MASTER THESIS

**TITLE:** Full REST Native APP Using SDK Framework Adopting New Tendencies To Solve Face-To-Face Procedures “e-Paperwork”

**MASTER DEGREE:** Master's Degree in Applied Telecommunications and Engineering Management (MASTEAM)

**AUTHOR:** J. Damián Roth Velasco

**DIRECTOR:** Miguel Valero

**SUPERVISOR:** Fréderic Casanovas

**DATE:** Thursday, 31 August 2017
Overview

This document holds description, scope and prototype data of a mobile application, developed with a Software Development Kit (SDK) Framework in order to have a native application for Android, Windows Phone and IOS simultaneously, where the prototype data will show the structure of the programed application and interaction with different frameworks.

The purpose of this mobile application, is to facilitate bureaucratic activities within a company, reducing or eradicating face-to-face paperwork and long administrative processes by taking them all to a mobile device, for clients or employee. In this way, the target company may reduce time and cost at every process in addition to increase effectiveness.

These aim is done by applying methods for securely exchange data over Internet with the mobile application, been these different methods the core of the developed design and the key for any company to trust the data sent by the client.

Mainly, the method used to validate data and files in for this application is the hash function, applied in an easy way for the user to fully understand how it works using compatible frameworks.

The Frameworks used, simplifies development of native mobile application assisting entrepreneurs which have no specific programing skills. Therefore one of the intentions presented, is to describe all the new tendencies and technologies that have been used to create this application which have rare information, support and very few developments at the field.

Furthermore, to document this new tendencies and their interaction can be valuable to increase framework versatility.
INDEX

CHAPTER 1.  INTRODUCTION.................................................................................. 1
  1.1  Problem Description .................................................................................. 1
  1.2  Current solutions or partial solutions for the problem............................... 2
  1.3  No face-to-face procedures ....................................................................... 2
  1.4  Native APP and platform options............................................................... 3

CHAPTER 2.  PROVIDING TRUST ................................................................. 7
  2.1  Data validation method ............................................................................. 7
  2.2  Organized and structured processes ....................................................... 10

CHAPTER 3.  ENVIRONMENT .......................................................................... 14
  3.1  Backend.................................................................................................... 14
    3.1.1  VPS ...................................................................................................... 14
    3.1.2  Django .................................................................................................. 15
    3.1.3  PostgreSQL .......................................................................................... 16
  3.2  Frontend ..................................................................................................... 18
    3.2.1  Ionic .................................................................................................... 19
    3.2.2  Angular ................................................................................................. 20
    3.2.3  NodeJS .................................................................................................. 21
    3.2.4  Cordova ................................................................................................. 22
  3.3  Setup Environment and Emulator ............................................................. 23

CHAPTER 4.  APP DESIGN ............................................................................ 25
  4.1  Database Design and Relations ............................................................... 25
  4.2  App Design and Views ............................................................................. 30

CHAPTER 5.  HOW IT WORKS ...................................................................... 36
  5.1  Certificate request example ..................................................................... 36

CHAPTER 6.  BUSINESS PLAN .................................................................... 44
  6.1  Executive Summary .................................................................................. 44
  6.2  Company Description .............................................................................. 45
  6.3  Products & Services .................................................................................. 46
  6.4  Market research ....................................................................................... 47
6.5 Marketing Plan

CHAPTER 7. REFERENCES

CHAPTER 8. GLOSSARY

ANNEXES

Annex A
- Company Description Worksheet .......................................................... 58
- Product & Service Description Worksheet ................................................ 59
- SWOT Analysis Worksheet ...................................................................... 60

Annex B
- Backend Code ......................................................................................... 61
- Frontend Code ......................................................................................... 75

Figure 1: Interest over Time Ionic, Phonegap and Xamarin (stackshare.io, 2017) ........ 6
Figure 2: pem Package commit (Josef Fröhle, 2017) ........................................ 7
Figure 3: Type Script MD5 package commit (Takach, 2016) ................................. 7
Figure 4: Merkle Tree (Göthberg, 2012). ......................................................... 9
Figure 5: Block chain structure (Wander, 2013) ................................................ 10
Figure 6: Django graphical interphase ............................................................. 11
Figure 7: DigitalOcean droplet with Ubuntu 16.04 ............................................. 12
Figure 8a: Login  Figure 8b: Available paperwork ............................................ 13
Figure 9a: Paperwork steps  Figure 9b: Step Form ........................................... 13
Figure 10: Cordova Application Architecture (Cordova, 2015) ............................ 22
Figure 11: Django Project Structure ............................................................... 25
Figure 12: Django Default Tables Relations .................................................... 28
Figure 13: e-Paperwork Tables Relations ......................................................... 29
Figure 14a: Form hash in app_userprocedure table ......................................... 30
Figure 15b: File hash in app_userfilesuploads table ....................................... 30
Figure 16: Ionic Project Structure ................................................................ 30
Set of Figures 17: View Design ..................................................................... 34
Figure 18: MD5 generator (OnlineMD5, 2012 - 2017) ....................................... 43
Figure 19: Interest by region according chart4 ................................................. 49

Table 1: Comparison Hybrid vs Native App (Alexseyenko, 2017) ......................... 4
Table 2: List of Top App Development Frameworks (KS, 2017) & (Jscrambler, 2017) ........................................................................................................ 4
Table 3: Roll created for e-Paperwork app ...................................................... 17
Table 4: Database created for e-Paperwork app .............................................. 17
Table 5: AngularJS / Angular 2 (Olo Technic, 2016) ......................................... 20
Table 6: Models for e-Paperwork ................................................................. 26
Table 7: App Page Description .................................................................... 35
Table 8: Enterprise Distribution by Size (Dirección General de Industria y de la Pequeña y Mediana Empresa, 2016) ......................................................... 50
Chart 1: Google play store data ................................................................. 48
Chart 2: Competitors comparison ............................................................... 48
Chart 3: Global Trend with English Search (Google, 2017) ....................... 49
Chart 4: Global Interest by Topic Search Spanish vs English (Google, 2017) . 49
Chart 5: Spain Interest over Time ............................................................... 50
Chart 6: Spain Enterprise Population Evolution (2006-2016) ..................... 50
CHAPTER 1. INTRODUCTION

1.1 Problem Description

Every public or private entity that provides a service may actually incur on bureaucracy, and all clients and administrative employee have to ask, send or provide data for a certain procedure on many administrative levels.

Because most organizations have hierarchical structure, bureaucracy also adopts and becomes a hierarchical system, been in this way a problem that commonly includes inefficiency and disorder. Therefore it is most certain that when a given organization is larger, the bureaucracy gets also larger as the amount of operations that every person inside that procedure has to perform, causing even a larger disorder.

When a procedure is very common or it is condensed to be done in a short period of time, long waiting lines can also become a problem. Typically it is needed all morning or all afternoon to attend or perform any organization´s paperwork, often just to do a part of it.

Besides the waiting lines and the time spent on them, not all paperwork that one party needs to complete is nearby their home or office. Consequently the interested person needs to move large distances to finish this tedious procedure at the respective organization’s office, so this is also a waste of time and money for them.

Furthermore, everyone or most of the people that needs to do some important paperwork, like renew their identification, passport, residence or even their university enrolment, are at work fulfilling their labour hours. This means that this person would need to get out of work just to comply some paperwork with waiting lines across the city. This is maybe because the entity on which the paperwork that needs to be done, also admits people just inside working labour hours; let’s say between 8:00 and 18:00 but not on Saturday or Sunday.

All these drawbacks end up on a waste of time and money not just for the interested party, but for the entities, organizations or enterprises that offers the service. It is certain that no one likes to waist their precious time locked in a room filled with people on an uncomfortable chair or even worse, on their feet waiting for their turn.

At the present time, there are still face-to-face paperwork or procedures due to the lack of confidence on digital documents and their validity, mainly because there are too many ways of making digital frauds, and the existing ways to avoid this frauds are complicated and difficult to assimilate for common people.

There can be found all kind of face-to-face paperwork everywhere, and all the people that actually goes to site are probably used to make these lines. This is
actually because most of those entities do not offer their own trustful platform to avoid that.

1.2 Current solutions or partial solutions for the problem

There have been several attempts to wildly solve the fear of using digital processes and leave behind the tons of papers created every year for different kind of paperwork. The most robust and well known is the solution that gives Adobe Acrobat to sign PDF documents, that besides creating e-signature, it provides digital certificates and time stamps to validate the document.

Vieira is another good yet expensive solution for different platforms that provides authentication, e-signature, digital custody and verification that can be used to simplify the use of digital certificates giving inclusion as a service. In the same way that some other solutions can be used with a fee like Viafirma, other solutions with similar benefits but with no cost and maybe with less features can be found like eCoFirma provided by the ministry of industry, energy and tourism of the Spain government.

All current solutions are focused on a general point of view for these problems, providing a tool to accredit electronically and without doubt the identity of a registered user and to sign electronic documents giving them legal validity as anyone can have with a physical signed document.

The disadvantage with these solutions, as mentioned before, is that where developed for general purpose or for people and entities in general, therefore they do not offer a personalized solutions nor a personalized service to solve different issues and in some cases the solution is restricted to one document at a time and not a procedure.

1.3 No face-to-face procedures

By using general solutions for all the processes, that company or entity would have to apply that specific software for each one of the process using the tool at every step, this may be uncomfortable and unsystematic for the user.

The solution projected in this master thesis, focuses on solving just one entity’s problem type at a time, offering a mobile application as a service (MAaaS), having in mind that the common problem for most entities in general is bureaucracy.

This application (e-Paperwork) is thought and designed for the “Universitat Politècnica de Catalunya” (UPC). Nevertheless it can be easily redesigned for any public or private entity like banks, government offices, law firms, or any other business that needs to exchange important documents. Certainly e-Paperwork can be suitable for any process that commonly handles documents which are highly susceptible to modifications or sends physical documents that may travel long distances, where they can be misplaced or delayed.
Because e-Paperwork is an application as a service, it is presented as a prototype, with limited coverage for all possible procedures the UPC may have. The final solution or the end product, will need coordination with the UPC or the interested entity to establish the flux, steps and details for the process needed to be covered by e-Paperwork.

To avoid face-to-face procedures, it will evolve depending the needs of the client, by adding or deleting steps, forms, features, users, etc. In this way, with just one application, it can be possible to have several or even all the paperwork that is perform within a year, compressed, organized and controlled in a single mobile application.

As mentioned before, all kind of paperwork can be included at e-Paperwork, and not only the ones that needs signed documents, important documents to avoid modifications or plain personal documentation. e-Paperwork is also suitable for paperwork that consist only of filling forms, forms that need validation control and data which needs modification control from the source.

By using new and versatile frameworks like Ionic, e-Paperwork is easy to redesign orienting the structure of frontend and backend to different environments, giving it the ability to change client targets, add new functions and facilities as the client request. This is why e-Paperwork is a MAaaS reducing time and costs at both sides: business and client.

### 1.4 Native APP and platform options

When a great idea with the latest tendencies is needed to become a mobile application project, the hardest decision is how to start building it. With so many options in platforms, frameworks and how fast they are evolving, the considerations must involve time, cost, development capabilities, user experience and some other technical factors like if it meet multi-platform functionalities, or if it is for Hybrid HTML or just Native development.

Having in mind the following definitions (Lozano, 2017):

- “Hybrid HTML frameworks use web technologies JavaScript/HTML/CSS and are interpreted and displayed within the device’s native web view as a standalone application.”

- “Cross-Platform Native frameworks use one programming language across multiple platforms to develop and compile straight into an intermediate language.”

The development of e-Paperwork involved a lot of comparison and planning to find the best options to start the process, knowing that if a better option came up it would be a waste of time to change course.

First of all, the way to create the app must be clear when choosing to develop Native App or a Web App. A Native App is developed to work in one particular
mobile device with specific characteristics, and are written in a specific programming language for a specific platform.

