen3.text = dimen3
            val txtfh: TextView = findViewById(R.id.input_FH)
            val fh = documentSnapshot.getString("Inspection FH").toString()
            txtfh.text = fh

            val txtfc: TextView = findViewById(R.id.input_FC)
            val fc = documentSnapshot.getString("Inspection FC").toString()
            txtfc.text = fc

            val txtdays: TextView= findViewById(R.id.input_Days)
            val days = documentSnapshot.getString("Inspection days").toString()
            txtdays.text = days

            val txtfh2: TextView = findViewById(R.id.input_FH_repair)
            val fh2 = documentSnapshot.getString("Repair FH").toString()
            txtfh2.text = fh2

            val txtfc2: TextView = findViewById(R.id.input_FC_repair)
            val fc2 = documentSnapshot.getString("Repair FC").toString()
            txtfc2.text = fc2

            val txtdays2: TextView= findViewById(R.id.input_Days_repair)
            val days2 = documentSnapshot.getString("Repair days").toString()
            txtdays2.text = days2

            val txtentry: TextView = findViewById(R.id.input_entry)
            val entry = documentSnapshot.getString("Entry").toString()
            txtentry.text = entry

            val txtclosing: TextView = findViewById(R.id.input_closing)
            val closing = documentSnapshot.getString("Closing").toString()
            txtclosing.text = closing

            val txtpn: TextView = findViewById(R.id.input_PN)
            val pn = documentSnapshot.getString("PN").toString()
            txtpn.text = pn

            val txtsn: TextView = findViewById(R.id.input_SN)
            val sn = documentSnapshot.getString("SN").toString()
            txtsn.text = sn

            val txtref: TextView = findViewById(R.id.input_ref)
            val ref = documentSnapshot.getString("References").toString()
            txtref.text = ref

            val txtcomment: TextView= findViewById(R.id.inputComment)
            val comment = documentSnapshot.getString("Comments").toString()
            txtcomment.text = comment

        }
    }

    private fun setSpinText(spin: Spinner, text: String?) {
        for (i in 0 until spin.adapter.count) {
            if (spin.adapter.getItem(i).toString().contains(text.toString())) {
                spin.setSelection(i)
            }
        }
    }

}
```

A.14. Edit user activity

```
class EditUserActivity : AppCompatActivity() {

    var button: ImageButton? = null
    private val PICK_IMAGE = 100
    var imageUri: Uri? = null

    override fun onCreate(savedInstanceState: Bundle?) {

        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_edit_user)

        val title : TextView = findViewById(R.id.tool_title)
        title.text = "Edit user info"

        val btn_menu : ImageButton = findViewById(R.id.btn_menu)
        btn_menu.visibility = View.GONE

        val name: EditText = findViewById(R.id.inputNameRegister)
        val phone: EditText = findViewById(R.id.inputPhone)
        val email: EditText = findViewById(R.id.inputEmailRegister)
        val pin: EditText = findViewById(R.id.inputPIN)

        name.onFocusChangeListener = OnFocusChangeListener { b, hasFocus -
>
            if (!hasFocus) {
                name.setHint(R.string.NameHint)
                hideKeyboard(b)
            } else name.hint = ""
        }
        phone.onFocusChangeListener = OnFocusChangeListener { b, hasFocus
->
            if (!hasFocus) {
                phone.setHint(R.string.PhoneHint)
                hideKeyboard(b)
            } else phone.hint = ""
        }
        email.onFocusChangeListener = OnFocusChangeListener { a, hasFocus -
>
            if (!hasFocus) {
                email.setHint(R.string.EmailHint)
                hideKeyboard(a)
            } else email.hint = ""
        }
        pin.onFocusChangeListener = OnFocusChangeListener { b, hasFocus ->
            if (!hasFocus) {
                pin.setHint(R.string.PINHint)
                hideKeyboard(b)
            } else pin.hint = ""
        }

        val SHARED_PREF_USER = "user";
        val KEY_USER = "key_user";
        val sp = getSharedPreferences(SHARED_PREF_USER,
MODE_PRIVATE)
        val mail = sp.getString(KEY_USER, null)
        val edit = findViewById<TextView>(R.id.inputEmailRegister)
        if (mail != null) {
            edit.text = "$mail"
        }

        val db = FirebaseFirestore.getInstance()
        val df: DocumentReference =
db.collection("Users").document(mail.toString())
        df.get().addOnSuccessListener { documentSnapshot ->

            val value = documentSnapshot.getString("Permission").toString()
            val perm = resources.getStringArray(R.array.permission_array)
            val spinner2: Spinner = findViewById(R.id.spinner_permission)
            val adapter2 = ArrayAdapter(this, R.layout.spinner_item_selected,
perm)
            adapter2.setDropDownViewResource(R.layout.spinner_dropdown_item)
            spinner2.adapter = adapter2
            spinner2.avoidDropdownFocus()
            spinner2.onItemSelectedListener = object : OnItemSelectedListener {
                override fun onItemSelected(parent: AdapterView<*>, view: View?,
position: Int, id: Long) {
                    if (parent.getItemAtPosition(position) == value) {
                    }
                }
                override fun onNothingSelected(parent: AdapterView<*>?) {}
            }

            val job = documentSnapshot.getString("Job").toString()
            val roles = resources.getStringArray(R.array.job_array)
            val spinner: Spinner = findViewById(R.id.spinner_edit_user)
            val adapter = ArrayAdapter(this, R.layout.spinner_item_selected, roles)
            adapter.setDropDownViewResource(R.layout.spinner_dropdown_item)
            spinner.adapter = adapter
            spinner.avoidDropdownFocus()
            spinner.onItemSelectedListener = object : OnItemSelectedListener {
                override fun onItemSelected(parent: AdapterView<*>, view: View?,
position: Int, id: Long) {
                    if (parent.getItemAtPosition(position) == "Please choose a job
role") {
                    }
                }
                override fun onNothingSelected(parent: AdapterView<*>?) {}
            }


        }

        val back : ImageButton = findViewById(R.id.btn_back)
        back.setOnClickListener {
            val intent = Intent(this, UserActivity::class.java)
            startActivity(intent)
            finish()
        }

        displayInfo()

        val btn_add : Button = findViewById(R.id.btn_edit)
        btn_add.setOnClickListener {
            updateData()
        }

        val card : CardView = findViewById(R.id.card_profile)
        val selectPicture = card.findViewById<ImageView>(R.id.profilePicture)
        selectPicture.setOnClickListener { openGallery()}

        val remove : ImageButton = findViewById(R.id.removePicture)
        remove.setOnClickListener {
            val imageView = findViewById<ImageView>(R.id.profilePicture)
            imageView.setImageResource(0)
            imageView.drawable
        }

    }

    private fun openGallery() {
        val gallery = Intent(Intent.ACTION_PICK,
MediaStore.Images.Media.INTERNAL_CONTENT_URI)
        startActivityForResult(gallery, PICK_IMAGE)
    }
```

```
   override fun onActivityResult(requestCode: Int, resultCode: Int, data:
Intent?) {
      val imageView = findViewById<ImageView>(R.id.profilePicture)
      super.onActivityResult(requestCode, resultCode, data)
      if (resultCode == RESULT_OK && requestCode == PICK_IMAGE) {
         imageUri = data?.data
         imageView!!.setImageURI(imageUri)
      }
   }

   private fun hideKeyboard(view: View) {
      val inputMethodManager =
getSystemService(INPUT_METHOD_SERVICE) as InputMethodManager
      inputMethodManager.hideSoftInputFromWindow(view.windowToken, 0)
   }

   private fun Spinner.avoidDropdownFocus() {
      try {
         val isAppCompat = this is
androidx.appcompat.widget.AppCompatSpinner
         val spinnerClass = if (isAppCompat)
androidx.appcompat.widget.AppCompatSpinner::class.java else
Spinner::class.java
         val popupWindowClass = if (isAppCompat)
androidx.appcompat.widget.ListPopupWindow::class.java else
android.widget.ListPopupWindow::class.java

         val listPopup = spinnerClass
               .getDeclaredField("mPopup")
               .apply { isAccessible = true }
               .get(this)
         if (popupWindowClass.isInstance(listPopup)) {
            val popup = popupWindowClass
                  .getDeclaredField("mPopup")
                  .apply { isAccessible = true }
                  .get(listPopup)
            if (popup is PopupWindow) {
               popup.isFocusable = false
            }
         }
      } catch (e: Exception) {
         e.printStackTrace()
      }
   }

   private fun checkJob(): Boolean {
      val spinner: Spinner = findViewById(R.id.spinner_edit_user)
      if (spinner.selectedItem.toString() == "Please choose a job role")
      {
         val builder = AlertDialog.Builder(this)
```

```
            builder.setTitle("Error")
            builder.setMessage("Necessary to select a job role")
            builder.setPositiveButton("Accept", null)
            val dialog: AlertDialog = builder.create()
            dialog.show()
            return false
        } else {
            return true
        }
    }

    private fun checkName(): Boolean {
        val box: TextInputLayout = findViewById(R.id.inputNameBox)
        val name: EditText = findViewById(R.id.inputNameRegister)
        if (name.text.toString() == "") {
            box.error = "Please enter full name"
            return false
        } else if  (name.length() >= 50) {
            box.error = "Full name too long"
            return false
        } else {
            box.error = null
            return true
        }
    }

    private fun checkPIN(): Boolean {
        val box: TextInputLayout = findViewById(R.id.boxPIN)
        val pin: EditText = findViewById(R.id.inputPIN)
        if (pin.text.toString() == "") {
            box.error = "Please enter a custom PIN"
            return false
        } else if  (pin.length() > 4) {
            box.error = "PIN not valid"
            return false
        } else {
            box.error = null
            return true
        }
    }

    private fun checkPhone(): Boolean {
        val box: TextInputLayout = findViewById(R.id.boxPhone)
        val phone: EditText = findViewById(R.id.inputPhone)
        if (phone.text.isEmpty()) {
            box.error = "Please enter your mobile phone number"
            return false
        } else if (phone.length() >= 10) {
            box.error = "Mobile phone number too long"
            return false
        } else {
            box.error = null
            return true
        }
    }

    private fun checkEmail(): Boolean {
        val email: EditText = findViewById(R.id.inputEmailRegister)
        val box: TextInputLayout = findViewById(R.id.boxEmail)
        if (email.text.isEmpty()) {
            box.error = "Please enter email address"
            return false
        } else {
            box.error = null
            return true
        }
    }

    private fun setSpinText(spin: Spinner, text: String?) {
        for (i in 0 until spin.adapter.count) {
            if (spin.adapter.getItem(i).toString().contains(text.toString())) {
                spin.setSelection(i)
            }
        }
    }


    private fun displayInfo() {

        val SHARED_PREF_USER = "user";
        val KEY_USER = "key_user";
        val sp = getSharedPreferences(SHARED_PREF_USER,
MODE_PRIVATE)
        val mail = sp.getString(KEY_USER, null)
        val edit = findViewById<TextView>(R.id.inputEmailRegister)
        if (mail != null) {
            edit.text = "$mail"
        }

        val db = FirebaseFirestore.getInstance()
        val df: DocumentReference =
db.collection("Users").document(mail.toString())
        df.get().addOnSuccessListener { documentSnapshot ->

            val spinner: Spinner = findViewById(R.id.spinner_edit_user)
            val value = documentSnapshot.getString("Job")
            setSpinText(spinner, value)


            val txtpin : TextView = findViewById(R.id.inputPIN)
            val pin = documentSnapshot.getString("PIN")
            txtpin.text = pin

            val textname: TextView = findViewById(R.id.inputNameRegister)
            val name = documentSnapshot.getString("Name")
            textname.text = name

            val textphone: TextView = findViewById(R.id.inputPhone)
            val phone = documentSnapshot.getString("Phone").toString()
            textphone.text = phone


            val spinner2: Spinner = findViewById(R.id.spinner_permission)
            val value2 = documentSnapshot.getString("Permission").toString()
            setSpinText(spinner2, value2)

        }

        val ref = FirebaseStorage.getInstance().reference.child("profile/$mail")
        val profile : CardView = this.findViewById(R.id.card_profile)
        val imageView = profile.findViewById<ImageView>(R.id.profilePicture)
        imageView.drawable
        ref.downloadUrl.addOnSuccessListener { Uri ->
            imageView.setBackgroundColor(Color.TRANSPARENT);
            val imageURL = Uri.toString()
            Glide.with(this)
                    .load(imageURL)
                    .into(imageView)
        }.addOnFailureListener {}

    }

    private fun updateData() {

        val fullname: EditText = findViewById(R.id.inputNameRegister)
        val phone: EditText = findViewById(R.id.inputPhone)
        val email: EditText = findViewById(R.id.inputEmailRegister)
        val job : Spinner = findViewById(R.id.spinner_edit_user)
        val item = job.selectedItem.toString()
        val permission : Spinner = findViewById(R.id.spinner_permission)
       val item2 = permission.selectedItem.toString()
       val pin : EditText = findViewById(R.id.inputPIN)
        val db = FirebaseFirestore.getInstance()

        if (email.text.toString().isEmpty() || fullname.text.toString().isEmpty()
                || phone.text.toString().isEmpty() || item == "Please choose a job
role" || pin.text.toString().isEmpty()) {
            showFail()
        } else if (checkName() and checkPhone() and checkEmail()
                and checkJob() and checkPIN()) {
```

```
            db.collection("Users").document(email.text.toString()).delete()
            val revisionInfo = hashMapOf(
                "Name" to fullname.text.toString(),
                "Phone" to phone.text.toString(),
                "Email" to email.text.toString(),
                "Job" to item,
                "Permission" to item2,
                "PIN" to pin.text.toString())
            db.collection("Users").document(email.text.toString()).set(revisionInfo)
            showSuccess()
         }
        val fileRef = FirebaseStorage.getInstance().reference.child("profile/" +
email.text.toString())
        val imageView = findViewById<ImageView>(R.id.profilePicture)
        if (hasNullOrEmptyDrawable(imageView))
        {

        } else {
            val bitmap = (imageView.drawable as BitmapDrawable).bitmap
            imageView.isDrawingCacheEnabled = true
            imageView.buildDrawingCache()
            val baos = ByteArrayOutputStream()
            bitmap.compress(Bitmap.CompressFormat.JPEG, 100, baos)
            val data = baos.toByteArray()
            fileRef.putBytes(data)
        }