A Web App are basically internet-enabled apps where it is not necessary to download the app and it works directly through a Web browser. Furthermore a hybrid app is possible because of third party tools that facilitates the interaction of the web view and the native platforms like Apache Cordova. When a hybrid app is built, it will be compiled transforming the web application into a native app (Alexseyenko, 2017).

There are many considerations to think through, but it depends most of all on the type of project and the type of client to choose an option. Table 1 shows the basic concepts to have in mind why e-Paperwork is developed with a hybrid mobile framework.

**Table 1**: Comparison Hybrid vs Native App (Alexseyenko, 2017).

<table>
<thead>
<tr>
<th>Hybrid App</th>
<th>Native App</th>
</tr>
</thead>
<tbody>
<tr>
<td>Developed using HTML, CSS and JavaScript</td>
<td>Developed in platform specific language, Objective-C or Swift for iOS, Java for Android, etc.</td>
</tr>
<tr>
<td>Write Once, Run Anywhere</td>
<td>Separate code for each platform</td>
</tr>
<tr>
<td>Medium performance comparable to Native apps</td>
<td>Fastest and most responsive experience to users</td>
</tr>
<tr>
<td>Save Time and Money</td>
<td>Higher investment of time, talent and resources</td>
</tr>
<tr>
<td>Faster development cycle</td>
<td>Higher cost and development time</td>
</tr>
<tr>
<td>E.g. Baskin Robbin, Sworkit, Untappd</td>
<td>E.g. PayPal, Gmail</td>
</tr>
</tbody>
</table>

As mentioned before, there are so many frameworks with such different features that drives to not consider just hybrid frameworks, the tools offered, pre-built functionalities and templates can help a lot depending the complexity of the project. Table 2 shows a list of some top frameworks available with pros and cons of each one, most of all taking in consideration the basic options that Ionic has.

**Table 2**: List of Top App Development Frameworks (KS, 2017) & (Jscrambler, 2017).

<table>
<thead>
<tr>
<th>Platform frameworks</th>
<th>Type</th>
<th>Pros</th>
<th>Cons</th>
</tr>
</thead>
</table>
| Xamarin             | Cross-Platform Native frameworks | • Uses C# on devices  
• Native Controls and performance  
• C# libraries | • Not use of open-source libraries for iOS and Android  
• Free version is seriously limited |
| Appcelerator        | Cross-Platform Native frameworks | • Provides tools for development  
• ArrowDB  
• Seamless integration  
• Pre-built connectors | • It has bugs at the platform  
• Poor support  
• Increasing complexity at development when |
<table>
<thead>
<tr>
<th>Framework</th>
<th>Hybrid HTML frameworks</th>
<th>Application Complexity Increases</th>
<th>With Commercial Licensing</th>
<th>Poor Community</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Sencha Touch</strong></td>
<td>Hybrid HTML frameworks</td>
<td>• Built-in native-looking themes&lt;br&gt;• Supports Phonegap/Cordova integration</td>
<td></td>
<td>Poor Community</td>
</tr>
<tr>
<td><strong>Codename One</strong></td>
<td>Cross-Platform Native frameworks</td>
<td>• Supports most of IDEs</td>
<td>Limited visual themes</td>
<td>Limited quantity of plugins for native APIs</td>
</tr>
<tr>
<td><strong>PhoneGap</strong></td>
<td>Hybrid HTML frameworks</td>
<td>• Backend by Adobe&lt;br&gt;• Supports more platforms&lt;br&gt;• Single code base for iOS, Android, Windows Phone, BlackBerry, Firefox OS and more.&lt;br&gt;• in-app integrated payments via App Store</td>
<td>Poor for graphic-intensive apps&lt;br&gt;Suppor</td>
<td>building native applications via the cloud.</td>
</tr>
<tr>
<td><strong>Ionic Framework</strong></td>
<td>Hybrid HTML frameworks</td>
<td>• Also a Cross-Platform Native Mobile Framework&lt;br&gt;• Fast prototyping&lt;br&gt;• Best for Angular&lt;br&gt;• Uses Typescript&lt;br&gt;• Easy to use and learn&lt;br&gt;• Much more&lt;br&gt;• 100% Free with grate community</td>
<td>In-app performance lower than natively develop.&lt;br&gt;AngularJS demands specific skillset for complex apps.&lt;br&gt;ui-router is difficult to manoeuvre.</td>
<td></td>
</tr>
<tr>
<td><strong>Framework7</strong></td>
<td>Hybrid HTML frameworks</td>
<td>• Free and open source&lt;br&gt;• It has page transaction animation&lt;br&gt;• Multiple view support</td>
<td>Just for iOS and Android&lt;br&gt;Poor support</td>
<td></td>
</tr>
<tr>
<td><strong>Mobile Angular UI</strong></td>
<td>Hybrid HTML frameworks</td>
<td>• Uses Bootstrap and Angular&lt;br&gt;• It has fantastick.js and overthrow.js</td>
<td>Not mature enough&lt;br&gt;Below average documentation&lt;br&gt;Really poor community</td>
<td></td>
</tr>
<tr>
<td><strong>Onsen UI</strong></td>
<td>Hybrid HTML frameworks</td>
<td>• It can use Angular, Angular2, React, Vue.js and Meteor&lt;br&gt;Also pure JavaScript capable&lt;br&gt;Free</td>
<td>Too new&lt;br&gt;Support only through the StackOverflow</td>
<td></td>
</tr>
</tbody>
</table>

Most of the Native developing frameworks imply a cost and very good programing skills on a specific programing language. That is why, the use of Ionic seemed more suitable for this project.

A very good tool to choose and make decisions on which frame to use, is Stackshare ([https://stackshare.io](https://stackshare.io)) where the comparisons are made with all kind of frameworks giving a basic description, amount of favourite votes and users to
relatively realize how is the community. It also shows recent news on Hacker News, Reddit and Overflow Stats, GitHub Stats, strongest features vote, pricing, companies using the framework, interest over time and much more.

One of the final facts that drive to choose Ionic is the higher interest of people in the framework, which means more support, more stability due to bug fix because it is open-source, and much more code examples where e-Paperwork can and had use them as base for some functionalities. Figure 1 shows a graphic with the interest over time of Ionic, Phonegap and Xamarin where clearly there is a lot of difference.

![Interest over Time Ionic, Phonegap and Xamarin](image)

**Figure 1**: Interest over Time Ionic, Phonegap and Xamarin (stackshare.io, 2017)

-Numbers represent search interest relative to the highest point on the chart for the given region and time. A value of 100 is the peak popularity for the term. A value of 50 means that the term is half as popular. Likewise a score of 0 means the term was less than 1% as popular as the peak.
CHAPTER 2. PROVIDING TRUST

2.1 Data validation method

As expected, additional security issues besides login and password are considered inside e-Paperwork. Using modern cryptography is the main feature of e-Paperwork, therefore for exchanging information between each user and the corresponding entity, hash functions are generated from each attached document and each filled form by the user to give e-Paperwork a trustful feature.

By the time e-Paperwork project has started, the best choice for exchanging data, was symmetric-key and not asymmetric-key cryptography, because at that moment there was no programming libraries that could handle private keys and certificates.

Figure 2, shows the date of the last commit for “pem” package which contains the latest version for creating private keys and certificates with node.js, which perhaps will provide the basis for the next e-Paperwork version.

Furthermore, when choosing options to use with hash function versions, the processing time to perform a message digest with the latest algorithms like SHA-3 where too long, resulting on heavy performance at e-Paperwork.

This is why the use of Message-Digest Algorithm 5 package (MD5) for TypeScript (Figure 3) became important at the very moment of programming with Ionic, providing a more secure system than its predecessors but without leaving behind the trust of a robust tool.

Figure 2: pem Package commit (Josef Fröhle, 2017).

Figure 3: Type Script MD5 package commit (Takach, 2016).
Although MD5 is not the latest version in message digest algorithms and it was the first available cryptography choice for e-Paperwork, it is very unlikely to have an attack just to break a hash function for documents or filled forms, which in this case are PDF and JSON files. The normal attacks to hash functions are for one word password (no paragraphs) because, unlike encryption or encoding, hashing is a one way process for a text, file or program that returns a fixed character length, and it is impossible to generate the same document, file or program from that very same string. (Singh, 2015)

Using this powerful tool, e-Paperwork can use this 32 hexadecimal symbols string generated to corroborate if the data has been modified since the source sent it, and all this, just by a simple comparison performing an MD5 at destination.

In digital data or digital document exchange, the attackers can modify simple information like bank accounts, names, dates or even scanned documents for their own benefit or just for fun. All this may not be common cases at any university, but it can be very common with financial entities.

Regarding attacks to the e-Paperwork, there is one extra fact why it is better to use symmetric cryptography instead of asymmetric cryptography. The most dangerous attacks to both types of cryptography will be quantum attacks. Although it is unlikely to have such a threat in the near feature, it is useful to mention it and clarify a little bit to have an idea on how secure can be this application.

"On traditional computers, it takes on the order of $2^{128}$ basic operations to get a private key associated with its public key. This number is so massively large that any attack using traditional computers is completely impractical. However, it is known for sure that it would take a sufficiently large quantum computer on the order of only $128^3$ basic quantum operations to be able to break a key using Shor's Algorithm. This might take some time, especially since the first quantum computers are likely to be extremely slow, but it is still very practical.

For symmetric cryptography, quantum attacks will be also possible, but are less dangerous. Using Grover's Algorithm, the number of operations required to attack a symmetric algorithm is square-rooted. For example, finding some data which hashes to a specific SHA-256 hash requires $2^{256}$ basic operations on a traditional computer, but $2^{128}$ basic quantum operations. Both of these are impractically large. Also, since quantum computers will be extremely expensive than traditional computers for decades after they are invented, quantum attacks against symmetric crypto seem unlikely to be especially common or maybe just for large company tests" (bitcoinwiki, 2016).

For some future uses, this design can evolve to be a more robust system. For example, like recent trends which are aiming to use Merkle tree or a Blockchain to secure data, e-Paperwork can easily adopt one of this structures and not only for security but also for marketing purposes. Therefore, this is another advantage of using symmetric cryptography at e-Paperwork.
Using the hash function of each data group sent by the user on each paperwork procedure inside e-Paperwork, it is possible to make a Merkle tree (Figure 4) to efficiently verify and secure large data structures, or maybe a Block chain structure (Figure 5) to also add timestamps, and in a basic use, to add a very resistant-to-modification feature to this app.

![Merkle Tree Diagram](image)

**Figure 4:** Merkle Tree (Göthberg, 2012).

A Merkle tree or a hash tree, is a data structure where every node is tagged with the hash of its child nodes. With this method, a large amount of separated data like the app forms and files, can be linked to a unique hash value that would be the root node. So to demonstrate that a specific form belongs to a tree, it is required a processing data amount proportional to the logarithm of the number of nodes of the tree (Becker, 2008).

On the other hand, a block chain structure can provide more information like shown in Figure 4 and a different way to secure data. Unlike a Merkle tree, the first hash string is hashed together with the second data block, generating another hash string that will be used at the following block. This maintains a continuously growing list of records and making it inherently resistant to modification (Wikipedia, 2017).
Furthermore e-Paperwork is not just about security, it also offers a structured exchange of valid data and documentation that is also interesting for financial and none financial corporations by dealing with bureaucracy.

### 2.2 Organized and structured processes

To provide a service well designed, e-Paperwork is thought to be an application that constantly can evolve and grow, using modifiable forms with a JSON structure, which are stored at the backend using Django.