    }

    private fun hasNullOrEmptyDrawable(iv: ImageView?): Boolean {
        val drawable = iv!!.drawable
        val bitmapDrawable = if (drawable is BitmapDrawable) drawable else null

        return bitmapDrawable == null || bitmapDrawable.bitmap == null
    }

    private fun showSuccess() {
        val builder = AlertDialog.Builder(this)
        builder.setTitle("Information")
        builder.setMessage("User edited successfully.")
        builder.setPositiveButton("Accept") { dialog, which ->
            val intent = Intent(this, UserActivity::class.java)
            startActivity(intent)
            finish()
        }
        val dialog: AlertDialog = builder.create()
        dialog.show()
    }

    private fun showFail() {
        val builder = AlertDialog.Builder(this)
        builder.setTitle("Error")
        builder.setMessage("All fields must be filled.")
        builder.setPositiveButton("Accept") { dialog, which ->}
        val dialog: AlertDialog = builder.create()
        dialog.show()
    }

}
```

A.15. Edit user information activity

```
class EditUserInformation : AppCompatActivity() {

    var button: ImageButton? = null
    private val PICK_IMAGE = 100
    var imageUri: Uri? = null

    override fun onCreate(savedInstanceState: Bundle?) {

        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_edit_info)

        val title : TextView = findViewById(R.id.tool_title)
        title.text = "Edit information"

        val user = Firebase.auth.currentUser
        val email= user.email
        val ref = FirebaseStorage.getInstance().reference.child("profile/$email")
        val profile : CardView = this.findViewById(R.id.card_profile)
        val imageView = profile.findViewById<ImageView>(R.id.profilePicture)
        imageView.drawable
        ref.downloadUrl.addOnSuccessListener { Uri ->
            imageView.setBackgroundColor(Color.TRANSPARENT);
            val imageURL = Uri.toString()
            Glide.with(this)
                    .load(imageURL)
                    .into(imageView)
        }.addOnFailureListener {}

        val back = findViewById<ImageButton>(R.id.btn_back)

        back.setOnClickListener() {
            openActivity1()
        }

        val name: TextInputEditText = findViewById(R.id.inputNameRegister)
        val phone: EditText = findViewById(R.id.inputPhone)
        val signature:EditText=findViewById(R.id.inputPIN)

        val box: TextInputLayout = findViewById(R.id.inputNameBox)
        val box2: TextInputLayout = findViewById(R.id.boxPIN)
        val box3: TextInputLayout = findViewById(R.id.boxPhone)

        val apply : Button = findViewById(R.id.btn_apply)
        apply.setOnClickListener {
            if (name.text.toString() == "" || phone.text.toString().isEmpty() ||
signature.text.toString().isEmpty()) {
                val builder = AlertDialog.Builder(this)
                builder.setTitle("Error").setMessage("All fields must be filled.")
                builder.setNegativeButton("Accept", null)
                val alert11 = builder.create()
                alert11.show()
            }
            else if (name.length() >= 50) {
                box.error = "Full name too long"
            }
            else if  (signature.length() > 4) {
                box2.error = "PIN not valid"
            }
            else if (phone.length() >= 10) {
                box3.error = "Mobile phone number too long"
            }
            else {
                changeInfo()
                changePicture()
                val builder = AlertDialog.Builder(this)
                builder.setTitle("Success").setMessage("Your information has
been changed successfully")
                builder.setNegativeButton("Accept", null)
                val alert11 = builder.create()
                alert11.show()
            }
        }

        name.onFocusChangeListener = View.OnFocusChangeListener { b,
hasFocus ->
            if (!hasFocus) {
                name.setHint(R.string.NameHint)
                hideKeyboard(b)
            } else {
                name.hint = ""
                box.error = null
            }
        }
        phone.onFocusChangeListener = View.OnFocusChangeListener { b,
hasFocus ->
            if (!hasFocus) {
```

```
            phone.setHint(R.string.PhoneHint)
            hideKeyboard(b)
        } else {
            phone.hint = ""
            box3.error = null
        }
    }
    signature.onFocusChangeListener = View.OnFocusChangeListener { b,
hasFocus ->
        if (!hasFocus) {
            signature.setHint(R.string.PINHint)
            hideKeyboard(b)
        } else {
            signature.hint = ""
            box2.error = null
        }
    }

    val card : CardView = findViewById(R.id.card_profile)
    val selectPicture = card.findViewById<ImageView>(R.id.profilePicture)
    selectPicture.setOnClickListener { openGallery()}

    val remove : ImageButton = findViewById(R.id.removePicture)
    remove.setOnClickListener {
        val imageView = findViewById<ImageView>(R.id.profilePicture)
        imageView.setImageResource(0)
        imageView.drawable
    }

    val fullname : TextView = findViewById(R.id.inputNameRegister)
    val numberphone : TextView = findViewById(R.id.inputPhone)
    val pin : TextView = findViewById(R.id.inputPIN)
    val db = FirebaseFirestore.getInstance()
    val df: DocumentReference = db.collection("Users").document(email)
    df.get().addOnSuccessListener { documentSnapshot ->
        val name2 = documentSnapshot.getString("Name").toString()
        val pin2 = documentSnapshot.getString("PIN").toString()
        val phone2 = documentSnapshot.getString("Phone").toString()
        fullname.text = name2
        pin.text = pin2
        numberphone.text = phone2
    }

    submenu()

}

private fun openGallery() {
    val gallery = Intent(Intent.ACTION_PICK,
MediaStore.Images.Media.INTERNAL_CONTENT_URI)
```

```
        startActivityForResult(gallery, PICK_IMAGE)
    }

    override fun onActivityResult(requestCode: Int, resultCode: Int, data:
Intent?) {
        val imageView = findViewById<ImageView>(R.id.profilePicture)
        super.onActivityResult(requestCode, resultCode, data)
        if (resultCode == RESULT_OK && requestCode == PICK_IMAGE) {
            imageUri = data?.data
            imageView!!.setImageURI(imageUri)
        }
    }

    private fun changePicture() {
        val user = Firebase.auth.currentUser
        val email= user.email
        val fileRef =
FirebaseStorage.getInstance().reference.child("profile/$email")
        val imageView = findViewById<ImageView>(R.id.profilePicture)
        fileRef.delete()
        if (hasNullOrEmptyDrawable(imageView))
        {

        } else {
            val bitmap = (imageView.drawable as BitmapDrawable).bitmap
            imageView.isDrawingCacheEnabled = true
            imageView.buildDrawingCache()
            val baos = ByteArrayOutputStream()
            bitmap.compress(Bitmap.CompressFormat.JPEG, 100, baos)
            val data = baos.toByteArray()
            fileRef.putBytes(data)
        }
    }

    private fun openActivity1() {

        val user = FirebaseAuth.getInstance().currentUser
        val userEmail = user.email
        val db = FirebaseFirestore.getInstance()
        val df = db.collection("Users").document(userEmail)
        df.get().addOnSuccessListener { documentSnapshot ->

            if (documentSnapshot.getString("Job") == "MRO engineer" ||
                    documentSnapshot.getString("Job") == "CAMO engineer"
                    || documentSnapshot.getString("Job") == "Total access
engineer") {
                val intent = Intent(this, EngineerHomeActivity::class.java)
                startActivity(intent)
                finish()
            }
            if (documentSnapshot.getString("Job") == "Administrator") {
                val intent = Intent(this, AdminHomeActivity::class.java)
                startActivity(intent)
                finish()
            }

            if (documentSnapshot.getString("Job") == "Airplane maintenance
technician") {
                val intent = Intent(this, AMTHomeActivity::class.java)
                startActivity(intent)
                finish()
            }
        }
    }

    private fun changeInfo() {
        val name : EditText = findViewById(R.id.inputNameRegister)
        val phone : EditText = findViewById(R.id.inputPhone)
        val pin : EditText = findViewById(R.id.inputPIN)
        val user = Firebase.auth.currentUser
        val email= user.email
        val db = FirebaseFirestore.getInstance()

db.collection("Users").document(email).update("Name",name.text.toString(),
                "Phone",phone.text.toString(),
                "PIN",pin.text.toString())
    }

    private fun hasNullOrEmptyDrawable(iv: ImageView?): Boolean {
        val drawable = iv!!.drawable
        val bitmapDrawable = if (drawable is BitmapDrawable) drawable else null

        return bitmapDrawable == null || bitmapDrawable.bitmap == null
    }

    private fun submenu() {
        val menu : ImageButton = findViewById(R.id.btn_menu)
        menu.setOnClickListener {
            val inflater = LayoutInflater.from(this)
            val popupView: View = inflater.inflate(R.layout.menu_home, null)
            val focusable = true
            val popupWindow = PopupWindow(popupView, 400,400,focusable)
            popupWindow.showAsDropDown(menu,0,0)
        }
    }

    private fun hideKeyboard(view: View) {
        val inputMethodManager =
getSystemService(INPUT_METHOD_SERVICE) as InputMethodManager
```

```
        inputMethodManager.hideSoftInputFromWindow(view.windowToken, 0)
    }

}
```

A.16. Engineer home activity

```
class EngineerHomeActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {

        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_home_engineer)

        val btnEngineer = findViewById<ImageButton>(R.id.btn_engineer)
        btnEngineer.setOnClickListener { openActivity1() }

        val menu : ImageButton = findViewById(R.id.btn_menu)
        menu.visibility = View.GONE

        val btnMail = findViewById<ImageButton>(R.id.btn_message)
        btnMail.setOnClickListener {
            val intent = Intent(this, MailActivity::class.java)
            startActivity(intent) }

        val calendar = Calendar.getInstance()
        val date = calendar.time

        val stringYear= DateFormat.format("yy", date) as String
        val stringMonth = DateFormat.format("MM", date) as String
        val stringNumber = DateFormat.format("dd", date) as String
        val stringDay = DateFormat.format("EEEE", date) as String
        val textDay = findViewById<TextView>(R.id.tool_title)
        textDay.text = ("$stringDay , $stringNumber / $stringMonth / $stringYear")

        val back : ImageButton = findViewById(R.id.btn_back)
        back.setBackgroundResource(R.drawable.ic_baseline_logout_24)

        back.setOnClickListener {
            val builder = AlertDialog.Builder(this)
            builder.setTitle("Warning!").setMessage("You sure, that you want to
logout?")
            builder.setPositiveButton(
                    "Yes"
            ) { dialog, id ->
                val i = Intent(this, WelcomeActivity::class.java)
                startActivity(i)
            }
            builder.setNegativeButton(
                "No"
            ) { dialog, id -> dialog.cancel() }
            val alert = builder.create()
            alert.show()
        }

        val edit : ImageButton = findViewById(R.id.btn_edit_info)
        edit.setOnClickListener {
            val intent = Intent(this, EditUserInformation::class.java)
            startActivity(intent)
        }

        displayEmailJob()
    }

    private fun displayEmailJob() {
        val user = Firebase.auth.currentUser
        val email= user.email
        val emailTextView = findViewById<TextView>(R.id.mailTextView)
        val jobTextView = findViewById<TextView>(R.id.loginAsTextView)
        val nameTextView = findViewById<TextView>(R.id.welcomeTextView)
        emailTextView.text = ("$email")
        val db = FirebaseFirestore.getInstance()
        val df: DocumentReference = db.collection("Users").document(email)
        df.get().addOnSuccessListener { documentSnapshot ->
            val job = documentSnapshot.getString("Job")
            jobTextView.text = ("$job")
            val name = documentSnapshot.getString("Name")
            nameTextView.text= ("$name")
        }

        val ref = FirebaseStorage.getInstance().reference.child("profile/$email")
        val profile : CardView = this.findViewById(R.id.card_profile)
        val imageView = profile.findViewById<ImageView>(R.id.profilePicture)
        imageView.drawable
        ref.downloadUrl.addOnSuccessListener { Uri ->
            val imageURL = Uri.toString()
            Glide.with(this)
                .load(imageURL)
                .into(imageView)
        }.addOnFailureListener {
        }
    }

    private fun openActivity1() {
        val intent = Intent(this, CheckListActivity::class.java)
        startActivity(intent)
    }

}
```

A.17. Fleet activity

```
class FleetActivity : AppCompatActivity() , MyItemClickListener,
MyItemLongClickListener {

    private var mAdapter: FleetAdapter? = null

    override fun onCreate(savedInstanceState: Bundle?) {

        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_fleet)

        val text : TextView = findViewById(R.id.tool_title)
        text.text = "Aircraft list"

        val back = findViewById<View>(R.id.btn_back) as ImageButton
        back.setOnClickListener { openActivity1() }

        initView()
        initData()

        showMenu()

        val btn_search : Button = findViewById(R.id.button_search)
        btn_search.onFocusChangeListener = View.OnFocusChangeListener { a,
hasFocus ->
            if (!hasFocus) {
                btn_search.setHint("Search")
                hideKeyboard(a)
            } else btn_search.hint = ""
        }

        btn_search.setOnClickListener {
            val search : EditText = findViewById(R.id.search)
            if (search.text.toString().isEmpty()) {

            } else {
                searchBar()
            }
        }

        val btn_reset: Button = findViewById(R.id.button_reset)
        btn_reset.setOnClickListener {
            initData()
            btn_search.setHint("Search")
        }