JSON file example to edit a form:

```json
{
   "form": [
      {
         "name": "Pasos1",
         "description": "",
         "form_fields": [
            {
              "type": "text",
              "id": "first_name",
              "label": "Name",
              "readonly": false,
              "data_source": {},
              "options": []
            }
         ]
      }
   ]
}
```

A registered administrator can add fields at any process form by login in to Django graphical interphase (Figure 6) and modify the JSON file of the desired form, which consists on an easy and basic structure like previous example so it won’t take long to modify if necessary. The modified form will be updated automatically with the desired change after using a valid JSON structure.
Adapting to entities needs is done just by coordinating every step of each procedure has and stablish a well described flux in the application, so entity staff and clients know how to manage them just by following steps and reading every step information.

This way not just forms can be adaptive, but also process; nevertheless to add, delete or modify a process or procedure is also easy if there is not a data correlation to the data base delete action. If user data exists already, and it was added by using a procedure that will be deleted, a modification far more difficult will take place. Therefore a constant technical support can be provided (as a service) to have always an updated application with good growth in process and providing the user, at both sides (client and staff), a comfortable way to work.

As organized as a file drawer, e-Paperwork for this prototype, stores all the user data in a secure cloud Virtual Private Server (VPS) using Ubuntu (Figure 7), where the security and reliability is provided by DigitalOcean, which also offers scalable storage capacity with high availability, storage resizing, aggregable storage blocks and reliable network using encrypted data (DigitalOcean, 2017).
By doing so, all the data stored can be retrieved by using a user ID which can be full name, identification number or an entity’s register number that are related to a database at the VPS.

The design of e-Paperwork also allows to separate users by types. For example at UPC’s e-Paperwork, the user types can be students, professors, UPC staff and administrator, where UPC staff handles, checks and accepts or rejects procedures of paperwork started by one student or professor. In this way, this application works dynamically at both sides allowing a ubiquitous way of work.

Administrators in the other hand, are in charge of managing Django interface, adding, deleting or modifying users which are registered and therefore allowed to use the application. For special determinations, administrators can also be allowed to retrieve personal data and documentation to provide it to UPC staff if necessary.

After registering as a valid user by means of a username and password to log in (Figure 8a), e-Paperwork provides a list of valid procedures or list of paperwork for that specific type of user, which are allowed to perform with a detailed information about them (Figure 8b). Inside each listed item, it provides also the specific steps that each procedure has.
Each step (Figure 9a) contains also a specific form (e.g. Figure 9b) which is to be filled by the logged user where the required data or files can be uploaded, so entity staff can validate them also by logging with the corresponding account. There, UPC staff in this case, can retrieved all the paperwork that have been submitted by a client user. In this way, e-Paperwork becomes interactive providing both sides a step by step working method.

Using this step by step method, the client side will not jump steps, they will provide the requested documentation on a single step and all the process will be finished faster since it is not necessary to perform it on site. Staff side will also have the documentation organized in a digital space, accessible at any time through a smartphone and time used on bureaucratic steps will be reduced.

To fulfil these objectives, this interesting system was developed in two parts as every normal mobile application, Backend and Frontend with the main technology options mentioned before.
CHAPTER 3. ENVIRONMENT

3.1 Backend

It is well known that every complete system has a user interface which is designed to be friendly, and a server side where all the data is stored and managed. For this prototype, a Mobile backend as a service (MBaaS) model was developed to link e-Paperwork to a cloud storage and suitable application programing interface (APIs).

By doing so, it will also provide features such as user management, push notifications and maybe integration with some other functions that can be added in the future. The tools for this side, as mentioned before, are Django with a PostgreSQL data base within a DigitalOcean VPS.

3.1.1 VPS

A virtual private server is a method to divide a physical server in several others. In this way there can be several dedicated VPS, one for each entity stored in one safe place avoiding spending large amounts of money in hardware, security and maintenance.

With this option, each virtual server is able to work under its own Operative System, it can be rebooted individually, it can be connected to a network to interact with others or it can work individually (Wikipedia, 2016).

By using a cloud service provider to store the VPS, the service is well protected against many kind of contingencies like hacker attacks, energy blackout and network rupture. All these benefits could cost too much by installing its own server at entity’s facilities, unless all the structure is already implemented. Then again, by using a cloud service provider, the monthly fee is low and the system will have support 24 hours a day.

In this prototype, all backend is built in this cloud environment with Ubuntu because it has long term support releases, which are updated regularly and has 5 years of life support. The clearly advantage of using recent releases, is because they are compatible with the latest features of applications and different software like Django or Ionic (Reiner, 2017).

The following steps were used to set up the server at DigitalOcean:

- Create Droplet (Ubuntu 16.04.1 x64)
  [https://www.digitalocean.com/](https://www.digitalocean.com/) IP Adress: 104.236.76.253
- Install latest version of Python (Do not delete or replace the default version to avoid deleting useful resources and libraries).
  
  ```
  $ sudo apt-get install python-pip
  ```
- Install a virtual environment in case the server will harbour more e-Paperwork projects.
  
  ```
  $ sudo pip install virtualenv
  ```
• Install dependency packages and libraries for extend Python interpreter in the application (complements to use PostgreSQL).
  libpq, is a set of library functions that allow the client programs to pass queries to PostgreSQL
  
  ```
  $ sudo apt-get install python3-dev libpq-dev
  ```

• Crate a directory for the project and the virtual environment.
  At developer/python/django/epapperwork
  
  ```
  $ virtualenv -p /usr/bin/python3.5 epapperwork
  ```

Until this point, a directory with a virtual environment “epapperwork” has been created. A virtual environment is useful to isolate dependencies, versions and libraries, e.g. if we need to install a different version of Django for a different project.

### 3.1.2 Django

Because this master thesis also intends to explain how someone that is not an expert programmer can make, in an easy and novel way a mobile application, Django is a very good choice since it is Python based which is may be the easiest programming language to learn.

> “With its use of natural language constructs (e.g. paragraph-like layout and indentation) and simple to learn syntax, Python makes understanding program structure and flow significantly easier to learn than other popular languages” (The Django Book, 2017).

This means that Django is a high-level Python framework which can provide an easy design and fast development with a lot of embedded and ready to go commands to help at programming.

Besides being designed to be an easy to use framework, Django has as set of very interesting features like a built-in admin application, authentication framework, PostgreSQL specific features and contribution packages from a large community of developers that can be up to 3400 apps, sites and tools to use as examples. (The Django Book, 2017)

One risk of using an open source framework is the lack of interest of the programming community which can derive in absence of support. Currently Django is actively developed and the 1.11 Long Term Support (LTS) version will have full support until 2020. Besides having support from –the Django Software Foundation- in the US. (The Django Book, 2017).

As Django is based on a Model-View-Controller (MVC) design, e-Paperwork code can be separated in backend and frontend, having also all JSON files, PDF files, images, etc. organized (The Django Book, 2017).

The following steps are used to setup a Django project inside developer/python/django/epapperwork directory:
• Activate the virtual environment created for the specific project (virtual environment prompt is activated (*epaperwork*)).

```bash
$ source venv/bin/activate
```

• Install the latest version of Django inside the virtual environment.

```bash
$ pip install Django==1.11
```

• Install Psycopg2 package as an implementation adapter for PostgreSQL (wrapper for libpq).

```bash
$ pip install psycopg2
```

• At the same directory where the virtual environment folder is created, the Django project must also be created.

```bash
$ django-admin.py startproject epaperwork
```

While working with virtual environments to separate projects, versions or libraries, one must always be sure to have the corresponding virtual environment activated for that project. This will give control and order on what you install on the server. Everything that is installed with the virtual environment activated will be placed at the virtual environment folder that will be next to the project folder.

The folder created by starting a Django project, will contain the basic structure to start working on the application’s backend.

```
[virtualenv]/
[projectname]/
  [projectname]/
   ├── __init__.py
   │    └── settings.py
   │         ├── urls.py
   │         └── wsgi.py
   └── manage.py
```

However, with the `startproject` command, a project template can also be created adding a *template* argument. (Mischback, 2015)

### 3.1.3 PostgreSQL

PostgreSQL is an open source object-relational database (ORD) system, which means that objects, classes and inheritance are supported in the database scheme and at the query language, and where one of the extra advantage for the application is the support for data models with custom data-types and methods. (The PostgreSQL Global Development Group, 2017)

Furthermore, PostgreSQL handles most of data types like integer, numeric, Boolean, char, varchar, date, interval and timestamp. However, for e-Paperwork besides this last three being very important, the storage support for binary large objects, including PDF files, pictures or JSON files is essential (The PostgreSQL Global Development Group, 2017).

It is a common mistake to think in the same way about all relational database management systems. For example there are radical differences between MySQL and PostgreSQL. One reasonable comparison is that both are open source, nevertheless MySQL is distributed under licence which make it more restrictive. PostgreSQL is
distributed under more permissive terms that allows create commercial derivatives without having to pay. (2ndQuadrant Ltd, 2017)

During its development, PostgreSQL has focused in web applications were the main concern is the optimisation of simple queries, which by the way, is suitable for mobile applications like e-Paperwork where the structure is also simple. Although the response time with small databases is quite slow, the same speed is maintained to manage very large databases which is laudable and in fact, it is expected to e-Paperwork get bigger and bigger. (2ndQuadrant Ltd, 2017)

With a good hardware, which is the case with DigitalOcean, this database system is capable to support a greater amount of simultaneous queries and in some cases three times what can support MySQL. (2ndQuadrant Ltd, 2017)

The following steps are used to setup PostgreSQL, the database and the user, so it is suitable to start the app developing and run the server:

- Install PostgreSQL to manage the database in order to avoid using SQLite which is the default database system that comes with Django.
  
  
  
  $ sudo apt-get install postgresql

- To enter postgres command prompt and create the role and the database with the corresponding privileges

  
  
  
  $ sudo -i -u postgres

  postgres@ createuser --interactive --pwprompt

  postgres@ createdb epaperwork

- To grant privileges to the created database for the new roll at psql command prompt:

  
  
  
  Postgres=# grant all privileges on database epaperwork to epaperwork ;

With this commands, PostgreSQL is set and the database is created along with the user as shown at Table 3 and Table 4.

Table 3: Roll created for e-Paperwork app

<table>
<thead>
<tr>
<th>Role name</th>
<th>Attributes</th>
<th>Member of</th>
</tr>
</thead>
<tbody>
<tr>
<td>epaperwork</td>
<td>Superuser, Create role, Create DB, Replication, Bypass RLS</td>
<td>()</td>
</tr>
<tr>
<td>postgres</td>
<td>Superuser</td>
<td>()</td>
</tr>
</tbody>
</table>

Table 4: Database created for e-Paperwork app

<table>
<thead>
<tr>
<th>Name</th>
<th>Owner</th>
<th>Encoding</th>
<th>Collate</th>
<th>Ctype</th>
<th>Access privileges</th>
</tr>
</thead>
<tbody>
<tr>
<td>epaperwork</td>
<td>postgres</td>
<td>UTF8</td>
<td>en_US.UTF-8</td>
<td>en_US.UTF-8</td>
<td>postgres=Cto/postgres</td>
</tr>
</tbody>
</table>

- To link Django and PostgreSQL, some final changes are needed to be done at the file “settings.py” inside the Django project that was created making sure that the engine used is changed from SQLite to PostgreSQL.
Databases = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql_psycopg2',
        'NAME': 'epaperwork',
        'USER': 'epaperwork',
        'PASSWORD': 'epaperworkadmin',
        'HOST': 'localhost',
        'PORT': '',
    }
}

- To start programming the backend application, the following command creates a python package which consist on a specific convention and structure.

$ Python manage.py startapp app

The difference between the project created and the application folder created, is that inside the application folder will be contained the web application, and the project is a set of configurations for it.

### 3.2 Frontend

As there is a backend, there must be a frontend to interact with all users, where all the production of JavaScript, HTML and CSS for Web Applications take place to present the product in a way that these users feel comfortable using it. The objective of the mobile application design is that it must be intuitive and easy to use, having all relevant information in a way that is well organized and comprehensible.

Because users have all kind of devises with different sizes, operative systems and brands, it is very difficult to design a mobile application considering all these variables. If it is expected to have a design for Android, iOS and Windows for example, the time used in developing the frontend can be doubled or tripled depending on the compatibility of the code.

Also it is very challenging to stay up to date with all the developing tendencies, tools and frameworks, especially for web and mobile developing at frontend because within a short period of time, let’s say a few weeks or months, new versions of each framework appears.

e-Paperwork originally started to be developed with Android Studio having the enormous drawback of been a native application just for Android besides been too difficult to program with. After a few failed tries, IONIC framework seemed a better option, because it’s adaptations of programing to work as a native application and of course, it provided a structure far more easy to understand and use for people that has no background on programing but has certain knowledge with HTML5.

This master thesis also provides detailed information on how the application have been made using currently new framework versions and their advantages or disadvantages to achieve a native mobile application, all the interaction between them and in addition, it provides structured information about them which currently is very narrow.
3.2.1 Ionic

To achieve e-Paperwork, there have been considered several platforms and frameworks that could allow native functions like Android Studio, Aperi.io, JQuery and others. However by using Ionic, besides working with a framework that builds native applications, it is a Software Development Kit (SDK) and a framework of frameworks; this means that Ionic uses several frameworks to build an application, for example, the use of Angular in Ionic that helps to create and control interactions, or CORDOVA that allows to access to native functionalities like the camera.

Another advantage of using Ionic framework, is that has become the best option for cross-platform development due to its ability to create at the same time applications that are compatible for Android, iOS and Windows phone. (Beauvois, 2015)

As e-Paperwork started to be developed with Ionic 1, the app design, it’s views and forms where very easy to program as it was most of all HTML and JavaScript. Even so, with Ionic 1 some difficulties where found in the way, like how to display a PDF file in the application or how to hash the JSON forms and files.

Because Ionic is built on top of Angular and uses its directives, their version are strictly linked. Since Angular with its second version had large improvements, especially over mobile app development, it was another hint to migrate e-Paperwork to Ionic 2 besides knowing that Ionic 2 will have more support in a long term.

One of the main difference between Ionic 1 and Ionic 2 is that the first version uses controllers for logic and template for views, and in the second version there are classes instead of controllers and the structure is more organized having every page or component of the application inside its own folder with its own class file and style file (Morony, 2017).

This new framework has quite useful automated tools that creates and setup all files needed. It can automatically generate pages, providers, tabs, pipes, components and directives with simple commands at the Ionic client. The files created by Ionic, already have the basic code needed to start the app. In the case of Ionic 1 it would be on JavaScript and in the case of Ionic 2 it would be TypeScript.

Between the two versions, also the syntax has changed where Ionic 2 uses ES6 syntax but with the same structure, this makes the code look cleaner and shorter. All this code is mostly done at the app folder which is completely separated from the “www” folder which must only contain the code sent to the browser. Unlike with Ionic 1 where all is imported to the index.html file (Morony, 2017).

New versions of Ionic (like Ionic 3), won’t have so drastic transitions, onwards it will be the same basic framework but with updates, and this framework will be referred just as “Ionic”. All this is a good thing for e-Paperwork, because as mentioned before, is hard to keep up with all new releases if they have so much variations (Morony, 2017).
3.2.2 Angular

It is an open source framework supported by Google Inc. that controls and creates interactions and custom directives to use them with Ionic tags. The second Angular version uses TypeScript that is a super set of JavaScript, which gives the possibility to create one page applications and currently is one of the most popular frontend frameworks.

Most of all projects are still using the first Angular version 1.x release even though there is a way to allow projects to run both versions 1.x and 2.0 at the same time. Because the new version is so different, Angular was being left behind, but because of this gradual upgrade, the second version took strength over time (Kuhn, 2015).

Angular 1 or AngularJS has essential differences with Angular 2 and they are not compatible at all just like Ionic. Clearly it is because they are tie to each other, in fact Angular 2 is not the upgrade of Angular1.

Development on Angular 2.0 or later emphasizes removing the unnecessary complexity to the framework with some interesting goals like performance improvements, native app support and server-side rendering (Kuhn, 2015).

While developing an app, the Angular library reads the HTML file that has the attributes of the personalized tags. So it follows the directives and links the input and outputs to a model represented by the JavaScript variables, where this variables can also be retrieved from JSON resources (Angular.io, 2017).

The main purpose of changing to Angular 2 is because this framework is mobile oriented unlike its predecessor. Furthermore, as the latest versions instead of working with “$scope” and “controllers”, they work with zone.js to detect changes and “Components” to collect metadata (Angular.io, 2017).

A component is the combination of HTML template and a component class that controls a portion of screen, it is one of the most important building blocks in the Angular system. In fact, it is a directive with a companion template, and Zone is a mechanism to encapsulate and intercept JavaScript application’s asynchronous activity (Angular.io, 2017).

The languages supported by Angular 2 besides TypeScript are ES5, ES6 and Dart, it interprets the code and translate it in a better and optimized way for JavaScript virtual machines. Some other features that have been changed are described at table 5 (Angular.io, 2017).

<table>
<thead>
<tr>
<th>Angular 1.x</th>
<th>Angular 2.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Modular programing concept</td>
<td>Component based programing</td>
</tr>
<tr>
<td>It is a JS library</td>
<td>It is a framework</td>
</tr>
<tr>
<td>No TypeScript</td>
<td>It uses TypeScript</td>
</tr>
<tr>
<td>It uses controller and $scope</td>
<td>It uses component and directives</td>
</tr>
</tbody>
</table>
Easy setup: add reference of the library

Difficult setup: depends on other binary libraries and install angular client

Unfit for camelCase syntax

It uses camelCase syntax for built-in directives

Dependency Injection is achieved via controller

Dependency Injection is achieved via constructors

3.2.3 NodeJS

Even if Node.js is not used as main part of the any mobile application project, it is necessary to install Node to get access to Ionic command line tools, and if are used, also Bower and Gulp can be installed by using Node´s package manager (npm). As the main use for Node.js is backend, basic information about it is covered here because it is considered the main framework of Ionic.

"Node.js® is a JavaScript runtime built on Chrome's V8 JavaScript engine. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient. Node.js' package ecosystem, npm, is the largest ecosystem of open source libraries in the world" (Node.js, 2017).

As an asynchronous event driven JavaScript runtime, Node is designed to build scalable network applications where several connections can be handled concurrently, and after each connection, the call-back is fired. Nevertheless to save resources, if there is no work to be done, Node remains sleeping until a new connection is made (Node.js, 2017).

It is more and more common to find concurrency models where Operative Systems, web browsers and mobile devises use threads, even if networking may become relatively inefficient. With Node.js this is fixed by not using locks to avoid dead-locking and asynchronous execution. This means that Node can be used in two ways depending on how the code is used: blocking methods (synchronously execution) and non-blocking methods (asynchronously execution) (Node.js, 2017).

Blocking in this case, is when the execution of additional JavaScript code inside Node.js process must wait until a non-JavaScript operation ends, e.g. if a I/O code like reading a PDF file with a blocking method is processed, the rest of the code is not executed until the PDF file is read.

With a non-blocking model, the code that is placed after reading the PDF file is executed without waiting for the PDF file to be read. This is a key design that allows higher throughput and with this model the program does not need to wait network delays.

Node.js uses a single thread to execute JavaScript and in this case concurrency refers to the event loop’s capacity to execute a JavaScript call-back after completing a work. To have in mind that Node uses a single thread is very important, because while coding there must be avoided mixing blocking with non-blocking models while dealing with inputs and outputs.
3.2.4 Cordova

Apache Cordova is an open-source mobile development framework needed to run the code as a native application for different mobile platforms using standard web technologies like HTML5, CSS3 and JavaScript. Cordova also helps to access native functionalities like camera or accelerometer and the application built is executed inside a wrapper for each platform (iOS, Android or Windows Phone) (Cordova, 2015).

This framework is very useful in case e-Paperwork will need to use, for example, fingerprint identification or to use different tools for future releases, packaging for all app store portals and mix native application component with WebView.

To fully understand this framework, how it works with the application created with Ionic and how interacts with the mobile OS, it is necessary to understand Cordova’s application architecture which is shown in Figure 10.

![Cordova Application Architecture](image)

**Figure 10**: Cordova Application Architecture (Cordova, 2015).

In this architecture, the “Web App” is where e-Paperwork code resides, it is implemented as a webpage where the tabs are actually different views of the webpage. For it to run properly, at every app exists the “index.html” file that references all CSS, JavaScript, documents and resources. Afterwards the app is executed within the native application wrapper and so it can be distribute it to the app stores.

When installing Cordova to work with an Ionic project, no plugins are installed by default, depending on which native components are needed for the project, the
corresponding plugins can be added. They provide an interface for Cordova and native components to communicate with each other (Cordova, 2015).

### 3.3 Setup Environment and Emulator

To setup Ionic environment and its emulator, it is very easy and there is not too much requisites needed:

- For Mac or Linux, just a plain terminal will do, but if the OS is Windows, in this project is suggested to have Git Bash installed so it will be easy to access the client, install dependencies, run commands and start the emulator.  
  [https://git-for-windows.github.io/](https://git-for-windows.github.io/)

- To get access to Ionic client and its tools, the latest version of NodeJS must be installed:  
  [https://nodejs.org/en/](https://nodejs.org/en/)

- To add Cordova framework to the environment, it can be installed through the client command interface, and to use it globally:  
  ```bash
  $ npm install -g ionic cordova
  ```

- After installing all dependencies, start Ionic project with version 2 providing the name of the app, this will generate the project folder with all the structure to start e-Paperwork. The project template can be selected form several options starting form a blank template up to a pre made tutorial with embedded help.  
  ```bash
  $ ionic start app blank --v2
  ```

The folder structure created varies from version to version, nevertheless the structure created with Ionic 2 for this e-Paperwork is the following (Complete folder structure will be provided at annex to match the final code):

```
/app/
  [node_modules] /
  [resources] /
  [src] /
   [app] /
    ├── app.component.ts
    ├── app.html
    ├── app.module.ts
    │   ├── app.scss
    │   └── main.ts
    [assets] /
    [pages] /
     └── [home] /
      ├── home.html
      │   ├── home.scss
      │   └── home.ts
    [theme] /
     └── variables.scss
    index.html
    manifest.json
    service-worker.js
  [www] /
  .gitignore
  config.xml
```
To run Ionic emulator for this project, and to enable cross-platform emulation, the following command can be used:

```bash
$ ionic serve -l
```
CHAPTER 4. APP DESIGN

4.1 Database Design and Relations

As expected, during the whole process of developing e-Paperwork, several tables where created to harbour different kind of data and link them to organize the needed information.

Figure 11 shows the Django project structure where the views and URLs are created for the API and APP interface, and the models are migrated to the project so the relationships between tables are established.