    }
```

```
   private fun searchBar() {
      val search : EditText = findViewById(R.id.search)
      val db : FirebaseFirestore= FirebaseFirestore.getInstance()
      val rvFleet = findViewById<RecyclerView>(R.id.recyclerViewFleetList)
      db.collection("Fleet")
            .whereEqualTo("Registration",
search.text.toString()).get().addOnSuccessListener { documents ->
               val mData: ArrayList<FleetModel> = ArrayList()
               for (document in documents) {
                  val taskItem: FleetModel =
document.toObject(FleetModel::class.java)
                  mData.add(taskItem)
               }
               val mAdapter = FleetAdapter(mData)
               rvFleet.adapter = mAdapter
               mAdapter.setOnItemClickListener(this)
               mAdapter.setOnItemLongClickListener(this)
            }
   }

   private fun hideKeyboard(view: View) {
      val inputMethodManager =
getSystemService(INPUT_METHOD_SERVICE) as InputMethodManager
      inputMethodManager.hideSoftInputFromWindow(view.windowToken, 0)
   }

   private fun openActivity1() {
      val intent = Intent(this, AdminHomeActivity::class.java)
      startActivity(intent)
   }

   private fun initView() {
      val rvUsers= findViewById<RecyclerView>(R.id.recyclerViewFleetList)
      rvUsers.layoutManager = LinearLayoutManager(this)
      rvUsers.itemAnimator = DefaultItemAnimator()
   }

   private fun initData() {
      val db : FirebaseFirestore = FirebaseFirestore.getInstance()
      val collection: CollectionReference = db.collection("Fleet")
      val query : Query = collection
      val rvFleet = findViewById<RecyclerView>(R.id.recyclerViewFleetList)
      query.get().addOnCompleteListener { task ->
         if (task.isSuccessful) {
            val mData: ArrayList<FleetModel> = ArrayList()
            for (document in task.result) {
               val taskItem: FleetModel =
document.toObject(FleetModel::class.java)
               mData.add(taskItem)
            }
```

```
                val mAdapter = FleetAdapter(mData)
                rvFleet.adapter = mAdapter
                mAdapter.setOnItemClickListener(this)
                mAdapter.setOnItemLongClickListener(this)
            }
        }

    }

    override fun onItemClick(view: View?, position: Int) {
        val db: FirebaseFirestore = FirebaseFirestore.getInstance()
        val collection: CollectionReference = db.collection("Fleet")
        val query: Query = collection
        query.get().addOnCompleteListener { task ->
            if (task.isSuccessful) {
                val mData: ArrayList<FleetModel> = ArrayList()
                for (document in task.result) {
                    val taskItem: FleetModel =
document.toObject(FleetModel::class.java)
                    mData.add(taskItem)
                }

                val bean: FleetModel = mData[position - 1]
                val fleet = bean.Registration

                val user = FirebaseAuth.getInstance().currentUser
                val userEmail = user.email
                val db = FirebaseFirestore.getInstance()
                val df = db.collection("Users").document(userEmail)
                df.get().addOnSuccessListener { documentSnapshot ->

                    if (documentSnapshot.getString("Job") == "Cabin crew") {

                        val intent = Intent(this, DamageMapActivity::class.java)
                        startActivity(intent)
                        finish()
                    }

                    if (documentSnapshot.getString("Job") == "Administrator") {

                        val inflater = LayoutInflater.from(this)
                        val popupView: View =
inflater.inflate(R.layout.pop_window_fleet, null)
                        val focusable = true
                        val popupWindow = PopupWindow(popupView, 850,
ViewGroup.LayoutParams.WRAP_CONTENT, focusable)

                        val popFleet: TextView =
popupView.findViewById(R.id.tool_title)
```

```
                        popFleet.text = fleet
                        popupWindow.showAtLocation(view, Gravity.CENTER, 0, 0)


popupWindow.contentView.findViewById<ImageButton>(R.id.btn_menu).visibilit
y = View.GONE

popupWindow.contentView.findViewById<ImageButton>(R.id.btn_back).visibilit
y = View.GONE


popupWindow.contentView.findViewById<Button>(R.id.pop_damage_map).set
OnClickListener {
                            popupWindow.dismiss()
                            val intent = Intent(this, DamageMapActivity::class.java)
                            startActivity(intent)
                            finish()
                        }


popupWindow.contentView.findViewById<Button>(R.id.pop_documentation).set
OnClickListener {
                            val SHARED_PREF_AIRCRAFT = "aircraft"
                            val KEY_AIRCRAFT = "key_aircraft"
                            val sp = getSharedPreferences(SHARED_PREF_AIRCRAFT,
MODE_PRIVATE)
                            val editor = sp.edit()
                            editor.putString(KEY_AIRCRAFT, fleet)
                            editor.apply()
                            val intent = Intent(this, DocumentsActivity::class.java)
                            startActivity(intent)
                        }


popupWindow.contentView.findViewById<Button>(R.id.pop_delete_fleet).setOn
ClickListener {

                            val inflater3 = LayoutInflater.from(this)
                            val popupView3: View =
inflater3.inflate(R.layout.pop_window_pin, null)
                            val focusable3 = true
                            val popupWindow3 = PopupWindow(popupView3, 900, 400,
focusable3)
                            popupWindow3.showAtLocation(view, Gravity.CENTER, 0, 0)

                            val user = FirebaseAuth.getInstance().currentUser
                            val userEmail = user.email
                            val db = FirebaseFirestore.getInstance()
                            val df = db.collection("Users").document(userEmail)
                            df.get().addOnSuccessListener { documentSnapshot ->
                        val txtpin =
popupWindow3.contentView.findViewById<TextView>(R.id.inputPIN)
                        val delete: Button =
popupWindow3.contentView.findViewById(R.id.pop_pin)
                        delete.setOnClickListener {
                           val builder = AlertDialog.Builder(this)
                           builder.setTitle("CAUTION")
                           builder.setMessage("Are you sure you want to
continue?")
                           builder.setPositiveButton("Accept") { dialog, which ->
                              if (documentSnapshot.getString("PIN") ==
txtpin.text.toString()) {

db.collection("Fleet").document(fleet.toString()).delete()
                                 popupWindow3.dismiss()
                                 val intent = intent
                                 finish()
                                 startActivity(intent)

                              } else {
                                 val builder1 = AlertDialog.Builder(this)
                                 builder1.setTitle("Error")
                                 builder1.setMessage("Incorrect PIN, please try
again.")
                                 builder1.setPositiveButton("Accept", null)
                                 val dialog1: AlertDialog = builder1.create()
                                 dialog1.show()
                                 popupWindow3.dismiss()
                              }
                           }
                           builder.setNegativeButton("Cancel", null)
                           val dialog: AlertDialog = builder.create()
                           dialog.show()

                        }
                     }


popupWindow.contentView.findViewById<Button>(R.id.pop_edit_fleet).setOnCli
ckListener {

                        val inflater2 = LayoutInflater.from(this)
                        val popupView2: View =
inflater2.inflate(R.layout.activity_addfleet, null)
                        val focusable2 = true
                        val popupWindow2 = PopupWindow(popupView2, 950,
1000, focusable2)
                        popupWindow2.showAtLocation(popupView2,
Gravity.CENTER, 0, 0)
```

```
                val pop : TextView =
popupView2.findViewById(R.id.tool_title)
                pop.text = fleet

                val calendar = Calendar.getInstance()
                val manufacture =
popupWindow2.contentView.findViewById<EditText>(R.id.inputManufacture)
                val yy = calendar.get(Calendar.YEAR)
                val mm = calendar.get(Calendar.MONTH)
                val dd = calendar.get(Calendar.DAY_OF_MONTH)
                val datePicker = DatePickerDialog(
                    this,
                    { _, year, monthOfYear, dayOfMonth ->
                      val date =
                          (dayOfMonth.toString() + "/" + (monthOfYear +
1).toString() + "/" + year.toString())
                      manufacture.setText(date)
                    },
                    yy,
                    mm,
                    dd
                )
                datePicker.setButton(DialogInterface.BUTTON_NEUTRAL,
"CLEAR") { dialog, which ->
                  if (which == DialogInterface.BUTTON_NEUTRAL) {
                    manufacture.setText("-")
                  }
                }

                manufacture.setOnClickListener {
                  datePicker.show()
                }

                val db = FirebaseFirestore.getInstance()
                val df: DocumentReference =
db.collection("Fleet").document(fleet.toString())
                df.get().addOnSuccessListener { documentSnapshot ->

                  val txtregis : TextView =
popupWindow2.contentView.findViewById(R.id.inputRegistration)
                  val regis = documentSnapshot.getString("Registration")
                  txtregis.text = regis

                  val texttype: TextView =
popupWindow2.contentView.findViewById(R.id.inputType)
                  val name = documentSnapshot.getString("Type")
                  texttype.text = name

                  val textdate = manufacture as TextView
                  val manf =
documentSnapshot.getString("Manufacture").toString()
                            textdate.text = manf

                        }

                        val aircraft =
popupWindow2.contentView.findViewById<EditText>(R.id.inputRegistration)
                        val type =
popupWindow2.contentView.findViewById<EditText>(R.id.inputType)
                        aircraft.onFocusChangeListener =
View.OnFocusChangeListener { b, hasFocus ->
                            if (!hasFocus) {
                                aircraft.setHint(R.string.NoHint)
                                hideKeyboard(b)
                            } else aircraft.hint = ""
                        }

                        manufacture.onFocusChangeListener =
View.OnFocusChangeListener { b, hasFocus ->
                            if (!hasFocus) {
                                manufacture.setHint(R.string.NoHint)
                                hideKeyboard(b)
                            } else manufacture.hint = ""
                        }

                        type.onFocusChangeListener =
View.OnFocusChangeListener { b, hasFocus ->
                            if (!hasFocus) {
                                type.setHint(R.string.NoHint)
                                hideKeyboard(b)
                            } else type.hint = ""
                        }

                        val btn : Button =
popupWindow2.contentView.findViewById(R.id.btn_AddFleet)
                        btn.setOnClickListener {
                            val db = FirebaseFirestore.getInstance()
                            if (aircraft.text.isEmpty() || manufacture.text.isEmpty() ||
type.text.isEmpty()) {
                                val builder = AlertDialog.Builder(this)
                                builder.setTitle("Information")
                                builder.setMessage("All fields must be filled")
                                builder.setPositiveButton("Accept", null)
                                val dialog: AlertDialog = builder.create()
                                dialog.show()
                            }

                            if (aircraft.text.isNotEmpty() &&
manufacture.text.isNotEmpty() && type.text.isNotEmpty()) {
                                val df: DocumentReference =
db.collection("Fleet").document(aircraft.text.toString())
                                        df.get().addOnSuccessListener { documentSnapshot -
>
                                            if (documentSnapshot.getString("Registration") ==
aircraft.text.toString()) {
                                                aircraft.error = "Aircraft is already in database."
                                            }
                                            if (aircraft.text.toString().isNotEmpty() ||
manufacture.text.toString().isNotEmpty() || type.text.toString().isNotEmpty()) {
                                                val revisionInfo = hashMapOf(
                                                        "Registration" to aircraft.text.toString(),
                                                        "Manufacture" to
manufacture.text.toString(),
                                                        "Type" to type.text.toString(),
                                                )

db.collection("Fleet").document(aircraft.text.toString())
                                                        .set(revisionInfo)

db.collection("Fleet").document(aircraft.text.toString()).delete()
                                                success()
                                                popupWindow2.dismiss()
                                                val intent = intent
                                                finish()
                                                startActivity(intent)
                                            }
                                        }
                                    }
                                }


popupWindow2.contentView.findViewById<ImageButton>(R.id.btn_menu).visibil
ity = View.GONE

popupWindow2.contentView.findViewById<ImageButton>(R.id.btn_back).visibili
ty = View.GONE

                            }

                            popupView.setOnTouchListener { v, event ->
                                popupWindow.dismiss()
                                true
                            }
                        }
                    }
                }
            }
        }
    }
```

```
   private fun success() {
      val builder = AlertDialog.Builder(this)
      builder.setTitle("Information")
      builder.setMessage("Aircraft edited successfully.")
      builder.setPositiveButton("Accept") { dialog, which ->
         openActivity1()
      }
      val dialog: AlertDialog = builder.create()
      dialog.show()

   }

   private fun showMenu() {
      val button : ImageButton = findViewById(R.id.btn_menu)
      button.setOnClickListener {
         val inflater = LayoutInflater.from(this)
         val popupView: View = inflater.inflate(R.layout.menu, null)
         val focusable = true
         val popupWindow = PopupWindow(popupView, 500, 1000, focusable)
         button.x.toInt()
         button.y.toInt()
         popupWindow.showAsDropDown(button)


popupWindow.contentView.findViewById<LinearLayout>(R.id.sortBy).visibility =
View.GONE

popupWindow.contentView.findViewById<LinearLayout>(R.id.advancedSearch)
.visibility = View.GONE

         popupWindow.contentView.findViewById<TextView>(R.id.add_text).text
= "ADD AIRCRAFT"


popupWindow.contentView.findViewById<LinearLayout>(R.id.add).setOnClickLi
stener {
            popupWindow.dismiss()
            val inflater2 = LayoutInflater.from(this)
            val popupView2: View = inflater2.inflate(R.layout.activity_addfleet,
null)
            val focusable2 = true
            val popupWindow2 = PopupWindow(popupView2, 750, 1000,
focusable2)
            popupWindow2.showAtLocation(popupView2, Gravity.CENTER, 0, 0)

popupWindow2.contentView.findViewById<TextView>(R.id.tool_title).text =
"Add aircraft"
            val aircraft =
popupWindow2.contentView.findViewById<EditText>(R.id.inputRegistration)
            aircraft.onFocusChangeListener = View.OnFocusChangeListener { b,
hasFocus ->
          if (!hasFocus) {
             aircraft.setHint(R.string.NoHint)
             hideKeyboard(b)
          } else aircraft.hint = ""
       }

       val manufacture =
popupWindow2.contentView.findViewById<EditText>(R.id.inputManufacture)
       manufacture.onFocusChangeListener =
View.OnFocusChangeListener { b, hasFocus ->
          if (!hasFocus) {
             manufacture.setHint(R.string.NoHint)
             hideKeyboard(b)
          } else manufacture.hint = ""
       }

       val type =
popupWindow2.contentView.findViewById<EditText>(R.id.inputType)
       type.onFocusChangeListener = View.OnFocusChangeListener { b,
hasFocus ->
          if (!hasFocus) {
             type.setHint(R.string.NoHint)
             hideKeyboard(b)
          } else type.hint = ""
       }

       val model =
popupWindow2.contentView.findViewById<EditText>(R.id.inputModel)
       model.onFocusChangeListener = View.OnFocusChangeListener { b,
hasFocus ->
          if (!hasFocus) {
             model.setHint(R.string.NoHint)
             hideKeyboard(b)
          } else model.hint = ""
       }

       val company =
popupWindow2.contentView.findViewById<EditText>(R.id.inputCompany)
       company.onFocusChangeListener = View.OnFocusChangeListener {
b, hasFocus ->
          if (!hasFocus) {
             company.setHint(R.string.NoHint)
             hideKeyboard(b)
          } else company.hint = ""
       }

       val calendar = Calendar.getInstance()
       val yy = calendar.get(Calendar.YEAR)
       val mm = calendar.get(Calendar.MONTH)
```