For this design, twenty-two models have been created, where eleven are from Django default installation, seven are useful for UPC e-Paperwork project and the rest are for future uses. Table 6 shows the description for what the models have been created and the respective foreign keys as relationships.
<table>
<thead>
<tr>
<th>No</th>
<th>Model</th>
<th>Description</th>
<th>Relationships</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>app_filetype</td>
<td>Future use: Load different type of files to e-Paperwork.</td>
<td><code>file_type_id</code> from app_userfilerequest</td>
</tr>
<tr>
<td>2</td>
<td>app_image</td>
<td>Future use: Save User image.</td>
<td>No Relationship</td>
</tr>
<tr>
<td>3</td>
<td>app_menu</td>
<td>To load side bar menu.</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>app_procedure</td>
<td>To create procedures inside e-Paperwork depending the type of user.</td>
<td><code>procedure_id</code> from app_userprocedureinitial, app_userprocedure, and app_userfilesuploads</td>
</tr>
<tr>
<td>5</td>
<td>app_signature</td>
<td>Future use: Save e-signature from user.</td>
<td><code>user_id</code> to app_userapp</td>
</tr>
<tr>
<td>6</td>
<td>app_userapp</td>
<td>To link a valid user to the specific app format for its type of user.</td>
<td><code>user_id</code> from app_userfilerequest, app_signature, app_userfilesuploads, app_signature and app_userprocedure, user_id and user_type_id to auth_user and app_usertype</td>
</tr>
<tr>
<td>7</td>
<td>app_userfilerequest</td>
<td>Future use: Special procedure to request a specific stored file.</td>
<td><code>file_type_id</code> to app_filetype and <code>user_id</code> to app_userapp</td>
</tr>
<tr>
<td>8</td>
<td>app_userfilesuploads</td>
<td>To store files and respective hash uploaded by users.</td>
<td><code>user_id</code> to app_userapp and <code>procedure_id</code> to app_userprocedure</td>
</tr>
<tr>
<td>9</td>
<td>app_userprocedure</td>
<td>To store step data and respective form’s hash.</td>
<td><code>userProcedure_id</code> from app_userfilesuploads and <code>user_id</code> to app_userapp, user_procedure_initial to app_userprocedurinitial and <code>procedure_id</code> to app_procedure</td>
</tr>
</tbody>
</table>
To avoid skipping steps of active procedures, `user_id` to `app_userapp`, `procedure_id` to `app_procedure` and `user_procedure_initial_id` from `app_userprocedure`.

To differentiate types of user, `user_type_id` from `app_userapp`, `app_procedure` and `app_menu`.

This schema is connected to e-Paperwork models through the `user ID`, generated when a user is validated at "app_userapp" to interact with Django administration tool or with the mobile application.
In Figure 13 there can be appreciated how e-Paperwork behaves internally. By means of the user_id value created with each new user, a request of the corresponding user_type_id is performed so the corresponding app_precedures are assigned to that user. In this way, a student won’t get professor’s or UPC staff’s procedures or vice versa.
Figure 13: e-Paperwork Tables Relations.
Furthermore, the most important relationship is `user_procedure-id` which along with `file_userProcedure_id` fixes each generated hash of the uploaded file and filled form to their respective tables (`app_userprocedure` and `app_userfilesuploads`).

For example, at Figure 14a, there is the first form registered at `app_userprocedure` that shows the corresponding stored hash, which will be used at the mobile app to verify data legitimacy. In the same way at Figure 14b, the first PDF file uploaded for that procedure is stored along with its generated hash, which in every case will be one of a kind.

**Figure 14a:** Form hash in `app_userprocedure` table.

**Figure 15b:** File hash in `app_userfilesuploads` table.

### 4.2 App Design and Views

The view design for e-Paperwork is carefully thought to follow in the most dynamic way, any standard procedure that an enterprise can have. Figure 16 shows the Ionic project structure, which contains the different page views that e-Paperwork displays to each type of user. In this way every user can perform these enterprise´s procedures without a problem.

**Figure 16:** Ionic Project Structure
First to clarify the design, it must be understood that the “userType” can be subdivided in three categories, Normal User, Administrative or Staff Users and Administrators.

Normal User’s App view, is design for people who needs or wants to perform a specific paperwork with the entity, it can be clients or employees with different position at the enterprise. In UPC’s e-Paperwork case it can be students and professors.

Administrative or Staff Users, are exclusively administrative personnel which commonly are in charge of office process, paperwork, etc. and in this case they can actually be UPC Staff working at “Oficina Oberta”. Furthermore, Administrators are people in charge of Django administration tool which can add, delete users, forms, steps and retrieve data.

Consequently, for e-Paperwork to manage in a structured way and exchange dynamically all the data, a predesign of views was necessary. The following set of figures describe how the flow of each page should work, and also how the interaction between types of users should be.

*Side Menu: List of Procedures, Pending Procedures, Finished Procedures and Logout.
*Note: Procedures, steps, forms and form information can be dynamically changed.

*Note: Step will belong to “pending Procedures” if it is not accepted or rejected.
*Note: Form view are disabled when it has been already processed by Staff user.
*Comments: Comments are shown if step is rejected

*Note: The list of users that have started a procedure are separated on types of users.
*Note: Pending steps, accepted steps and rejected steps are differentiated with colours.

*Note: If step is rejected, a note with the reason will be added.

<table>
<thead>
<tr>
<th>Legend</th>
</tr>
</thead>
<tbody>
<tr>
<td>Normal user starting point page</td>
</tr>
<tr>
<td>Next page view</td>
</tr>
<tr>
<td>Button</td>
</tr>
<tr>
<td>Information textbox</td>
</tr>
<tr>
<td>Staff user starting point page</td>
</tr>
</tbody>
</table>

**Set of Figures 17:** View Design
From all pages displayed at Ionic project structure in Figure 16, twelve of them are the used views created according the previous design. The following table describes the use of each view.

**Table 7: App Page Description**

<table>
<thead>
<tr>
<th>No</th>
<th>Page</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>fieldsBySteps</td>
<td>App view for normal users* that provides the “from to be filled” by entering through “steps.”</td>
</tr>
<tr>
<td>2</td>
<td>infoHash</td>
<td>This view provides the hash information of forms and attached files. In addition it provides the “step rejection comment” for normal users.</td>
</tr>
<tr>
<td>3</td>
<td>login</td>
<td>Initial view to authorize the use of e-Paperwork to only registered users. This view also works as user type selector.</td>
</tr>
<tr>
<td>4</td>
<td>pdfViewer</td>
<td>View to display stored PDF documents.</td>
</tr>
<tr>
<td>5</td>
<td>procedures</td>
<td>Displays the procedure lists according to the type of user. These views are only for normal users.</td>
</tr>
<tr>
<td>6</td>
<td>proceduresByUsers</td>
<td>Procedure list view as child of normal user list, this page is just for staff users.</td>
</tr>
<tr>
<td>7</td>
<td>proceduresFilledByUsers</td>
<td>Parent view of proceduresByUsers, this page displays the list of all normal users that have started a step from a procedure.</td>
</tr>
<tr>
<td>8</td>
<td>proceduresInProcess</td>
<td>This view displays all the procedures that have a pending step, the step can be not reviewed or rejected. This page is for normal users only.</td>
</tr>
<tr>
<td>9</td>
<td>Steps</td>
<td>This view displays the list of steps that a procedure have. The number of steps will depend on how many data is expected to be reviewed and entity’s number of process to finish a procedure. This page is for normal users only.</td>
</tr>
<tr>
<td>10</td>
<td>stepsFilledByUsers</td>
<td>List of steps for staff users</td>
</tr>
<tr>
<td>11</td>
<td>stepsInProcessFilledByUser</td>
<td>Child view of proceduresInProcess. Shows a list of pending steps and rejected steps.</td>
</tr>
<tr>
<td>12</td>
<td>stepsResultByUser</td>
<td>View of finished procedures for normal users.</td>
</tr>
</tbody>
</table>

The best way to describe the design and flow of e-Paperwork, is with an actual and real procedure held at the entity. That way, even a business opportunity can also appear when showing how a novel mobile application can solve problems.
CHAPTER 5. How it works

For e-Paperwork to work, all the information that the entity needs from the user is asked through the frontend by forms generated according the steps of each procedure. The forms are saved in a database differentiated by user and type of user, where all sensitive information can be retrieved to make a digest algorithm out of them.

The integration of all frameworks that Ionic uses, gave the opportunity to easily handle and organize data generated from each one of the PDF files and JSON forms attached to the applications which may have sensitive information. This generated data (through a Message-Digest Algorithm 5) is a HASH function that will confirm the authenticity of the information.

After generating a digest, the HASH function retrieved is used as a fingerprint of the document or form, and it is stored to use it as document originality validation. In this way, source and destination can verify and be certain that no changes have done by third parties during the whole process.

For a detailed example, a simple procedure available at e-Secretaria of UPC is described below in a set of figures. The most common procedure done by web is a certificate request where the student receives a valid certificate in PDF format. The same procedure will be described with e-Paperwork comparing it with e-Secretaria process and of course in the same way any face-to-face procedure can also be described with e-Paperwork.

5.1 Certificate request example

A user is registered to the system; in the case of a registration for e-Paperwork, an administrator will register the user through Django graphical interphase http://104.236.76.253:8000/admin/:

Registration process:

- e-Secretaria
  - The registration process from a user is through the “Oficina Oberta” or e-mail, where administrative personnel sends the data to the new student.

- e-Paperwork
  - The registration process must be performed by a Django administrator at graphical interphase with two steps. First administrator must add the student information and register it through web interphase at http://104.236.76.253:8000/admin/auth/user/ and then, to assign that user to a type of user.


- The data information is requested when the new user for e-Paperwork is added.

![Django administration](image)

- Continuing filling the data, the administrator must fill personal information and confirm if the user will be an active e-Paperwork user (students, professors or UPC staff) or Django administrator.

![Add user](image)

- The permissions are selected according to the type of user which can be divided by groups.
To assign the type of forms the user will have (type of user) in the mobile application, it must be linked to a user app type to have a student view at http://104.236.76.253:8000/admin/app/userapp/:

- **User apps** view, the user Std1 is being linked to the format of e-Paperwork specially designed for students (std), and their mobile phone number can also be registered at this step for contact the student if necessary.
At this point a new user for e-Paperwork has been created

Certificate Request:

- Login to e-Secretaría/e-Paperwork with the user and password provided by UPC´s administrator (for this example iOS platform is emulated).

- The procedures available for a student are displayed as a list after login to e-Secretaria/e-Paperwork (All kind of procedures can be added to e-Paperwork).
• And a detailed information for the procedure is displayed if info is tapped.

• The necessary information is required through a form and registered.

• A confirmation of the request is displayed and at e-Paperwork application, the pending procedure can be found at the sidebar.

• In pending Procedures, at “info” button, the hash function of the requesting form is displayed for security confirmation. This signature can be used to identify if the form has been compromised.
Certificate Request Approval/Rejection:

- Once the requests have been made, UPC staff can log in to the application after being registered as “adm”. This will open the e-Paperwork format for personnel that will be in charge of approving, rejecting or sending certificates of the designated procedures to students or professors. The “Procedure List” at “adm” format is divided in procedures initiated by professors and students.

- The procedure provides information about the process at the “info” button and inside of that procedure step, there can be found the detailed information provided by the student. Here, UPC staff can reject with a commentary or upload the requested PDF and approve the procedure.
• If the procedure is accepted, the step of the current procedure will be displayed in green, and if is rejected it will be displayed in red. Furthermore, if a file has been uploaded by any UPC staff, the corresponding signature (hash function) of the document will be displayed at info button.

• At student side, the user will be able to download the file attached by UPC staff if the procedure is accepted.

At both sides (student and staff) the same hash function of the form and document are available, in this way they can be check if some modifications have being made along the process. As explained before in “Providing Trust” section, to acquire an exact hash function by means of MD5 modifying a document is practically impossible.
To corroborate if the process with MD5 is trustworthy, a MD5 generator can be used like onlinemd5 webpage (http://onlinemd5.com/) (Figure 11) to check if e-Paperwork generates the same hash function. With all actual tendencies the use of symmetric cryptography, as the most secure method, will be used more and more as stronger security is needed when exchanging data.

Every entity, organization or enterprise must start to change the old methods of performing their work. Tons of paper from simple procedures are generated every month all over the world. With e-Paperwork not only companies and organizations will save time and money but they will also save natural resources which is essential for a better and sustainable future.

Note 1: For most of Frontend coding, the following set of tutorials were used to acheeve e-Paperwork:

- https://www.youtube.com/watch?v=P8CE13R_F3Y&list=PLo0J5TRjgp2VH_Qi6gHEBcd_WYkDsvAUQ
  - set of tutorial videos (various authors)
- http://ionicframework.com/docs/intro/installation/
  - Ionic2 manual (Ionic docs)

Note 2: For most of Backend coding and setup, the following set of tutorials where used to acheeeve e-Paperwork:

- https://www.youtube.com/watch?v=KiGE2KpiwtA
  - The Best Django Tutorial - How to Use Django 1.9 with PostgreSQL & Bootstrap (Chris Hawkes)
- https://www.youtube.com/watch?v=2OD3oeodNms&t=452s
  - How to Install Django 1.7.x on Linux Mint / Ubuntu with Python 3.4 and PostgreSQL (Dototot)
- https://www.youtube.com/watch?v=qgGIqRFvFFk&list=PL6gx4Cwl9DGBlmzzFclgLhKTTfNLIx1IK
  - set of Django tutorials (thenewboston)

Note 3: All the code for e-Paperwork (Frontend and Backend) was validated and reviewed by Hernan Rengel, La Paz-Bolivia.
CHAPTER 6. Business Plan

6.1 Executive Summary

1. **Overview.**
   e-Paperwork is a mobile application, suitable for all platforms. It is designed to manage all kind of paperwork in a smartphone, to avoid face-to-face procedures and digitalize process.

2. **Description.**
   Bureaucratic procedures are common at every enterprise or entity, they store tons of paper at warehouse with all kind of sensitive data and large waiting lines are always a good waste of time. e-Paperwork aims to eradicate all these problems, customizing a mobile application for that specific entity’s procedures.

   To provide trust when handling these sensitive information, e-Paperwork uses Asymmetric key cryptography to validate data originality. When sending files or forms between users with e-Paperwork, this method works as a digital signature providing a unique fingerprint for each document or form.

3. **Your goals for the business.**
   Within one year, e-Paperwork is expected to be used in various types of entities, like banks, government offices, regulation institutions and universities.
   Within five years, e-Paperwork is expected to be an important and widely used mobile application in several cities.

4. **Ideal target market.**
   Governmental entities that regulates the financial sector.

5. **Competition and differences.**
   All mobile applications used to electronically sign documents and fill forms like Adobe Fill & sign or DocuSign.
   The differences between these Apps are:
   - e-Paperwork uses symmetric key cryptography as a validation fingerprint.
   - e-Paperwork is customized for customer depending their procedures.
   - e-Paperwork integrates an interactive way of doing entity’s paperwork with programmable forms.

6. **Management team.**
   Every customer will have a personalized attention until e-Paperwork is fully functional with all the requested procedures are integrated.
   This type of service is not available at any of the competitors’ solutions.
6.2 Company Description

1. Company mission statement
   “Provide our customers a fast, comfortable and ubiquitous way of doing all kind of paperwork”.

2. Company vision
   “To be the most important and trustful mobile application, used to exchange valid information and documentation”.

3. Company values
   - Comfort.
   - Agility
   - Usefulness
   - Trustworthiness
   - Efficiency

4. Company goals
   Short-term
   - Place e-Paperwork as a successful Fintech mobile application.
   - Expand to three main cities of the country, reaching not only financial enterprises.
   - Expand to nearby countries and keep expanding.
   Long-term
   - Evolve world’s way of life.

5. Target market
   e-Paperwork targets are all medium and large companies, which at their daily work involve large amount of paperwork between staff/clients or staff/staff.

6. Industry
   Mobile application industry is growing too fast. By the year 2020 there may be 5 million apps at the App Store. Nevertheless, developing of mobile applications with synchronous and asynchronous features that securely exchanges data are rare in the market.

   Mobile application development will keep growing along with developing technologies, therefore e-Paperwork right now is up to date for short-term competition, but it will keep evolving for long term competition.

   There is few competing software for securely exchange documentation and even fewer that runs on smartphones. e-Paperwork solution gives personalized Mobile Application as a Service (MAaaS) that is not seen at present market.

7. Legal structure
a. As a start-up, the best form of business is sole proprietorship to take care of the restricted income and organize it in the best way.
b. The best option as a sole proprietorship is to register ROTH-IS as a limited liability company which at Catalonia the equivalent will be “Sociedad Limitada Unipersonal (SLU)”

6.3 Products & Services

1. **Products and services**: e-Paperwork is a mobile application created with the latest programming technologies, so it can run natively on any mobile platform. This App is designed according to customers’ needs, coordinating along with them so their paperwork flows are transposed to e-Paperwork in the best way.

2. **The problem e-Paperwork solves**: Every enterprise faces bureaucracy in some way, mainly with processes that involve their customers and of course between their staff. Most of these processes need to send documents between offices, print them, sign them, deliver them to their customers, sign them again, ask for more documents or client’s data and so on.

   e-Paperwork solves these problems by taking all those processes and paperwork to a smartphone, so all the work is done without the necessity of going to site. By using cryptographic methods to validate the exchange of legal documents, this mobile application can set up a secure paperwork environment that easily can improve the way of life.

   Other products are focused on a general use and only digital documents. e-paperwork is focused on individualize the application for each entity, that is, an app for one entity, customized just for its available procedures or paperwork, within the enterprise staff or with its customers.

3. **Other advantages**: The main differences between other solutions are that e-paperwork is personally customized for the customer, according to their daily paperwork. With this, not only the PDF files are secured with cryptographic methods, but also the web forms.

   Another advantage is the use of symmetric key cryptography to validate data originality, which unlike with digital certificates (asymmetric cryptography) e-Paperwork is far more secure to quantum computing attacks.

   And last but not least, the design of e-Paperwork allows the user to follow procedures which will replace any on site paperwork providing data in a trustful environment.
4. **Pricing:** For each customer (company or entity), e-Paperwork will have the following pricing method:

   a. **Initial cost:** Eleven thousand Euros (£ 11.000). This fee entails the coordination with entities' administrative staff to establish how each paperwork will be translated or transposed to e-Paperwork. Subsequently, the deployment of the base application will be done along with the backend and database management.

   b. **Procedure cost:** Four hundred and forty Euros (£ 440). This fee will be charged after each full procedure is integrated and fully operational at e-Paperwork.

   c. **Monthly maintenance.** Two hundred and twenty Euros (£ 220). This monthly fee entails the maintenance service for the Virtual Private Server, cloud storage and e-Paperwork update for newest features.

Even if the number of procedures to migrate is large, the competitive landscape in terms of pricing will be low end. This is because e-Paperwork is not charged by the user, it is charged by service, and most of enterprises, even if they are small business, have a lot of clients or employees.

With constant customer support and constant e-Paperwork evolving, the projected profit margin with one customer with 5 procedures to migrate or transpose will be 23%.

6.4 **Market research**

- **The total size industry**
  There is no mobile applications that manages entire procedures using symmetric key cryptography as secure data exchange method. Nevertheless, mobile applications that uses electronic signature with different formats and securing them with asymmetric key cryptography, are more common. The following charts show the usage of main mobile applications that uses electronic signature to sign digital documents and asymmetric cryptography security.
Chart 1: Google play store data

<table>
<thead>
<tr>
<th>DocuSign</th>
<th>SignEasy</th>
<th>Adobe F&amp;S</th>
<th>SIGNificant</th>
</tr>
</thead>
<tbody>
<tr>
<td>Estimated Downloads</td>
<td>1.000.000</td>
<td>500.000</td>
<td>1.000.000</td>
</tr>
<tr>
<td>Average rating</td>
<td>4,5</td>
<td>4,2</td>
<td>4,3</td>
</tr>
</tbody>
</table>

Chart 2: Competitors comparison

<table>
<thead>
<tr>
<th>DocuSign</th>
<th>SignEasy</th>
<th>Adobe F&amp;S</th>
<th>SIGNificant</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ever used</td>
<td>24648</td>
<td>7576</td>
<td>8778</td>
</tr>
<tr>
<td>Active users</td>
<td>17716</td>
<td>4790</td>
<td>5459</td>
</tr>
</tbody>
</table>
• Trends in the industry

Chart 3: Global Trend with English Search (Google, 2017)

Chart 4: Global Interest by Topic Search Spanish vs English (Google, 2017)

Interest over time:
Numbers represent search interest relative to the highest point on the chart for the given region and time. A value of 100 is the peak popularity for the term. A value of 50 means that the term is half as popular. Likewise a score of 0 means the term was less than 1% as popular as the peak.

Figure 19: Interest by region according chart4

Because language influence the region search tendencies, the comparison by country varies. In Spain for example, digital certificate trend is the strongest one by far. In the following chart, yellow represents the digital certificate interest in Spain.
The same tendency replicates in all Spain cities, nevertheless, there is a reduced interest on mobile applications which provides electronic signature and file validation management. That is why, the electronic signature and digital certificate industry are still thought as new technologies, so there is a huge potential market to reach and introduce e-Paperwork.

- The total size of target market.

The number of active enterprises increased 1.6% during 2015 reaching 3.24 millions in Spain. 15.8% of active enterprises have 20 or more years old, while 20.1% has less than 20 years old. ((DIRCE), 2016)

Table 8: Enterprise Distribution by Size (Dirección General de Industria y de la Pequeña y Mediana Empresa, 2016)

<table>
<thead>
<tr>
<th>Number of employees</th>
<th>DIRCE 2015 (1/1/15)</th>
</tr>
</thead>
<tbody>
<tr>
<td>With no employees</td>
<td>1,751,964</td>
</tr>
<tr>
<td>1 to 9</td>
<td>1,297,861</td>
</tr>
<tr>
<td>10 to 50</td>
<td>110,086</td>
</tr>
<tr>
<td>51 to 249</td>
<td>18,497</td>
</tr>
<tr>
<td>Total SMEs</td>
<td>3,178,408</td>
</tr>
<tr>
<td>250 and more</td>
<td>3,913</td>
</tr>
</tbody>
</table>

The share expected to be reached by e-Paperwork is 30% of enterprises with 10 or more employees in Spain. The total number of enterprises at
target market is 132,496, having a total of 39,748 enterprises considering just the 30%.
According the “General Direction of Industry, Small and Medium Business”, Barcelona has 0.16% of large enterprises. This means that an estimated number of 6,359 enterprises are possible clients for e-Paperwork. (Dirección General de Industria y de la Pequeña y Mediana Empresa, 2016)

- Trends in the target market
  There has been a lot of effort trying to widely introduce the use of digital and electronic signature with different methods and formats. Despite the failure, there are still being developed several directives and conventions. One of the most recent and important regulations is the “electronic IDentification, Authentication and trust Services” (eIDAS), which means authorities are still pushing this trend adoption.

  Besides the European Union authorities that started to push adoption on digital and electronic signature, telecommunication enterprises like Movistar began the incorporation of these trends to their business. The world may be seeing the beginning of digital and electronic signature adoption.

6.5 Marketing Plan

Barriers to entry

- High start-up costs.
  e-Paperwork do not have high start-up costs.
- High production costs.
  Only mass production (3 to above customers) at the same time will incur on high production costs. Which means acquiring programmers for each client.
- High marketing costs.
  Webpage and social network marketing will be useful at the beginning of the project.
- Brand recognition challenges.
  To successfully please a client will boost brand recognition.
- Finding qualified employees.
  Social media pages related to specific skills are useful to contact possible employees.
- Need for specialized technology or patents.
  All specialized technology used at e-Paperwork is open source.
- Tariffs and quotas.
  Not available for software development.
- Unionization in your industry.
  e-Paperwork comply with regulatory entities.
Threats and opportunities

- Changes in government regulations.
  Exchange of valid legal documentation is strictly bond to government regulations, any change will imply to actualize e-Paperwork´s method of verifying the attached documents and data filled at app´s forms validity.

- Changes in technology.
  Because e-Paperwork is built with the latest technology available, it is expected to be in constant actualization providing new versions to customers.