```
            val dd = calendar.get(Calendar.DAY_OF_MONTH)
            val datePicker = DatePickerDialog(
                  this,
                  { _, year, monthOfYear, dayOfMonth ->
                     val date =
                           (dayOfMonth.toString() + "/" + (monthOfYear +
1).toString() + "/" + year.toString())
                     manufacture.setText(date)
                  },
                  yy,
                  mm,
                  dd
            )
            datePicker.setButton(DialogInterface.BUTTON_NEUTRAL,
"CLEAR") { dialog, which ->
               if (which == DialogInterface.BUTTON_NEUTRAL) {
                  manufacture.setText("-")
               }
            }

            manufacture.setOnClickListener {
               datePicker.show()
            }

popupWindow2.contentView.findViewById<ImageButton>(R.id.btn_menu).visibil
ity = View.GONE
popupWindow2.contentView.findViewById<ImageButton>(R.id.btn_back).visibili
ty = View.GONE
            val btn_AddFleet : Button =
popupWindow2.contentView.findViewById(R.id.btn_AddFleet)

            btn_AddFleet.setOnClickListener {
               val db = FirebaseFirestore.getInstance()
               if (aircraft.text.isEmpty() || manufacture.text.isEmpty() ||
type.text.isEmpty() || model.text.isNotEmpty() || company.text.isNotEmpty()) {
                  val builder = AlertDialog.Builder(this)
                  builder.setTitle("Information")
                  builder.setMessage("All fields must be filled")
                  builder.setPositiveButton("Accept", null)
                  val dialog: AlertDialog = builder.create()
                  dialog.show()
               }

               if (aircraft.text.isNotEmpty() && manufacture.text.isNotEmpty() &&
type.text.isNotEmpty() && model.text.isNotEmpty() &&
company.text.isNotEmpty() ) {
                  val df: DocumentReference =
db.collection("Fleet").document(aircraft.text.toString())
                  df.get().addOnSuccessListener { documentSnapshot ->
                     if (documentSnapshot.getString("Registration") ==
aircraft.text.toString()) {
                        aircraft.error = "Aircraft is already in database."
                    }
                    if (aircraft.text.toString().isNotEmpty() ||
manufacture.text.toString().isNotEmpty() || type.text.toString().isNotEmpty()) {
                        val revisionInfo = hashMapOf(
                            "Registration" to aircraft.text.toString(),
                            "Manufacture" to manufacture.text.toString(),
                            "Type" to type.text.toString(),
                            "Model" to model.text.toString(),
                            "Company" to company.text.toString())
                        db.collection("Fleet").document(aircraft.text.toString())
                            .set(revisionInfo)
                        showSuccess()

                        aircraft.text.clear()
                        aircraft.hint = "-"
                        manufacture.text.clear()
                        manufacture.hint = "-"
                        type.text.clear()
                        type.hint = "-"
                        model.text.clear()
                        model.hint = "-"
                        company.text.clear()
                        company.hint = "-"
                    }
                }
            }
        }
      }
    }
  }


  private fun showSuccess() {
    val builder = AlertDialog.Builder(this)
    builder.setTitle("Information")
    builder.setMessage("Aircraft added successfully.")
    builder.setPositiveButton("Accept", null)
    val dialog: AlertDialog = builder.create()
    dialog.show()
  }

  override fun onItemLongClick(view: View?, postion: Int) {
    TODO("Not yet implemented")
  }

}
```

A.18. Mail activity

```
class MailActivity : AppCompatActivity() , MyItemClickListener,
MyItemLongClickListener {

    private var mAdapter: MailAdapter? = null

    override fun onCreate(savedInstanceState: Bundle?) {

        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_mail)

        val back = findViewById<View>(R.id.btn_back) as ImageButton
        back.setOnClickListener { openActivity1() }

        val text: TextView = findViewById(R.id.tool_title)
        text.text = "AirDocs mail"

        val btn_search : Button = findViewById(R.id.button_search)
        btn_search.onFocusChangeListener = View.OnFocusChangeListener { a,
hasFocus ->
            if (!hasFocus) {
                btn_search.setHint("Search")
                hideKeyboard(a)
            } else btn_search.hint = ""
        }

        btn_search.setOnClickListener {
            val search : EditText = findViewById(R.id.search)
            if (search.text.toString().isEmpty()) {

            } else {
                searchBar()
            }
        }

        val btn_reset: Button = findViewById(R.id.button_reset)
        btn_reset.setOnClickListener {
            initData()
            btn_search.setHint("Search")
        }

        initView()
        initData()
        showMenu()
    }

    private fun initData() {
        val db: FirebaseFirestore = FirebaseFirestore.getInstance()
        val user = Firebase.auth.currentUser
        val email= user.email
        val collection: CollectionReference =
db.collection("Users").document(email).collection("Messages")
        val query: Query = collection
        val rvMail = findViewById<RecyclerView>(R.id.recyclerViewMailList)
        query.get().addOnCompleteListener { task ->
            if (task.isSuccessful) {
                val mData: ArrayList<MailModel> = ArrayList()
                for (document in task.result) {
                    val taskItem: MailModel =
document.toObject(MailModel::class.java)
                    mData.add(taskItem)
                }
                mData.reverse()
                mAdapter = MailAdapter(mData)
                rvMail.adapter = mAdapter
                mAdapter!!.setOnItemClickListener(this)
                mAdapter!!.setOnItemLongClickListener(this)
                mAdapter!!.notifyDataSetChanged()
            }
        }

    }

    private fun initView() {
        val rvMail = findViewById<RecyclerView>(R.id.recyclerViewMailList)
        rvMail.layoutManager = LinearLayoutManager(this)
        rvMail.itemAnimator = DefaultItemAnimator()
    }

    private fun openActivity1() {

        val user = FirebaseAuth.getInstance().currentUser
        val userEmail = user.email
        val db = FirebaseFirestore.getInstance()
        val df = db.collection("Users").document(userEmail)
        df.get().addOnSuccessListener { documentSnapshot ->

            if (documentSnapshot.getString("Job") == "MRO engineer" ||
                    documentSnapshot.getString("Job") == "CAMO engineer"
                    || documentSnapshot.getString("Job") == "Total access
engineer") {
                val intent = Intent(this, EngineerHomeActivity::class.java)
                startActivity(intent)
                finish()
            }

            if (documentSnapshot.getString("Job") == "Administrator") {
                val intent = Intent(this, AdminHomeActivity::class.java)
                startActivity(intent)
            finish()
        }

        if (documentSnapshot.getString("Job") == "Airplane maintenance
technician") {
            val intent = Intent(this, AMTHomeActivity::class.java)
            startActivity(intent)
            finish()
        }

      }
    }

    private fun searchBar() {
      val search : EditText = findViewById(R.id.search)
      val db : FirebaseFirestore= FirebaseFirestore.getInstance()
      val rvMail= findViewById<RecyclerView>(R.id.recyclerViewMailList)
      db.collection("Users").document(search.text.toString())
          .collection("Messages")
          .whereEqualTo("From",
search.text.toString()).get().addOnSuccessListener { documents ->
            val mData: ArrayList<MailModel> = ArrayList()
            for (document in documents) {
              val taskItem: MailModel =
document.toObject(MailModel::class.java)
              mData.add(taskItem)
            }
            mAdapter = MailAdapter(mData)
            rvMail.adapter = mAdapter
            mAdapter!!.setOnItemClickListener(this)
            mAdapter!!.setOnItemLongClickListener(this)
            mAdapter!!.notifyDataSetChanged()
          }
    }

    private fun hideKeyboard(view: View) {
      val inputMethodManager =
getSystemService(INPUT_METHOD_SERVICE) as InputMethodManager
      inputMethodManager.hideSoftInputFromWindow(view.windowToken, 0)
    }

    override fun onItemClick(view: View?, position: Int) {
      val db: FirebaseFirestore = FirebaseFirestore.getInstance()
      val user = Firebase.auth.currentUser
      val email= user.email
      val collection: CollectionReference =
db.collection("Users").document(email).collection("Messages")
      val query: Query = collection
      query.get().addOnCompleteListener { task ->
        if (task.isSuccessful) {
                val mData: ArrayList<MailModel> = ArrayList()
                for (document in task.result) {
                    val taskItem: MailModel =
document.toObject(MailModel::class.java)
                    mData.add(taskItem)
                }
                mData.reverse()

            }
        }


    }

    private fun showMenu() {
        val button : ImageButton = findViewById(R.id.btn_menu)
        button.setOnClickListener {

            val inflater = LayoutInflater.from(this)
            val popupView: View = inflater.inflate(R.layout.menu_mail, null)
            val focusable = true
            val popupWindow = PopupWindow(popupView, 700, 1000, focusable)
            button.x.toInt()
            button.y.toInt()
            popupWindow.showAsDropDown(button)


popupWindow.contentView.findViewById<LinearLayout>(R.id.send_email).setO
nClickListener {

                val inflater2 = LayoutInflater.from(this)
                val popupView2: View = inflater2.inflate(R.layout.pop_window_mail,
null)
                val popupWindow2 = PopupWindow(popupView2, 900,
ViewGroup.LayoutParams.WRAP_CONTENT, true)

                popupWindow2.showAtLocation(popupView2, Gravity.CENTER, 0, 0)

                val send =
popupWindow2.contentView.findViewById<Button>(R.id.button_send)

                send.setOnClickListener {
                    val timestamp_current = Calendar.getInstance().timeInMillis
                    val hour = Calendar.HOUR_OF_DAY
                    val  formatter = SimpleDateFormat("dd/MM/yyyy");
                    val dateString = formatter.format(Date(timestamp_current))

                    val from =
popupWindow2.contentView.findViewById<EditText>(R.id.editTo)
                    val body =
popupWindow2.contentView.findViewById<EditText>(R.id.editBody)
                val subject =
popupWindow2.contentView.findViewById<EditText>(R.id.editSubject)

                val db = FirebaseFirestore.getInstance()
                val df = db.collection("Users").document(from.text.toString())
                    .collection("Messages")
                df.get().addOnSuccessListener { documentSnapshot ->
                  val revisionInfo = hashMapOf(
                      "From" to from.text.toString(),
                      "Body" to body.text.toString(),
                      "Subject" to subject.text.toString(),
                      "Day" to dateString,
                      "Hour" to hour.toString())
                  df.document().set(revisionInfo)

                  val builder = AlertDialog.Builder(this)
                  builder.setTitle("Success")
                  builder.setMessage("Mail send successfully")
                  builder.setPositiveButton("Accept") { dialog, id ->
                      popupWindow2.dismiss()
                  }
                  val dialog: AlertDialog = builder.create()
                  dialog.show()

                }

            }
          }


popupWindow.contentView.findViewById<LinearLayout>(R.id.send_other).setO
nClickListener {

            popupWindow.dismiss()
            val selectorIntent = Intent(Intent.ACTION_SENDTO)
            selectorIntent.data = Uri.parse("mailto:")

            val emailIntent = Intent(Intent.ACTION_SEND)
            emailIntent.selector = selectorIntent

            this.startActivity(Intent.createChooser(emailIntent, "Send email..."))
          }
        }
    }
    override fun onItemLongClick(view: View?, position: Int) {

    }

}
```

A.19. Register activity

```
class RegisterActivity : AppCompatActivity() {

    var button: ImageButton? = null
    private val PICK_IMAGE = 100
    var imageUri: Uri? = null

    override fun onCreate(savedInstanceState: Bundle?) {

        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_register)

        val title : TextView = findViewById(R.id.tool_title)
        title.text = "Register"

        val menu : ImageButton = findViewById(R.id.btn_menu)
        menu.visibility = View.GONE

        val name: TextInputEditText = findViewById(R.id.inputNameRegister)
        val phone: EditText = findViewById(R.id.inputPhone)
        val email: EditText = findViewById(R.id.inputEmailRegister)
        val password: EditText = findViewById(R.id.inputPasswordRegister)
        val confirm: EditText = findViewById(R.id.inputConfirmPassword)
        val signature:EditText=findViewById(R.id.inputPIN)

        val box: TextInputLayout = findViewById(R.id.inputNameBox)
        val box2: TextInputLayout = findViewById(R.id.boxPIN)
        val box3: TextInputLayout = findViewById(R.id.boxPhone)
        val box4: TextInputLayout = findViewById(R.id.boxEmail)
        val box5: TextInputLayout = findViewById(R.id.boxPassword)
        val box6: TextInputLayout = findViewById(R.id.boxConfirm)

        val back: ImageButton = findViewById(R.id.btn_back)
        val intent = Intent(this, WelcomeActivity::class.java)
        back.setOnClickListener {
            startActivity(intent)
            finish()
        }

        name.onFocusChangeListener = OnFocusChangeListener { b, hasFocus -
>
            if (!hasFocus) {
                name.setHint(R.string.NameHint)
                hideKeyboard(b)
            } else {
                name.hint = ""
                box.error=null
            }
        }
        phone.onFocusChangeListener = OnFocusChangeListener { b, hasFocus
->
            if (!hasFocus) {
                phone.setHint(R.string.PhoneHint)
                hideKeyboard(b)
            } else {
                phone.hint = ""
                box3.error=null
            }
        }
        email.onFocusChangeListener = OnFocusChangeListener { a, hasFocus -
>
            if (!hasFocus) {
                email.setHint(R.string.EmailHint)
                hideKeyboard(a)
            } else {
                email.hint = ""
                box4.error = null
            }
        }
        password.onFocusChangeListener = OnFocusChangeListener { b,
hasFocus ->
            if (!hasFocus) {
                password.setHint(R.string.PasswordHint)
                hideKeyboard(b)
            } else {
                password.hint = ""
                box5.error=null
            }
        }
        confirm.onFocusChangeListener = OnFocusChangeListener { b, hasFocus
->
            if (!hasFocus) {
                confirm.setHint(R.string.ConfirmHint)
                hideKeyboard(b)
            } else {
                confirm.hint = ""
                box6.error=null
            }
        }
        signature.onFocusChangeListener = OnFocusChangeListener { b,
hasFocus ->
            if (!hasFocus) {
                signature.setHint(R.string.PINHint)
                hideKeyboard(b)
            } else {
                signature.hint = ""
                box2.error=null
            }
        }
        val roles = resources.getStringArray(R.array.job_array)
        val spinner: Spinner = findViewById(R.id.spinner)
        val adapter = ArrayAdapter(this, R.layout.spinner_item_selected, roles)
        adapter.setDropDownViewResource(R.layout.spinner_dropdown_item)
        spinner.adapter = adapter
        spinner.avoidDropdownFocus()
        spinner.onItemSelectedListener = object : OnItemSelectedListener {
            override fun onItemSelected(
                    parent: AdapterView<*>,
                    view: View?,
                    position: Int,
                    id: Long
            ) {
                if (parent.getItemAtPosition(position) == "Please choose a job role")
{
                } else {
                    val item = parent.getItemAtPosition(position).toString()
                    Toast.makeText(parent.context, "Selected: $item",
Toast.LENGTH_SHORT).show()
                }
            }
            override fun onNothingSelected(parent: AdapterView<*>?) {}
        }

        val signUp: Button = findViewById(R.id.btn_SignUp)
        signUp.setOnClickListener {
            signUpUser()
        }

        val card : CardView = findViewById(R.id.card_profile)
        val selectPicture = card.findViewById<ImageView>(R.id.profilePicture)
        selectPicture.setOnClickListener { openGallery()}

        val remove : ImageButton = findViewById(R.id.removePicture)
        remove.setOnClickListener {
            val imageView = findViewById<ImageView>(R.id.profilePicture)
            imageView.setImageResource(0)
            imageView.drawable
        }