- Changes in the economy.
  A drastic economy change, may lead in customers not acquiring e-Paperwork because lack of resources for new projects. Nevertheless, in long term periods e-paperwork can save in expenses budget to the company.

- Changes in your industry.
  If a better and easier solution with more security is developed to replace symmetric key cryptography, e-Paperwork can take advantage of its versatility and incorporate the new technology. If mobile applications industry is left behind, e-Paperwork would have to be adapted to the new environment.

*Note: Company Description Worksheet, Product & Service Description Worksheet and SWOT Analysis Worksheet are included at Annex A*
CHAPTER 7. References


Bernstein, D. J. (n.d.). *Introduction to post-quantum cryptography*. Chicago, USA.


CHAPTER 8. GLOSSARY

**API**: Application Programming Interface is a set of subroutine definitions, protocols, and tools for building application software. In general terms, it is a set of clearly defined methods of communication between various software components. A good API makes it easier to develop a computer program by providing all the building blocks, which are then put together by the programmer. An API may be for a web-based system, operating system, database system, computer hardware or software library.

**Asymmetric-key cryptography**: Is any cryptographic system that uses pairs of keys: public keys which may be disseminated widely, and private keys which are known only to the owner. This accomplishes two functions: authentication, which is when the public key is used to verify that a holder of the paired private key sent the message, and encryption, whereby only the holder of the paired private key can decrypt the message encrypted with the public key.

**Block Storage**: Block Storage allows you to create and attach additional storage volumes to your DigitalOcean Droplets. Volumes are an independent resource that can easily be moved from one Droplet to another within the same datacentre. Attached volumes function like locally connected storage drives, allowing you to manage your storage with familiar tools and techniques.

**Blockchain**: Is a distributed database that is used to maintain a continuously growing list of records, called blocks. Each block contains a timestamp and a link to a previous block. A blockchain is typically managed by a peer-to-peer network collectively adhering to a protocol for validating new blocks. By design, blockchains are inherently resistant to modification of the data.

**Cross-Platform**: Is computer software that is implemented on multiple computing platforms. Cross-platform software may be divided into two types; one requires individual building or compilation for each platform that it supports, and the other one can be directly run on any platform without special preparation, e.g., software written in an interpreted language or pre-compiled portable bytecode for which the interpreters or run-time packages are common or standard components of all platforms.

**CSS**: Is a style sheet language used for describing the presentation of a document written in a markup language.

**Digest algorithm**: Is a cryptographic hash function containing a string of digits created by a one-way hashing formula. Message digests are designed to protect the integrity of a piece of data or media to detect changes and alterations to any part of a message. They are a type of cryptography utilizing hash values that can warn the copyright owner of any modifications applied to their work.

**Digital certificates**: Is a computer file generated by a certification service entity that associates identity data to a natural person, body or company thus confirming its digital identity on the Internet.
**e-signature**: Refers to data in electronic form, which is logically associated with other data in electronic form and which is used by the signatory to sign. This type of signature provides the same legal standing as a handwritten signature as long as it adheres to the requirements of the specific regulation it was created under (e.g., eIDAS in the European Union, NIST-DSS in the USA or ZertES in Switzerland).

**Frameworks**: Is an abstraction in which software providing generic functionality can be selectively changed by additional user-written code, thus providing application-specific software. A software framework provides a standard way to build and deploy applications.

**Hash function**: Is any function that can be used to map data of arbitrary size to data of fixed size. The values returned by a hash function are called hash values, hash codes, digests, or simply hashes.

**HTML**: Is the standard markup language for creating web pages and web applications.

**JavaScript**: Is a high-level, dynamic, untyped, interpreted run-time language. It has been standardized in the ECMAScript language specification.

**JSON**: JavaScript Object Notation is an open-standard file format that uses human-readable text to transmit data objects consisting of attribute-value pairs and array data types (or any other serializable value). It is a very common data format used for asynchronous browser/server communication, including as a replacement for XML in some AJAX-style systems.

**Native APP**: Is a software application designed to run on a mobile device such as a smartphone or tablet computer. Mobile apps often stand in contrast to desktop applications that run on desktop computers, and with web applications which run in mobile web browsers rather than directly on the mobile device.

**Symmetric-key cryptography**: This method are algorithms for cryptography that use the same cryptographic keys for both encryption of plaintext and decryption of ciphertext. The keys may be identical or there may be a simple transformation to go between the two keys. The keys, in practice, represent a shared secret between two or more parties that can be used to maintain a private information link. This requirement that both parties have access to the secret key is one of the main drawbacks of symmetric key encryption.

**TypeScript**: Is a free and open-source programming language developed and maintained by Microsoft. It is a strict syntactical superset of JavaScript, and adds optional static typing and class-based object-oriented programming to the language.

**Web App**: Is a client–server software application in which the client (or user interface) runs in a web browser. Common web applications include webmail, online retail sales, online auctions, wikis, instant messaging services and many other functions.
**Web enabled:** Refers to a product or service that can be used through, or in conjunction with, the World Wide Web. A Web-enabled product may be accessed through a Web browser or be able to connect to other Web-based applications in order to synchronize data.
# ANNEXES

**Annex A**

**Company Description Worksheet**

<table>
<thead>
<tr>
<th><strong>Business Name/Product Name</strong></th>
<th>ROTH-IS/e-Paperwork</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Company Mission Statement</strong></td>
<td>Provide our customers a fast, comfortable and ubiquitous way of doing all kind of paperwork.</td>
</tr>
<tr>
<td><strong>Company Philosophy/Values</strong></td>
<td>Make life easier by saving time using a smartphone anywhere.</td>
</tr>
<tr>
<td><strong>Company Vision</strong></td>
<td>To be the most important and trustful mobile application, used to exchange valid information and documentation</td>
</tr>
</tbody>
</table>
| **Goals & Milestones** | 1. Place e-Paperwork as a successful Fintech mobile application.  
2. Expand to three main cities of the country reaching not only financial enterprises.  
3. Expand to nearby countries and keep expanding.  
4. Evolve world’s way of life. |
| **Target Market** | Medium and large enterprises with large amount of paperwork between staff/clients or staff/staff. |
| **Industry/Competitors** | 1. Electronic signing and stamping documents / DocuSign  
2. Electronic signing and stamping documents / Adobe Reader  
3. Electronic signing and stamping documents / XolidoSign |
| **Legal Structure/Ownership** | Sole proprietorship / Limited Liability Company (LLC) |
# Product & Service Description Worksheet

<table>
<thead>
<tr>
<th>Product Name/Logo</th>
<th>e-Paperwork</th>
</tr>
</thead>
<tbody>
<tr>
<td>Product/ Service Idea</td>
<td>Multiplatform mobile application to host any enterprise or entities’ paperwork, digitalizing forms, legal documents and securing this data exchange with cryptographic signing.</td>
</tr>
<tr>
<td>Special Benefits</td>
<td>With e-Paperwork, customers will be able to work from home without the need to attend an on-site paperwork, tons of paper used every year will be avoided and waiting lines can be eradicated.</td>
</tr>
</tbody>
</table>
| Unique Features | • Symmetric key cryptography.  
• Dedicated and customized app for customers.  
• Easy forms edition.  
• Easy procedure adaptation to the App. |
| Limits and Liabilities | • Not competitive with asymmetric key cryptography solutions.  
• Newly introduced to market.  
• Simple user interface design. |
| Production and Delivery | e-Paperwork is delivered in three steps:  
1. Server setup and base App provided to customer at contract signing.  
2. Design adaptation to customer requirements.  
3. Customer’s procedures or paperwork integrated to the App. |
| Suppliers | 1. DigitalOcean’s Virtual private server to host backend.  
2. Django to provide database integration and administration tool.  
3. PostgreSQL to provide database solution.  
4. IONIC to provide frontend solution. |
| Intellectual Property Special Permits | e-Paperwork must be used only by designated customer under distribution according contract terms.  
The distribution of e-Paperwork to other companies or entities from the client will incur on legal actions. |
| Product/ Service Description | e-Paperwork provides integration of all kind of procedures or paperwork that an enterprise may have. Its objective is to avoid the use of physical documentation, avoid face-to-face procedures by hosting them on a smartphone and save time by accessing the service anytime and anywhere.  
After a registered user is logged, the user will be able to access a list of all available procedures according to its type of user. At each procedure, the user must follow a set of steps to finish each paperwork where an administrative user can receive the information, process it, and accept it or reject the step. |
SWOT Analysis Worksheet

<table>
<thead>
<tr>
<th></th>
<th>Strengths</th>
<th>Weaknesses</th>
<th>Opportunities</th>
<th>Threats</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Product/Service</strong></td>
<td>Symmetric key cryptography, customized for customers and one of a kind.</td>
<td>Not competitive with asymmetric key cryptography solutions.</td>
<td>Novel trend and increasing popularity of related technologies.</td>
<td>Powerful competition with electronic signature solutions.</td>
<td></td>
</tr>
<tr>
<td><strong>Offering</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Brand/Marketing</strong></td>
<td>Innovative.</td>
<td>Without the support of a strong company.</td>
<td>Use of different new social media like Steemet.</td>
<td>Solution can be copied.</td>
<td></td>
</tr>
<tr>
<td><strong>Staff/HR</strong></td>
<td>Few staff needed.</td>
<td>Special programing skills needed.</td>
<td>New staff can learn about new technologies used.</td>
<td>HR can replicate the solution.</td>
<td></td>
</tr>
<tr>
<td><strong>Finance</strong></td>
<td>Not too much start-up cost needed.</td>
<td>Difficult to acquire financial aid.</td>
<td>Novel solution can be interesting for funders.</td>
<td>To mass production, larger financial aid will be needed.</td>
<td></td>
</tr>
<tr>
<td><strong>Operations/Management</strong></td>
<td>Easy to manage with few clients.</td>
<td>No too much of management experience.</td>
<td>Involve with Fintech environment.</td>
<td>Start-up business management.</td>
<td></td>
</tr>
<tr>
<td><strong>Market</strong></td>
<td>Great number of possible customers.</td>
<td>Product not suitable for micro business.</td>
<td>Uncollected market.</td>
<td>Enterprises may not be willing to change.</td>
<td></td>
</tr>
</tbody>
</table>

Can any of your strengths help with improving your weaknesses or combating your threats? If so, please describe how below.

The type of service provided, is designed to not compete directly with asymmetric key solutions. Customized service is not provided by competitors. If number of customers increase, costs may be suitable for small business.

Based on the information above, what are your immediate goals/next steps?

Create a webpage with product information and acquire the first customer to place a good example of use case.

Based on the information above, what are your long-term goals/next steps?