    }

    private fun openGallery() {
        val gallery = Intent(Intent.ACTION_PICK,
MediaStore.Images.Media.INTERNAL_CONTENT_URI)
        startActivityForResult(gallery, PICK_IMAGE)
    }

    override fun onActivityResult(requestCode: Int, resultCode: Int, data:
```

```kotlin
Intent?) {
        val imageView = findViewById<ImageView>(R.id.profilePicture)
        super.onActivityResult(requestCode, resultCode, data)
        if (resultCode == RESULT_OK && requestCode == PICK_IMAGE) {
            imageUri = data?.data
            imageView!!.setImageURI(imageUri)
        }
    }

    override fun onBackPressed() {
        super.onBackPressed()
        val intent = Intent(this@RegisterActivity, WelcomeActivity::class.java)
        startActivity(intent)
        finish()
    }

    private fun Spinner.avoidDropdownFocus() {
        try {
            val isAppCompat = this is
androidx.appcompat.widget.AppCompatSpinner
            val spinnerClass = if (isAppCompat)
androidx.appcompat.widget.AppCompatSpinner::class.java else
Spinner::class.java
            val popupWindowClass = if (isAppCompat)
androidx.appcompat.widget.ListPopupWindow::class.java else
android.widget.ListPopupWindow::class.java

            val listPopup = spinnerClass
                .getDeclaredField("mPopup")
                .apply { isAccessible = true }
                .get(this)
            if (popupWindowClass.isInstance(listPopup)) {
                val popup = popupWindowClass
                    .getDeclaredField("mPopup")
                    .apply { isAccessible = true }
                    .get(listPopup)
                if (popup is PopupWindow) {
                    popup.isFocusable = false
                }
            }
        } catch (e: Exception) {
            e.printStackTrace()
        }
    }

    private fun hideKeyboard(view: View) {
        val inputMethodManager =
getSystemService(INPUT_METHOD_SERVICE) as InputMethodManager
        inputMethodManager.hideSoftInputFromWindow(view.windowToken, 0)
    }
    private fun showValidate() {
        val builder = AlertDialog.Builder(this)
        builder.setTitle("Verification")
        builder.setMessage("Please check your email for verification.")
        builder.setPositiveButton("Accept", null)
        val dialog: AlertDialog = builder.create()
        dialog.show()
    }

    private fun showAlert() {
        val builder = AlertDialog.Builder(this)
        builder.setTitle("Error")
        builder.setMessage("An authentication error was detected. Please,
contact with the administrator.")
        builder.setPositiveButton("Accept", null)
        val dialog: AlertDialog = builder.create()
        dialog.show()
    }

    private fun showPermission() {
        val builder = AlertDialog.Builder(this)
        builder.setTitle("Success")
        builder.setMessage(
                "The administration has to verify your identity" +
                        " to give you permission. We will send you a mail informing
you."
        )
        builder.setPositiveButton("Accept")  { dialog, which ->
            val intent = Intent(this, WelcomeActivity::class.java)
            startActivity(intent)
            finish()
        }
        val dialog: AlertDialog = builder.create()
        dialog.show()

    }

    private fun signUpUser() {
        val email = findViewById<EditText>(R.id.inputEmailRegister)
        val password: EditText = findViewById(R.id.inputPasswordRegister)
        if (checkName() and checkPhone() and checkEmail()
                and checkPassword() and checkConfirm() and checkJob() and
checkPIN()) {
            FirebaseAuth.getInstance().createUserWithEmailAndPassword(
                    email.text.toString(),
                    password.text.toString()
            ).addOnCompleteListener {
                if (it.isSuccessful) {
                    val user = FirebaseAuth.getInstance().currentUser
                    user.sendEmailVerification()
                    saveUserData()
                    showPermission()
                    showValidate()
                } else {
                    showAlert()
                }
            }
        }
    }

    private fun saveUserData() {

        val name: EditText = findViewById(R.id.inputNameRegister)
        val phone: EditText = findViewById(R.id.inputPhone)
        val email = findViewById<EditText>(R.id.inputEmailRegister)
        val spinner = findViewById<View>(R.id.spinner) as Spinner
        val job = spinner.selectedItem.toString()
        val pin : EditText = findViewById(R.id.inputPIN)
        val db = FirebaseFirestore.getInstance()
        val fileRef = FirebaseStorage.getInstance().reference.child("profile/" +
email.text.toString())
        val imageView = findViewById<ImageView>(R.id.profilePicture)

        if (hasNullOrEmptyDrawable(imageView))
        {

        } else {
            val bitmap = (imageView.drawable as BitmapDrawable).bitmap
            imageView.isDrawingCacheEnabled = true
            imageView.buildDrawingCache()
            val baos = ByteArrayOutputStream()
            bitmap.compress(Bitmap.CompressFormat.JPEG, 100, baos)
            val data = baos.toByteArray()
            fileRef.putBytes(data)
        }

        val userInfo = hashMapOf(
                "Name" to name.text.toString(),
                "Phone" to phone.text.toString(),
                "Email" to email.text.toString(),
                "PIN" to pin.text.toString(),
                "Job" to job,
                "Permission" to "No"
        )

        db.collection("Users").document(email.text.toString()).set(userInfo)
    }

    private fun hasNullOrEmptyDrawable(iv: ImageView?): Boolean {
        val drawable = iv!!.drawable
        val bitmapDrawable = if (drawable is BitmapDrawable) drawable else null

        return bitmapDrawable == null || bitmapDrawable.bitmap == null
    }

    private fun checkName(): Boolean {
        val box: TextInputLayout = findViewById(R.id.inputNameBox)
        val name: EditText = findViewById(R.id.inputNameRegister)
        if (name.text.toString() == "") {
            box.error = "Please enter full name"
            return false
        } else if  (name.length() >= 50) {
            box.error = "Full name too long"
            return false
        } else {
            box.error = null
            return true
        }
    }

    private fun checkPIN(): Boolean {
        val box: TextInputLayout = findViewById(R.id.boxPIN)
        val pin: EditText = findViewById(R.id.inputPIN)
        if (pin.text.toString() == "") {
            box.error = "Please enter a custom PIN"
            return false
        } else if  (pin.length() > 4) {
            box.error = "PIN not valid"
            return false
        } else {
            box.error = null
            return true
        }
    }

    private fun checkPhone(): Boolean {
        val box: TextInputLayout = findViewById(R.id.boxPhone)
        val phone: EditText = findViewById(R.id.inputPhone)
        if (phone.text.isEmpty()) {
            box.error = "Please enter your mobile phone number"
            return false
        } else if (phone.length() >= 10) {
            box.error = "Mobile phone number too long"
            return false
        } else {
            box.error = null
            return true
        }
    }
    private fun checkEmail(): Boolean {
        val email: EditText = findViewById(R.id.inputEmailRegister)
        val box: TextInputLayout = findViewById(R.id.boxEmail)
        if (email.text.isEmpty()) {
            box.error = "Please enter email address"
            return false
        } else {
            box.error = null
            return true
        }
    }

    private fun checkPassword(): Boolean {
        val password: EditText = findViewById(R.id.inputPasswordRegister)
        val passwordREGEX = Pattern.compile("^(?=.*[0-
9])(?=\\S+$).{6,}").toRegex()
        val box: TextInputLayout = findViewById(R.id.boxPassword)
        if (password.text.isEmpty()) {
            box.error = "Please enter password"
            return false
        } else if (passwordREGEX.matches(password.toString())) {
            box.error = "At least 6 characters, minimum 1 letter and 1 number"
            return false
        } else {
            box.error=null
            return true
        }
    }

    private fun checkConfirm(): Boolean {
        val password: EditText = findViewById(R.id.inputPasswordRegister)
        val confirm : EditText = findViewById(R.id.inputConfirmPassword)
        val box: TextInputLayout = findViewById(R.id.boxConfirm)
        if (confirm.text.isEmpty()) {
            box.error = "Please confirm your password"
            return false
        }
        else if (password.text.toString() != confirm.text.toString()) {
            box.error = "Password does not match"
            return false
        } else {
            box.error=null
            return true
        }
    }

    private fun checkJob(): Boolean {
        val spinner = findViewById<View>(R.id.spinner) as Spinner
        if (spinner.selectedItem.toString() == "Please choose a job role")
        {
            val builder = AlertDialog.Builder(this)
            builder.setTitle("Error")
            builder.setMessage("Necessary to select a job role")
            builder.setPositiveButton("Accept", null)
            val dialog: AlertDialog = builder.create()
            dialog.show()
            return false
        } else {
            return true
        }
    }

}
```

A.20. SDT RB activity

```
class sdtRbActivity : AppCompatActivity() {

    private var mAdapter: AircraftAdapter? = null

    override fun onCreate(savedInstanceState: Bundle?) {

        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_add_damages)

        val title: TextView = findViewById(R.id.tool_title)
        title.text = "Report back"

        val button : ImageButton = findViewById(R.id.btn_menu)
        button.visibility = View.GONE

        val location = findViewById<EditText>(R.id.inputLocation)
        val dimen1 = findViewById<EditText>(R.id.input_dimen_1)
        val dimen2 = findViewById<EditText>(R.id.input_dimen_2)
        val dimen3 = findViewById<EditText>(R.id.input_dimen_3)
        val comment = findViewById<EditText>(R.id.inputComment)
        val ref: EditText = findViewById(R.id.input_ref)
        val sn: EditText = findViewById(R.id.input_SN)
        val pn: EditText = findViewById(R.id.input_PN)
        val fh: EditText = findViewById(R.id.input_FH)
        val fc: EditText = findViewById(R.id.input_FC)
        val days: EditText = findViewById(R.id.input_Days)
        val fh2: EditText = findViewById(R.id.input_FH_repair)
        val fc2: EditText = findViewById(R.id.input_FC_repair)
        val days2: EditText = findViewById(R.id.input_Days_repair)
        val dimen = resources.getStringArray(R.array.dimen_array)
        val spinner: Spinner = findViewById(R.id.spinner_dimen)
        val adapter = ArrayAdapter(this, R.layout.spinner_item_selected2, dimen)
        val type = resources.getStringArray(R.array.type_array)
        val spinner_type: Spinner = findViewById(R.id.spinner_Type)
        val adapter_type = ArrayAdapter(this, R.layout.spinner_item_selected,
type)

        ref.onFocusChangeListener = View.OnFocusChangeListener { b,
hasFocus ->
            if (!hasFocus) {
                ref.setHint(R.string.NoHint); hideKeyboard(b)
            } else ref.hint = ""
        }

        comment.onFocusChangeListener = View.OnFocusChangeListener { b,
hasFocus ->
            if (!hasFocus) {
                comment.setHint(R.string.NoHint); hideKeyboard(b)
            } else comment.hint = ""
        }

        sn.onFocusChangeListener = View.OnFocusChangeListener { b,
hasFocus ->
            if (!hasFocus) {
                sn.setHint(R.string.NoHint); hideKeyboard(b)
            } else sn.hint = ""
        }

        pn.onFocusChangeListener = View.OnFocusChangeListener { b,
hasFocus ->
            if (!hasFocus) {
                pn.setHint(R.string.NoHint); hideKeyboard(b)
            } else pn.hint = ""
        }

        location.onFocusChangeListener = View.OnFocusChangeListener { b,
hasFocus ->
            if (!hasFocus) {
                location.setHint(R.string.NoHint); hideKeyboard(b)
            } else location.hint = ""
        }

        dimen1.onFocusChangeListener = View.OnFocusChangeListener { b,
hasFocus ->
            if (!hasFocus) {
                dimen1.setHint(R.string.NoHint); hideKeyboard(b)
            } else dimen1.hint = ""
        }

        dimen2.onFocusChangeListener = View.OnFocusChangeListener { b,
hasFocus ->
```

```
            if (!hasFocus) {
                dimen2.setHint(R.string.NoHint); hideKeyboard(b)
            } else dimen2.hint = ""
        }

        dimen3.onFocusChangeListener = View.OnFocusChangeListener { b,
hasFocus ->
            if (!hasFocus) {
                dimen3.setHint(R.string.NoHint); hideKeyboard(b)
            } else dimen3.hint = ""
        }


        fh.onFocusChangeListener = View.OnFocusChangeListener { b, hasFocus
->
            if (!hasFocus) {
                fh.setHint(R.string.NoHint)
                hideKeyboard(b)
            } else fh.hint = ""
        }
        fc.onFocusChangeListener = View.OnFocusChangeListener { b, hasFocus
->
            if (!hasFocus) {
                fc.setHint(R.string.NoHint)
                hideKeyboard(b)
            } else fc.hint = ""
        }
        days.onFocusChangeListener = View.OnFocusChangeListener { b,
hasFocus ->
            if (!hasFocus) {
                days.setHint(R.string.NoHint)
                hideKeyboard(b)
            } else days.hint = ""
        }

        fh2.onFocusChangeListener = View.OnFocusChangeListener { b,
hasFocus ->
            if (!hasFocus) {
                fh2.setHint(R.string.NoHint)
                hideKeyboard(b)
            } else fh2.hint = ""
        }
        fc2.onFocusChangeListener = View.OnFocusChangeListener { b,
hasFocus ->
            if (!hasFocus) {
                fc2.setHint(R.string.NoHint)
                hideKeyboard(b)
            } else fc2.hint = ""
        }
        days2.onFocusChangeListener = View.OnFocusChangeListener { b,
hasFocus ->
            if (!hasFocus) {
                days2.setHint(R.string.NoHint)
                hideKeyboard(b)
            } else days2.hint = ""
        }

        adapter.setDropDownViewResource(R.layout.spinner_dropdown_item)
        spinner.adapter = adapter
        spinner.avoidDropdownFocus()
        spinner.onItemSelectedListener = object :
AdapterView.OnItemSelectedListener {
            override fun onItemSelected(
                    parent: AdapterView<*>,
                    view: View?,
                    position: Int,
                    id: Long
            ) {
                if (parent.getItemAtPosition(position) == "-") {
                } else {
                    val SHARED_PREF_DIMEN = "DIMEN";
                    val KEY_DIMEN = "key_DIMEN";
                    val mSpnValue: String = spinner.selectedItem.toString()
                    val myPrefs: SharedPreferences =
getSharedPreferences(SHARED_PREF_DIMEN, MODE_PRIVATE)
                    val prefsEditor = myPrefs.edit()
                    prefsEditor.putString(KEY_DIMEN, mSpnValue)
                    prefsEditor.apply()
                }
            }

            override fun onNothingSelected(parent: AdapterView<*>?) {}
        }


adapter_type.setDropDownViewResource(R.layout.spinner_dropdown_item)
        spinner_type.adapter = adapter_type
        spinner_type.avoidDropdownFocus()
        spinner_type.onItemSelectedListener = object :
AdapterView.OnItemSelectedListener {
            override fun onItemSelected(
                    parent: AdapterView<*>,
                    view: View?,
                    position: Int,
                    id: Long
            ) {
                if (parent.getItemAtPosition(position) == "-") {
                }
            }
```

```
           override fun onNothingSelected(parent: AdapterView<*>?) {}
        }


        val cat = resources.getStringArray(R.array.cat_array)
        val spinner1: Spinner = findViewById(R.id.spinner_cat)
        val adapter1 = ArrayAdapter(this, R.layout.spinner_item_selected, cat)
        adapter1.setDropDownViewResource(R.layout.spinner_dropdown_item)
        spinner1.adapter = adapter1
        spinner1.avoidDropdownFocus()
        spinner1.onItemSelectedListener = object :
AdapterView.OnItemSelectedListener {
           override fun onItemSelected(
                 parent: AdapterView<*>,
                 view: View?,
                 position: Int,
                 id: Long
           ) {

           }

           override fun onNothingSelected(parent: AdapterView<*>?) {}
        }

        val btn_back : ImageButton = findViewById(R.id.btn_back)
        btn_back.setOnClickListener {
           val intent = Intent(this, DamageActivity::class.java)
           startActivity(intent)
        }

        val calendar = Calendar.getInstance()
        val edit = findViewById<View>(R.id.input_entry) as EditText
        val yy = calendar.get(Calendar.YEAR)
        val mm = calendar.get(Calendar.MONTH)
        val dd = calendar.get(Calendar.DAY_OF_MONTH)
        val datePicker = DatePickerDialog(
              this@sdtRbActivity,
              { _, year, monthOfYear, dayOfMonth ->
                 val date =
                       (dayOfMonth.toString() + "/" + (monthOfYear + 1).toString() +
"/" + year.toString())
                 edit.setText(date)
              },
              yy,
              mm,
              dd
        )
        datePicker.setButton(DialogInterface.BUTTON_NEUTRAL, "CLEAR") { _,
which ->
           if (which == DialogInterface.BUTTON_NEUTRAL) {
              edit.setText("-")
           }
        }

        edit.setOnClickListener {
           datePicker.show()
        }

        val edit2 = findViewById<View>(R.id.input_closing) as EditText
        val calendar2 = Calendar.getInstance()
        val yy2 = calendar2.get(Calendar.YEAR)
        val mm2 = calendar2.get(Calendar.MONTH)
        val dd2 = calendar2.get(Calendar.DAY_OF_MONTH)
        val datePicker2 = DatePickerDialog(
              this@sdtRbActivity,
              { _, year, monthOfYear, dayOfMonth ->
                 val date2 =
                       (dayOfMonth.toString() + "/" + (monthOfYear + 1).toString() +
"/" + year.toString())
                 edit2.setText(date2)
              },
              yy2,
              mm2,
              dd2
        )
        datePicker2.setButton(DialogInterface.BUTTON_NEUTRAL, "CLEAR") {
_, which ->
           if (which == DialogInterface.BUTTON_NEUTRAL) {
              edit2.setText("-")
           }
        }

        edit2.setOnClickListener() {
           datePicker2.show()
        }

        changeButton()

        displayInfo()

        confirm()


}

private fun hideKeyboard(view: View) {
        val inputMethodManager =
getSystemService(INPUT_METHOD_SERVICE) as InputMethodManager
        inputMethodManager.hideSoftInputFromWindow(view.windowToken, 0)
}
```

```
    private fun changeButton() {
        val btn_add : Button = findViewById(R.id.btn_Add)
        val user = FirebaseAuth.getInstance().currentUser
        val userEmail = user.email
        val db = FirebaseFirestore.getInstance()
        val df3 = db.collection("Users").document(userEmail)
        df3.get().addOnSuccessListener { documentSnapshot ->
            if (documentSnapshot.getString("Job") == "MRO engineer") {
                btn_add.setText("Certify")
            }
            if (documentSnapshot.getString("Job") == "CAMO engineer") {
                btn_add.setText("Check")
            }
            if (documentSnapshot.getString("Job") == "Airplane maintenance
technician") {
                btn_add.setText("SDT RB")
            }
            if (documentSnapshot.getString("Job") == "Total access engineer"
                    || documentSnapshot.getString("Job") == "Administrator") {
                btn_add.setText("Certify/check")
            }
        }
    }

    private fun confirm() {
        val confirm: Button = findViewById(R.id.btn_Add)
        confirm.setOnClickListener {
            updateData()
            val intent = Intent(this, DamageActivity::class.java)
            startActivity(intent)
            finish()
        }
    }

    private fun updateData() {

        val SHARED_PREF_DAMAGE = "DAMAGE"
        val KEY_DAMAGE= "key_DAMAGE"
        val sp0= getSharedPreferences(SHARED_PREF_DAMAGE,
MODE_PRIVATE)
        val damage = sp0.getString(KEY_DAMAGE, null)

        val SHARED_PREF_AIRCRAFT = "aircraft";
        val KEY_AIRCRAFT = "key_aircraft";
        val sp = getSharedPreferences(SHARED_PREF_AIRCRAFT,
MODE_PRIVATE)
        val aircraft = sp.getString(KEY_AIRCRAFT, null)

        val SHARED_PREF_START = "start"
        val KEY_START = "key_start"
        val sp1 = getSharedPreferences(SHARED_PREF_START,
MODE_PRIVATE)
        val start = sp1.getString(KEY_START, null)

        val SHARED_PREF_check = "check"
        val KEY_check = "key_check"
        val sp2 = getSharedPreferences(SHARED_PREF_check,
MODE_PRIVATE)
        val check = sp2.getString(KEY_check, null)

        val items1 = start?.split("/".toRegex())?.toTypedArray()
        val day = items1?.get(0)
        val month = items1?.get(1)
        val year = items1?.get(2)

        val id = aircraft+"_"+check+"_"+day+month+year
        val db = FirebaseFirestore.getInstance()
        val df: DocumentReference = db.collection("Aircraft
revision").document(id)
                .collection("New damages").document(damage.toString())

        df.get().addOnSuccessListener { documentSnapshot ->

            val map = documentSnapshot.getString("Map")
            val certify = documentSnapshot.getString("Certified")
            val checked = documentSnapshot.getString("Checked")
            val certifyBy = documentSnapshot.getString("CertifyBY")
            val checkedBy = documentSnapshot.getString("CheckedBY")

            val spinner = findViewById<View>(R.id.spinner_cat) as Spinner
            val cat = spinner.selectedItem.toString()
            val spinner2: Spinner = findViewById(R.id.spinner_dimen)
            val dimen = spinner2.selectedItem.toString()
            val spinner3 = findViewById<View>(R.id.spinner_Type) as Spinner
            val type = spinner3.selectedItem.toString()
            val ata = findViewById<EditText>(R.id.inputATA)
            val location = findViewById<TextView>(R.id.inputLocation)
            val dimen1: TextView = findViewById(R.id.input_dimen_1)
            val dimen2: TextView = findViewById(R.id.input_dimen_2)
            val dimen3: TextView = findViewById(R.id.input_dimen_3)
            val fh: EditText = findViewById(R.id.input_FH)
            val fc: EditText = findViewById(R.id.input_FC)
            val days: EditText = findViewById(R.id.input_Days)
            val fh2: EditText = findViewById(R.id.input_FH_repair)
            val fc2: EditText = findViewById(R.id.input_FC_repair)
            val days2: EditText = findViewById(R.id.input_Days_repair)
            val entry: EditText = findViewById(R.id.input_entry)
            val closing: EditText = findViewById(R.id.input_closing)
            val pn: EditText = findViewById(R.id.input_PN)
```

```
        val sn: EditText = findViewById(R.id.input_SN)
        val ref: EditText = findViewById(R.id.input_ref)
        val comment: EditText = findViewById(R.id.inputComment)
        val regis : EditText = findViewById(R.id.inputAircraft)

        if (comment.text.toString().isEmpty()) {

            val revisionInfo = hashMapOf(
                "Aircraft" to regis.text.toString(),
                "Inspection FH" to fh.text.toString(),
                "Inspection FC" to fc.text.toString(),
                "Inspection days" to days.text.toString(),
                "Repair FH" to fh2.text.toString(),
                "Repair FC" to fc2.text.toString(),
                "Repair days" to days2.text.toString(),
                "Type" to type,
                "Dim" to dimen,
                "Dim1" to dimen1.text.toString(),
                "Dim2" to dimen2.text.toString(),
                "Dim3" to dimen3.text.toString(),
                "References" to ref.text.toString(),
                "Entry" to entry.text.toString(),
                "Closing" to closing.text.toString(),
                "Description" to type+"_"+location.text.toString(),
                "Comments" to comment.text.toString(),
                "Commentary" to "No",
                "PN" to pn.text.toString(),
                "SN" to sn.text.toString(),
                "Category" to cat,
                "ATA" to ata.text.toString(),
                "Location" to location.text.toString(),
                "Dimension" to dimen1.text.toString() + "x"
                    + dimen2.text.toString() + "x"
                    + dimen3.text.toString() + dimen,
                "ID" to damage,
                "RB" to "Done",
                "Certified" to certify.toString(),
                "Checked" to checked.toString(),
                "Map" to map.toString(),
                "CertifyBY" to certifyBy.toString(),
                "CheckedBY" to checkedBy.toString())

            db.collection("Aircraft revision").document(id)
                .collection("New
damages").document(damage.toString()).set(revisionInfo)

        }

        if (comment.text.toString().isNotEmpty()) {
            val revisionInfo = hashMapOf(
                "Aircraft" to regis.text.toString(),
                "Inspection FH" to fh.text.toString(),
                "Inspection FC" to fc.text.toString(),
                "Inspection days" to days.text.toString(),
                "Repair FH" to fh2.text.toString(),
                "Repair FC" to fc2.text.toString(),
                "Repair days" to days2.text.toString(),
                "Type" to type,
                "Dim" to dimen,
                "Dim1" to dimen1.text.toString(),
                "Dim2" to dimen2.text.toString(),
                "Dim3" to dimen3.text.toString(),
                "References" to ref.text.toString(),
                "Entry" to entry.text.toString(),
                "Closing" to closing.text.toString(),
                "Description" to type+"_"+location.text.toString(),
                "Comments" to comment.text.toString(),
                "Commentary" to "Yes",
                "PN" to pn.text.toString(),
                "SN" to sn.text.toString(),
                "Category" to cat,
                "ATA" to ata.text.toString(),
                "Location" to location.text.toString(),
                "Dimension" to dimen1.text.toString() + "x"
                    + dimen2.text.toString() + "x"
                    + dimen3.text.toString() + dimen,
                "ID" to damage,
                "RB" to "Done",
                "Certified" to certify.toString(),
                "Checked" to checked.toString(),
                "Map" to map.toString(),
                "CertifyBY" to certifyBy.toString(),
                "CheckedBY" to checkedBy.toString())

            db.collection("Aircraft revision").document(id)
                .collection("New
damages").document(damage.toString()).set(revisionInfo)

         }
      }
   }

   private fun Spinner.avoidDropdownFocus() {
      try {
         val isAppCompat = this is
androidx.appcompat.widget.AppCompatSpinner
         val spinnerClass = if (isAppCompat)
androidx.appcompat.widget.AppCompatSpinner::class.java else
Spinner::class.java
```

```
            val popupWindowClass = if (isAppCompat)
androidx.appcompat.widget.ListPopupWindow::class.java else
android.widget.ListPopupWindow::class.java

            val listPopup = spinnerClass
                .getDeclaredField("mPopup")
                .apply { isAccessible = true }
                .get(this)
            if (popupWindowClass.isInstance(listPopup)) {
                val popup = popupWindowClass
                    .getDeclaredField("mPopup")
                    .apply { isAccessible = true }
                    .get(listPopup)
                if (popup is PopupWindow) {
                    popup.isFocusable = false
                }
            }
        } catch (e: Exception) {
            e.printStackTrace()
        }
    }

    private fun displayInfo()  {

        val SHARED_PREF_DAMAGE = "DAMAGE"
        val KEY_DAMAGE= "key_DAMAGE"
        val sp0= getSharedPreferences(SHARED_PREF_DAMAGE,
MODE_PRIVATE)
        val damage = sp0.getString(KEY_DAMAGE, null)

        val SHARED_PREF_AIRCRAFT = "aircraft";
        val KEY_AIRCRAFT = "key_aircraft";
        val sp = getSharedPreferences(SHARED_PREF_AIRCRAFT,
MODE_PRIVATE)
        val aircraft = sp.getString(KEY_AIRCRAFT, null)

        val SHARED_PREF_START = "start"
        val KEY_START = "key_start"
        val sp1 = getSharedPreferences(SHARED_PREF_START,
MODE_PRIVATE)
        val start = sp1.getString(KEY_START, null)

        val SHARED_PREF_check = "check"
        val KEY_check = "key_check"
        val sp2 = getSharedPreferences(SHARED_PREF_check,
MODE_PRIVATE)
        val check = sp2.getString(KEY_check, null)

        val items1 = start?.split("/".toRegex())?.toTypedArray()
        val day = items1?.get(0)
        val month = items1?.get(1)
        val year = items1?.get(2)

        val id = aircraft+"_"+check+"_"+day+month+year
        val db = FirebaseFirestore.getInstance()
        val df: DocumentReference = db.collection("Aircraft
revision").document(id)
            .collection("New damages").document(damage.toString())
        df.get().addOnSuccessListener { documentSnapshot ->

            val txtplane = findViewById<TextView>(R.id.inputAircraft)
            val plane = documentSnapshot.getString("Aircraft").toString()
            txtplane.text = plane
            txtplane.isFocusable=false
            txtplane.setBackgroundResource(R.drawable.custom_input)

            val spinner = findViewById<View>(R.id.spinner_cat) as Spinner
            if (documentSnapshot.getString("Category") == "-") {
                for (i in 0 until spinner.adapter.count) {
                    if (spinner.adapter.getItem(i).toString().contains(aircraft.toString()))
{
                        spinner.setSelection(0)
                    }
                }
            } else {
                val value = documentSnapshot.getString("Category")
                setSpinText(spinner, value)
            }

            val spinner3 = findViewById<View>(R.id.spinner_Type) as Spinner
            if (documentSnapshot.getString("Type") == "-") {
                for (i in 0 until spinner3.adapter.count) {
                    if
(spinner3.adapter.getItem(i).toString().contains(aircraft.toString())) {
                        spinner3.setSelection(0)
                    }
                }
            } else {
                val value = documentSnapshot.getString("Type")
                setSpinText(spinner3, value)
            }

            val spinner2: Spinner = findViewById(R.id.spinner_dimen)
            if (documentSnapshot.getString("Dim") == "-") {
                for (i in 0 until spinner2.adapter.count) {
                    if
(spinner2.adapter.getItem(i).toString().contains(aircraft.toString())) {
                        spinner2.setSelection(0)
                    }
                }
            } else {
                val value = documentSnapshot.getString("Dim")
                setSpinText(spinner2, value)
            }

            val txtata = findViewById<TextView>(R.id.inputATA)
            val ata = documentSnapshot.getString("ATA").toString()
            txtata.text = ata

            val txtlocation = findViewById<TextView>(R.id.inputLocation)
            val location = documentSnapshot.getString("Location").toString()
            txtlocation.text = location

            val txtdimen1: TextView = findViewById(R.id.input_dimen_1)
            val dimen1 = documentSnapshot.getString("Dim1").toString()
            txtdimen1.text = dimen1

            val txtdimen2: TextView = findViewById(R.id.input_dimen_2)
            val dimen2 = documentSnapshot.getString("Dim2").toString()
            txtdimen2.text = dimen2


            val txtdimen3: TextView = findViewById(R.id.input_dimen_3)
            val dimen3 = documentSnapshot.getString("Dim3").toString()
            txtdimen3.text = dimen3

            val txtfh: TextView = findViewById(R.id.input_FH)
            val fh = documentSnapshot.getString("Inspection FH").toString()
            txtfh.text = fh

            val txtfc: TextView = findViewById(R.id.input_FC)
            val fc = documentSnapshot.getString("Inspection FC").toString()
            txtfc.text = fc

            val txtdays: TextView= findViewById(R.id.input_Days)
            val days = documentSnapshot.getString("Inspection days").toString()
            txtdays.text = days

            val txtfh2: TextView = findViewById(R.id.input_FH_repair)
            val fh2 = documentSnapshot.getString("Repair FH").toString()
            txtfh2.text = fh2

            val txtfc2: TextView = findViewById(R.id.input_FC_repair)
            val fc2 = documentSnapshot.getString("Repair FC").toString()
            txtfc2.text = fc2

            val txtdays2: TextView= findViewById(R.id.input_Days_repair)
            val days2 = documentSnapshot.getString("Repair days").toString()
            txtdays2.text = days2
```

```
            val txtentry: TextView = findViewById(R.id.input_entry)
            val entry = documentSnapshot.getString("Entry").toString()
            txtentry.text = entry

            val txtclosing: TextView = findViewById(R.id.input_closing)
            val closing = documentSnapshot.getString("Closing").toString()
            txtclosing.text = closing

            val txtpn: TextView = findViewById(R.id.input_PN)
            val pn = documentSnapshot.getString("PN").toString()
            txtpn.text = pn

            val txtsn: TextView = findViewById(R.id.input_SN)
            val sn = documentSnapshot.getString("SN").toString()
            txtsn.text = sn

            val txtref: TextView = findViewById(R.id.input_ref)
            val ref = documentSnapshot.getString("References").toString()
            txtref.text = ref

            val txtcomment: TextView= findViewById(R.id.inputComment)
            val comment = documentSnapshot.getString("Comments").toString()
            txtcomment.text = comment

        }

    }

    private fun setSpinText(spin: Spinner, text: String?) {
        for (i in 0 until spin.adapter.count) {
            if (spin.adapter.getItem(i).toString().contains(text.toString())) {
                spin.setSelection(i)
            }
        }
    }

}
```

A.21. User activity

```
class UserActivity : AppCompatActivity() , MyItemClickListener,
MyItemLongClickListener {

    private var mAdapter: UserAdapter? = null

    override fun onCreate(savedInstanceState: Bundle?) {

        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_users)
        val back = findViewById<View>(R.id.btn_back) as ImageButton
        back.setOnClickListener { openActivity1() }

        val text : TextView = findViewById(R.id.tool_title)
        text.text = "List of users"

        initView()
        initData()

        showMenu()

        val btn_search : Button = findViewById(R.id.button_search)
        btn_search.onFocusChangeListener = View.OnFocusChangeListener { a,
hasFocus ->
            if (!hasFocus) {
                btn_search.setHint("Search")
                hideKeyboard(a)
            } else btn_search.hint = ""
        }

        btn_search.setOnClickListener {
            val search : EditText = findViewById(R.id.search)
            if (search.text.toString().isEmpty()) {

            } else {
                searchBar()
            }
        }

        val btn_reset: Button = findViewById(R.id.button_reset)
        btn_reset.setOnClickListener {
            initData()
        }

    }

    private fun searchBar() {
        val search : EditText = findViewById(R.id.search)
        val db : FirebaseFirestore= FirebaseFirestore.getInstance()
        val rvUsers= findViewById<RecyclerView>(R.id.recyclerViewUsersList)
        db.collection("Users")
                .whereEqualTo("Name",
search.text.toString()).get().addOnSuccessListener { documents ->
                    val mData: ArrayList<UserModel> = ArrayList()
                    for (document in documents) {
                        val taskItem: UserModel =
document.toObject(UserModel::class.java)
                        mData.add(taskItem)
                    }
                mAdapter = UserAdapter(mData)
                rvUsers.adapter = mAdapter
                mAdapter!!.setOnItemClickListener(this)
                mAdapter!!.setOnItemLongClickListener(this)
                mAdapter!!.notifyDataSetChanged()
            }
    }

    private fun hideKeyboard(view: View) {
        val inputMethodManager =
getSystemService(INPUT_METHOD_SERVICE) as InputMethodManager
        inputMethodManager.hideSoftInputFromWindow(view.windowToken, 0)
    }

    private fun openActivity1() {
        val intent = Intent(this, AdminHomeActivity::class.java)
        startActivity(intent)
        finish()
    }

    private fun initView() {
        val rvUsers= findViewById<RecyclerView>(R.id.recyclerViewUsersList)
        rvUsers.layoutManager = LinearLayoutManager(this)
        rvUsers.itemAnimator = DefaultItemAnimator()
    }

    private fun initData() {
        val db : FirebaseFirestore = FirebaseFirestore.getInstance()
        val collection: CollectionReference = db.collection("Users")
        val query : Query = collection
        val rvUsers = findViewById<RecyclerView>(R.id.recyclerViewUsersList)
        query.get().addOnCompleteListener { task ->
            if (task.isSuccessful) {
                val mData: ArrayList<UserModel> = ArrayList()
                for (document in task.result) {
                    val taskItem: UserModel =
document.toObject(UserModel::class.java)
                    mData.add(taskItem)
                }
                val mAdapter = UserAdapter(mData)
                rvUsers.adapter = mAdapter
                mAdapter.setOnItemClickListener(this)
                mAdapter.setOnItemLongClickListener(this)
            }
        }

    }

    private fun showMenu() {
        val button : ImageButton = findViewById(R.id.btn_menu)
        button.setOnClickListener {
            val inflater = LayoutInflater.from(this)
            val popupView: View = inflater.inflate(R.layout.menu, null)
            val focusable = true
            val popupWindow = PopupWindow(popupView, 500, 1000, focusable)
            button.x.toInt()
            button.y.toInt()
            popupWindow.showAsDropDown(button)
            popupWindow.contentView.findViewById<TextView>(R.id.add_text).text
= "ADD USER"


popupWindow.contentView.findViewById<LinearLayout>(R.id.advancedSearch)
.visibility = View.GONE

popupWindow.contentView.findViewById<LinearLayout>(R.id.add).setOnClickLi
stener {
                val intent = Intent(this, AddUserActivity::class.java)
                startActivity(intent)
            }


popupWindow.contentView.findViewById<LinearLayout>(R.id.sortBy).setOnClic
kListener {

            }

            popupView.setOnClickListener { popupWindow.dismiss() }
            popupView.setOnTouchListener { v, event ->
                popupWindow.dismiss()
                true
            }

        }
    }

    override fun onBackPressed() {
        val intent = Intent(this, AdminHomeActivity::class.java)
        startActivity(intent)
        finish()
        return
    }

    override fun onItemClick(view: View?, position: Int) {
        val db: FirebaseFirestore = FirebaseFirestore.getInstance()
        val collection: CollectionReference = db.collection("Users")
        val query: Query = collection
        query.get().addOnCompleteListener { task ->
            if (task.isSuccessful) {
                val mData: ArrayList<UserModel> = ArrayList()
                for (document in task.result) {
                   val taskItem: UserModel =
document.toObject(UserModel::class.java)
                   mData.add(taskItem)
                }

                val bean: UserModel = mData[position - 1]

                val name = bean.Name
                val mail = bean.Email

                val inflater = LayoutInflater.from(this)
                val popupView: View = inflater.inflate(R.layout.pop_window_users,
null)
                val focusable = true

                val popName : TextView = popupView.findViewById(R.id.tool_title)
                val popupWindow = PopupWindow(popupView, 700, 500, focusable)

                popName.isAllCaps=false
                popName.text = name
                popupWindow.showAtLocation(view, Gravity.CENTER, 0, 0)

                val btn_menu : ImageButton =
popupWindow.contentView.findViewById(R.id.btn_menu)
                btn_menu.visibility = View.GONE

                val btn_back : ImageButton =
popupWindow.contentView.findViewById(R.id.btn_back)
                btn_back.visibility = View.GONE

                val user = Firebase.auth.currentUser
                val email= user.email
                val ref =
FirebaseStorage.getInstance().reference.child("profile/$mail")
                val profile : CardView =
popupWindow.contentView.findViewById(R.id.card_profile)
                val imageView =
profile.findViewById<ImageView>(R.id.profilePicture)
                imageView.drawable
                ref.downloadUrl.addOnSuccessListener {Uri->
                   val imageURL = Uri.toString()
                   Glide.with(this)
                         .load(imageURL)
                         .into(imageView)

                }


popupWindow.contentView.findViewById<Button>(R.id.pop_delete_user).setOn
```

```
ClickListener {

                val user = Firebase.auth.currentUser
                val currentUser= user.email
                val df2= db.collection("Users").document(currentUser.toString())
                df2.get().addOnSuccessListener { documentSnapshot ->
                    if (currentUser.toString() == mail.toString()) {
                        val builder = AlertDialog.Builder(this)
                        builder.setTitle("Error")
                        builder.setMessage("Can not delete your account from
here. Please contact with an administrator.")
                        builder.setPositiveButton("Accept", null)
                        val dialog: AlertDialog = builder.create()
                        dialog.show()

                    } else {
                        val inflater3 = LayoutInflater.from(this)
                        val popupView3: View = inflater3.inflate(
                            R.layout.pop_window_pin,
                            null
                        )
                        val focusable3 = true
                        val popupWindow3 = PopupWindow(popupView3, 900,
400, focusable3)
                        popupWindow3.showAtLocation(view, Gravity.CENTER, 0,
0)
                        val txtpin =
popupWindow3.contentView.findViewById<TextView>(R.id.inputPIN)
                        val delete : Button =
popupWindow3.contentView.findViewById(R.id.pop_pin)
                        val df3= db.collection("Users").document(mail.toString())
                            delete.setOnClickListener {
df3.get().addOnSuccessListener { documentSnapshot ->
                                val builder = AlertDialog.Builder(this)
                                builder.setTitle("CAUTION")
                                builder.setMessage("Are you sure you want to
continue?")
                                builder.setPositiveButton("Accept") { dialog, which ->
                                    if (documentSnapshot.getString("PIN") ==
txtpin.text.toString()) {
                                        df3.delete()
                                        val intent = Intent(this, UserActivity::class.java)
                                        startActivity(intent)
                                        finish()
                                    } else {
                                        val builder1 = AlertDialog.Builder(this)
                                        builder1.setTitle("Error")
                                        builder1.setMessage("Incorrect PIN, please try
again.")

                                        builder1.setPositiveButton("Accept", null)
                                        val dialog1: AlertDialog = builder1.create()
                                        dialog1.show()
                                    }
                                }
                                builder.setNegativeButton("Cancel", null)
                                val dialog: AlertDialog = builder.create()
                                dialog.show()
                            }
                        }
                    }
                }
            }

popupWindow.contentView.findViewById<Button>(R.id.pop_edit_user).setOnCli
ckListener {
                val intent = Intent(this, EditUserActivity::class.java)
                startActivity(intent)
            }

            val SHARED_PREF_USER = "user";
            val KEY_USER = "key_user";
            val sp = getSharedPreferences(SHARED_PREF_USER,
MODE_PRIVATE)
            val editor = sp.edit()
            editor.putString(KEY_USER, mail)
            editor.apply()

            val SHARED_PREF_JOB = "JOB";
            val KEY_JOB = "key_JOB";
            val sp2 = getSharedPreferences(SHARED_PREF_JOB,
MODE_PRIVATE)
            val editor2 = sp2.edit()
            editor2.putString(KEY_JOB, mail)
            editor2.apply()


            popupView.setOnTouchListener { v, event ->
                popupWindow.dismiss()
                true
            }
        }
    }
  }

  override fun onItemLongClick(view: View?, postion: Int) {
      TODO("Not yet implemented")
  }

}
```

A.22. Welcome activity

```
class WelcomeActivity : AppCompatActivity() {

   override fun onCreate(savedInstanceState: Bundle?) {

      super.onCreate(savedInstanceState)
      setContentView(R.layout.activity_welcome)

      val analytics: FirebaseAnalytics = FirebaseAnalytics.getInstance(this)
      val bundle = Bundle()
      bundle.putString("message", "integration of firebase complete")
      analytics.logEvent("InitScreen", bundle)

      val register = findViewById<View>(R.id.home_register) as Button
      register.setOnClickListener {openRegister()}

      val editText1 = findViewById<TextView>(R.id.inputEmailLogin)
      val editText2 = findViewById<TextView>(R.id.inputPasswordLogin)

      editText1.onFocusChangeListener = View.OnFocusChangeListener { v,
hasFocus ->
         if (!hasFocus) {
            editText1.setHint(R.string.EmailHint)
            hideKeyboard(v)
         } else editText1.hint = ""
      }

      editText2.onFocusChangeListener = View.OnFocusChangeListener { f,
hasFocus ->
         if (!hasFocus) {
            editText2.setHint(R.string.PasswordHint)
            hideKeyboard(f)
         } else editText2.hint = ""
      }

      displayEmail()
      displayPassword()

      val login : Button = findViewById(R.id.btn_login)
      val remember : CheckBox = findViewById(R.id.remember)
      login.setOnClickListener {
         if (remember.isChecked) {
            saveEmail()
            savePassword()
            setup()
         } else {
            setup()
         }
      }

      sendEmail()

   }

   private fun sendEmail() {
      val forgot : TextView = findViewById(R.id.forgotPassword)
      forgot.setOnClickListener {

         val inflater = LayoutInflater.from(this)
         val popupView: View = inflater.inflate(R.layout.pop_window_forgot, null)
         val focusable= true
         val popupWindow = PopupWindow(popupView, 900, 400, focusable)
         popupWindow.showAtLocation(popupView, Gravity.CENTER, 0, 0)

         val confirm =
popupWindow.contentView.findViewById<Button>(R.id.pop_forgot)

         val email : EditText =
popupWindow.contentView.findViewById(R.id.inputEmail)

         email.onFocusChangeListener = View.OnFocusChangeListener { v,
hasFocus ->
            if (!hasFocus) {
               email.setHint(R.string.EmailHint)
               hideKeyboard(v)
            } else email.hint = ""
         }

         confirm.setOnClickListener {
            val auth = FirebaseAuth.getInstance()
            auth.sendPasswordResetEmail(email.toString())
                  .addOnCompleteListener { task ->
                     if (task.isSuccessful) {
                        Toast.makeText(applicationContext, "Check your email",
Toast.LENGTH_SHORT).show()
                        popupWindow.dismiss()
                     } else {
                        Toast.makeText(this@WelcomeActivity,
task.exception!!.message, Toast.LENGTH_SHORT).show()
                     }
                  }
         }
      }
   }

   private fun displayEmail() {
      val SHARED_PREF_EMAIL = "email";
      val KEY_EMAIL = "key_email";
      val sp = getSharedPreferences(SHARED_PREF_EMAIL,
MODE_PRIVATE)
      val email = sp.getString(KEY_EMAIL, null)
      val editText1 = findViewById<TextView>(R.id.inputEmailLogin)
      if (email != null) {
         editText1.text = "$email"
      }
   }

   private fun displayPassword() {
      val SHARED_PREF_PASSWORD = "password";
      val KEY_PASSWORD = "key_pasword";
      val sp = getSharedPreferences(SHARED_PREF_PASSWORD,
MODE_PRIVATE)
      val password = sp.getString(KEY_PASSWORD, null)
      val editText2 = findViewById<TextView>(R.id.inputPasswordLogin)
      if (password != null) {
         editText2.text = "$password"
      }
   }

   private fun saveEmail() {
      val SHARED_PREF_EMAIL = "email";
      val KEY_EMAIL = "key_email";
      val editText1 = findViewById<TextView>(R.id.inputEmailLogin)
      val email: String = editText1.text.toString()

      val sp = getSharedPreferences(SHARED_PREF_EMAIL,
MODE_PRIVATE)
      val editor = sp.edit()
      editor.putString(KEY_EMAIL, email)
      editor.apply()

      editText1.text = ""
      displayEmail()
   }

   private fun savePassword() {
      val SHARED_PREF_PASSWORD = "password";
      val KEY_PASSWORD = "key_pasword";
      val editText2 = findViewById<TextView>(R.id.inputPasswordLogin)
      val password: String = editText2.text.toString()

      val sp = getSharedPreferences(SHARED_PREF_PASSWORD,
MODE_PRIVATE)
      val editor = sp.edit()
      editor.putString(KEY_PASSWORD, password)
      editor.apply()
      editText2.text = ""

      displayPassword()
   }

   private fun openRegister() {
      val intent = Intent(this, RegisterActivity::class.java)
      startActivity(intent)
   }

   private fun setup() {
      val password = findViewById<EditText>(R.id.inputPasswordLogin)
      val email = findViewById<EditText>(R.id.inputEmailLogin)
      val box2 : TextInputLayout = findViewById(R.id.boxPassword)
      val box : TextInputLayout = findViewById(R.id.boxEmail)
          if (checkPassword() and checkEmail()) {

FirebaseAuth.getInstance().signInWithEmailAndPassword(email.text.toString(),
                    password.text.toString()).addOnCompleteListener { task ->
                  if (task.isSuccessful) {
                    val db = FirebaseFirestore.getInstance()
                    val df: DocumentReference =
db.collection("Users").document(email.text.toString())
                    df.get().addOnSuccessListener { documentSnapshot ->
                        val user = FirebaseAuth.getInstance().currentUser
                        if (user.isEmailVerified &&
documentSnapshot.getString("Permission") == "Yes") {
                            checkUserAccessLevel()
                            val SHARED_PREF_PASSWORD = "password"
                            val KEY_PASSWORD = "key_password"
                            val sp =
getSharedPreferences(SHARED_PREF_PASSWORD, MODE_PRIVATE)
                            val editor = sp.edit()
                            editor.putString(KEY_PASSWORD,
password.text.toString())
                            editor.apply()
                            box.error = null
                            box2.error = null
                        } else if (user.isEmailVerified &&
documentSnapshot.getString("Permission") == "No") {
                            showPermissionError()
                            box.error = null
                            box2.error = null
                        } else {

FirebaseAuth.getInstance().signInWithEmailAndPassword(email.text.toString(),
                                  password.text.toString()).addOnCompleteListener {
task ->
                                val builder = AlertDialog.Builder(this)
                                builder.setTitle("Error")
                                builder.setMessage("You are not verified. Do you
want us to send you the verification email again? ")
                                builder.setPositiveButton("Accept") { dialog, id ->
                                    val user = FirebaseAuth.getInstance().currentUser
                                    user.sendEmailVerification()
                                }
                                builder.setNegativeButton("Cancel", null)
                                val dialog: AlertDialog = builder.create()
                                dialog.show()
                            }
                            box.error = null
                            box2.error = null
                        }
                    }
                } else {
                    validPassword()
                    validEmail()
                }
            }.addOnFailureListener {
                val builder = AlertDialog.Builder(this)
                builder.setTitle("Error")
                builder.setMessage("Detected some problems with your
account.")
                builder.setPositiveButton("Accept", null)
                val dialog: AlertDialog = builder.create()
                dialog.show()
            }
        }

    }

    private fun validPassword():Boolean {
        val box2 : TextInputLayout = findViewById(R.id.boxPassword)
        box2.error = "Error: check your password"
        return false
    }

    private fun validEmail():Boolean {
        val box : TextInputLayout = findViewById(R.id.boxEmail)
        box.error = "Error: check your email"
        return false
    }

    private fun checkPassword():Boolean {
        val password = findViewById<EditText>(R.id.inputPasswordLogin)
        val box2 : TextInputLayout = findViewById(R.id.boxPassword)
        if (password.text.isEmpty()) {
            box2.error = "Please enter password"
            return false
        } else {
            return true
        }
    }

    private fun checkEmail(): Boolean {
        val email: EditText = findViewById(R.id.inputEmailLogin)
        val box : TextInputLayout = findViewById(R.id.boxEmail)
        if (email.text.isEmpty()) {
            box.error = "Please enter email"
            return false
        } else {
            return true
        }
    }

    private fun checkUserAccessLevel() {
        val email = findViewById<EditText>(R.id.inputEmailLogin)
        val db = FirebaseFirestore.getInstance()
        val df: DocumentReference =
db.collection("Users").document(email.text.toString())
        df.get().addOnSuccessListener { documentSnapshot ->
            if (documentSnapshot.getString("Job") == "Administrator" &&
documentSnapshot.getString("Permission") == "Yes"){
                val job = documentSnapshot.getString("Job")
                val name = documentSnapshot.getString("Name")
                showAdmin(email.text.toString(), job.toString(), name.toString())
            }
            if (documentSnapshot.getString("Job") == "CAMO engineer" &&
documentSnapshot.getString("Permission") == "Yes"){
                val job = documentSnapshot.getString("Job")
                val name = documentSnapshot.getString("Name")
                showEngineer(email.text.toString(), job.toString(), name.toString())
            }
            if (documentSnapshot.getString("Job") == "MRO engineer" &&
documentSnapshot.getString("Permission") == "Yes"){
                val job = documentSnapshot.getString("Job")
                val name = documentSnapshot.getString("Name")
                showEngineer(email.text.toString(), job.toString(), name.toString())
            }
            if (documentSnapshot.getString("Job") == "Total access engineer"
&&  documentSnapshot.getString("Permission") == "Yes"){
                val job = documentSnapshot.getString("Job")
                val name = documentSnapshot.getString("Name")
                showEngineer(email.text.toString(), job.toString(), name.toString())
            }
            if (documentSnapshot.getString("Job") == "Airplane maintenance
technician" &&  documentSnapshot.getString("Permission") == "Yes"){
                val job = documentSnapshot.getString("Job")
                val name = documentSnapshot.getString("Name")
                showAMT(email.text.toString(), job.toString(), name.toString())
            }
            if (documentSnapshot.getString("Job") == "Cabin crew" &&
documentSnapshot.getString("Permission") == "Yes"){
                val job = documentSnapshot.getString("Job")
                val name = documentSnapshot.getString("Name")
                showCrew(email.text.toString(), job.toString(), name.toString())
            }
            if (documentSnapshot.getString("Permission") == "No") {
                showPermissionError()
            }
        }
    }

    private fun showPermissionError() {
        val builder = AlertDialog.Builder(this)
        builder.setTitle("Error")
        builder.setMessage("You have no permission to access this app. In
case you have registered before, " +
                "please wait for the administrator to give you access. If there is
an access problem, " +
                "please contact this email: airdocs.tests@gmail.com")
        builder.setPositiveButton("Accept", null)
        val dialog: AlertDialog = builder.create()
        dialog.show()
    }

    private fun hideKeyboard(view: View) {
        val inputMethodManager =
getSystemService(INPUT_METHOD_SERVICE) as InputMethodManager
        inputMethodManager.hideSoftInputFromWindow(view.windowToken, 0)
    }

    private fun showAdmin(email: String, job: String, name: String) {
        val homeIntent = Intent(this, AdminHomeActivity::class.java).apply {
            putExtra("email", email)
            putExtra("job", job)
            putExtra("name", name)
        }
        startActivity(homeIntent)
    }

    private fun showEngineer(email: String, job: String, name: String) {
        val homeIntent = Intent(this, EngineerHomeActivity::class.java).apply {
            putExtra("email", email)
            putExtra("job", job)
            putExtra("name", name)
        }
        startActivity(homeIntent)
    }
```

```
    private fun showAMT(email: String, job: String, name: String) {
        val homeIntent = Intent(this, AMTHomeActivity::class.java).apply {
            putExtra("email", email)
            putExtra("job", job)
            putExtra("name", name)
        }
        startActivity(homeIntent)
    }

    private fun showCrew(email: String, job: String, name: String) {
        val homeIntent = Intent(this, CrewHomeActivity::class.java).apply {
            putExtra("email", email)
            putExtra("job", job)
            putExtra("name", name)
        }
        startActivity(homeIntent)
    }

    override fun onBackPressed() {
        val alertDialog = AlertDialog.Builder(this)
        alertDialog.setTitle("Exit")
        alertDialog.setMessage("Are you sure you want exit the app?")
        alertDialog.setPositiveButton("No") { dialog, which -> dialog.cancel()}
        alertDialog.setNegativeButton("Yes") { dialog, which ->
            finish()
            android.os.Process.killProcess(android.os.Process.myPid()) }
        alertDialog.show()
    }

}
```