Adapt e-Paperwork for financial entities and reach most of the important entities in Barcelona as fixed customers.
Annex B

Backend Code.

```python
from django.contrib.auth.hashers import make_password
from django.contrib.auth.models import User
from rest_framework import serializers
from rest_framework.serializers import ModelSerializer
from app.models import *

class Base64ImageField(serializers.ImageField):
    def to_internal_value(self, data):
        from django.core.files.base import ContentFile
        import base64
        import six
        import uuid

        # Check if this is a base64 string
        if isinstance(data, six.string_types):
            # Check if the base64 string is in the "data:" format
            if 'data:' in data and ';base64,' in data:
                # Break out the header from the base64 content
                header, data = data.split(';base64,')
            # Try to decode the file. Return validation error if it fails.
            try:
                decoded_file = base64.b64decode(data)
            except TypeError:
                self.fail('invalid_image')

            # Generate file name:
            file_name = str(uuid.uuid4())[:12]  # 12 characters are more than enough.

            # Get the file name extension:
            file_extension = self.get_file_extension(file_name, decoded_file)

            complete_file_name = "%s.%s" % (file_name, file_extension,)
            data = ContentFile(decoded_file, name=complete_file_name)
        return super(Base64ImageField, self).to_internal_value(data)

    def get_file_extension(self, file_name, decoded_file):
        import imghdr
        extension = imghdr.what(file_name, decoded_file)
        extension = "jpg" if extension == "jpeg" else extension
        return extension

class ImageSerializer(serializers.ModelSerializer):
    image = Base64ImageField(
        max_length=None, use_url=True,
    )

class Meta:
```
```
62. model = Image
63. fields = ('id', 'image', 'time_created',)
64. 65.
66. class UserAuthSerializer(ModelSerializer):
67.     class Meta:
68.         model = User
69.         fields = '__all__'
70. 71.
72. class LoginSerializer(ModelSerializer):
73.     password = serializers.CharField(
74.         style={'input_type': 'password'}
75.     )
76. 77.
78. class MenuSerializer(ModelSerializer):
79.     class Meta:
80.         model = Menu
81.         fields = '__all__'
82. 83.
84. class UserTypeSerializer(ModelSerializer):
85.     class Meta:
86.         model = UserType
87.         fields = '__all__'
88. 89.
90. class SignatureSerializer(ModelSerializer):
91.     class Meta:
92.         model = Signature
93.         fields = '__all__'
94. 95.
96. class ProcedureSerializer(ModelSerializer):
97.     def to_representation(self, instance):
98.         ret = super(ProcedureSerializer, self).to_representation(instance)
99.         ret['form_fields'] = json.loads(ret['form_fields'])
100.        return ret
101. 102.
103. class UserAppSerializer(ModelSerializer):
104.     # menu = MenuSerializer(many=True)
105.     user = UserAuthSerializer(many=False, read_only=True)
106.     user_type = UserTypeSerializer(many=False, read_only=True)
107. 108.
109. class UserProcedureSerializerPost(ModelSerializer):
110.     def to_representation(self, instance):
111.         ret = super(UserProcedureSerializerPost, self).to_representation(instance)
112.         ret['full_form'] = json.loads(ret['full_form'])
113.        return ret
class Meta:
    model = UserProcedure
    fields = '__all__'

class UserProcedureSerializer(ModelSerializer):
    user = UserAppSerializer(many=False, read_only=True)
    user_type = UserTypeSerializer(many=False, read_only=True)
    procedure = ProcedureSerializer(many=False, read_only=True)

def to_representation(self, instance):
    ret = super(UserProcedureSerializer, self).to_representation(instance)
    ret['full_form'] = json.loads(ret['full_form'])
    return ret

class Meta:
    model = UserProcedure
    fields = '__all__'

class UserProcedureInitialSerializer(ModelSerializer):
    procedures = UserProcedureSerializer(many=True, read_only=True)

class Meta:
    model = UserProcedureInitial
    fields = '__all__'

class UserProcedureSerializerPut(ModelSerializer):
    user = UserAppSerializer(many=False, read_only=True)
    user_type = UserTypeSerializer(many=False, read_only=True)
    procedure = ProcedureSerializer(many=False, read_only=True)

def to_representation(self, instance):
    ret = super(UserProcedureSerializerPut, self).to_representation(instance)
    ret['full_form'] = json.loads(ret['full_form'])
    return ret

class Meta:
    model = UserProcedure
    fields = '__all__'
    read_only_fields = ('full_form', 'hash', 'step', 'active')

class UserFilesUploadsSerializer(ModelSerializer):
    class Meta:
        model = UserFilesUploads
        fields = '__all__'

class FileTypeSerializer(ModelSerializer):
    class Meta:
        model = FileType
        fields = '__all__'

class UserFileRequestSerializer(ModelSerializer):
    class Meta:
        model = UserFileRequest
        fields = '__all__'
```python
1. from django.conf.urls import url, include
2. from rest_framework.routers import DefaultRouter
3. from rest_framework.authtoken import views as auth_views
4. from rest_framework_swagger.views import get_swagger_view
5.
6. from api.views import *
7.
8. from app.admin import MenuAdmin
9. from . import views
10.
11. schema_view = get_swagger_view(title='ePaperwork API')
12.
13. router = DefaultRouter()
14. router.register(r'usersAuth', UserAuthListViewSet)
15. router.register(r'usersAppAuth', UserAppAuthListViewSet)
16. router.register(r'menu', MenuListViewSet)
17. router.register(r'users', UserListViewSet)
18. router.register(r'signature', SignatureListViewSet)
19. router.register(r'procedures', ProceduresListViewSet)
20. router.register(r'userFillByUserProcedures', ProcedureFillByUserListViewSet)
21. router.register(r'userProcedureInitialViewSet', UserProcedureInitialViewSet)
22. router.register(r'userFilesUploads', UserFilesUploadsViewSet)
23. router.register(r'fileType', FileTypeViewSet)
24. router.register(r'userFileRequest', UserFileRequestViewSet)
25.
26. urlpatterns = [
27.     url(r'^api-docs/$', schema_view),
28.     url(r'^$', views.index, name='index'),
29.     url(r'^', include(router.urls)),
30.     url(r'^usersProcedures/$', ProcedureFillByUserListView.as_view()),
31.     url(r'^token-auth/$', auth_views.obtain_auth_token)
32. ]
```
from api.serializers import *

from django.contrib.auth.models import User

from app.models import Menu, UserApp, Signature, UserFilesUploads

from rest_framework import authentication

from rest_framework import exceptions

def index(request):
    return HttpResponse("ePapperwork API."

class UserAuthListViewSet(viewsets.ModelViewSet):
    serializer_class = UserAuthSerializer
    queryset = User.objects.all()
    lookup_field = 'id'

    # permission_classes = permissions.IsAuthenticatedOrReadOnly,

def list(self, request, *args, **kwargs):
    print(request.user)
    return super(UserAuthListViewSet, self).list(request, *args, **kwargs)

class UserAuthAppListViewSet(viewsets.ModelViewSet):
    serializer_class = UserAuthSerializer
    queryset = User.objects.all()
    lookup_field = 'username'

    # permission_classes = permissions.IsAuthenticatedOrReadOnly,

def list(self, request, *args, **kwargs):
    print(request.user)
    return super(UserAuthAppListViewSet, self).list(request, *args, **kwargs)

class UserAppListDataViewSet(viewsets.ModelViewSet):
    serializer_class = UserAppSerializer
    queryset = UserApp.objects.all()
    lookup_field = 'user_id'

    # permission_classes = permissions.IsAuthenticatedOrReadOnly,

def list(self, request, *args, **kwargs):
    print(request.user)
    return super(UserAppListDataViewSet, self).list(request, *args, **kwargs)

class UserAppListViewSet(viewsets.ModelViewSet):
    serializer_class = UserAppSerializer
    queryset = UserApp.objects.all()
    lookup_field = 'id'

    # permission_classes = permissions.IsAuthenticatedOrReadOnly,

def list(self, request, *args, **kwargs):
    print(request.user)
    return super(UserAppListViewSet, self).list(request, *args, **kwargs)

class UserMenuListViewSet(viewsets.ModelViewSet):
    serializer_class = MenuSerializer
queryset = Menu.objects.all()
lookup_field = 'id'

# permission_classes = (permissions.IsAuthenticated,)
def list(self, request, *args, **kwargs):
    user = request.user
    return super(UserMenuListViewSet, self).list(request, *args, **kwargs)

def get_queryset(self):
    query = self.request.query_params
    queryset = self.queryset
    if "user_type" in query.keys():
        queryset = queryset.filter(user_type=query.get("user_type"))
    return queryset

class UserMenuListViewSet(viewsets.ModelViewSet):
    serializer_class = UserMenuSerializer
    queryset = Menu.objects.all()
    lookup_field = 'id'

def get_queryset(self):
    query = self.request.query_params
    queryset = self.queryset
    user = query.get("user")
    status = query.get("status")
    procedure = query.get("procedure")

    if user and (procedure is None) and (status is None):
        queryset = queryset.filter(user=user.get("user"))
    if user and status and (procedure is None):
        queryset = queryset.filter(user=user.get("user"),
                                 status=query.get("status"))
    if user and procedure and (status is None):
        queryset = queryset.filter(user=user.get("user"),
                                    procedure=query.get("procedure"))
    if user and procedure and status:
        queryset = queryset.filter(user=user.get("user"),
                                    procedure=query.get("procedure"),
                                    status=query.get("status"))
    return queryset

def list(self, request, *args, **kwargs):
    user = request.user
    return super(UserMenuListViewSet, self).list(request, *args, **kwargs)

class UserMenuListViewSet(viewsets.ModelViewSet):
    serializer_class = UserMenuSerializer
    queryset = Menu.objects.all()
    lookup_field = 'id'

class UserProcedureInitialViewSet(viewsets.ModelViewSet):
    serializer_class = UserProcedureInitialSerializer
    queryset = UserProcedureInitial.objects.all()
    lookup_field = 'id'
def get_queryset(self):
    query = self.request.query_params
    queryset = self.queryset
    user = query.get("user")
    status = query.get("status")
    procedure = query.get("procedure")

    if user and (procedure is None) and (status is None):
        queryset = queryset.filter(user=user.get("user"))
    if user and status and (procedure is None):
        queryset = queryset.filter(user=user.get("user"),
                                 status=query.get("status"))
    if user and procedure and (status is None):
        queryset = queryset.filter(user=user.get("user"),
                                    procedure=query.get("procedure"))
    if user and procedure and status:
        queryset = queryset.filter(user=user.get("user"),
                                    procedure=query.get("procedure"),
                                    status=query.get("status"))
    return queryset

def list(self, request, *args, **kwargs):
    user = request.user
    return super(UserProcedureInitialViewSet, self).list(request, *args, **kwargs)

class ProcedureFillByUserListviewSet(viewsets.ModelViewSet):
    serializer_class = UserProcedureSerializer
    queryset = UserProcedure.objects.all()
    lookup_field = 'id'
def get_queryset(self):
    query = self.request.query_params
    queryset = self.queryset
    user = query.get("user")
    status = query.get("status")
    procedure = query.get("procedure")

    if user and (procedure is None) and (status is None):
        queryset = queryset.filter(user=user.get("user"))
    if user and status and (procedure is None):
        queryset = queryset.filter(user=user.get("user"),
                                 status=query.get("status"))
    if user and procedure and (status is None):
        queryset = queryset.filter(user=user.get("user"),
                                    procedure=query.get("procedure"))
    if user and procedure and status:
        queryset = queryset.filter(user=user.get("user"),
                                    procedure=query.get("procedure"),
                                    status=query.get("status"))
    return queryset

def list(self, request, *args, **kwargs):
    user = request.user
    return super(UserProcedureInitialViewSet, self).list(request, *args, **kwargs)
# permission_classes = (permissions.IsAuthenticated,)
def list(self, request, *args, **kwargs):
    return super(ProcedureFillByUserListViewSet, self).list(request, *args, **kwargs)
def get_queryset(self):
    query = self.request.query_params
    queryset = self.queryset
    user = query.get("user")
    procedure = query.get("procedure")
    userProcedure = query.get("userProcedure")
    if user and (procedure is None) and (userProcedure is None):
        queryset = queryset.filter(user=query.get("user"))
    if procedure and (user is None) and (userProcedure is None):
        queryset = queryset.filter(procedure=query.get("procedure"))
    if procedure and user and (userProcedure is None):
        queryset = queryset.filter(procedure=query.get("procedure"), user=query.get("user"))
    if userProcedure and user and (procedure in None):
        queryset = queryset.filter(userProcedure=query.get("userProcedure"), user=query.get("user"))
    if userProcedure and procedure and user:
        queryset = queryset.filter(userProcedure=query.get("userProcedure"), procedure=query.get("procedure"), user=query.get("user"))
    return queryset
# def put(self, request, *args, **kwargs):
#     instance = self.get_object()
#     instance.active = request.data.get("status")
#     instance.save()
#     
#     serializer = self.get_serializer(instance)
#     serializer.is_valid(raise_exception=True)
#     self.perform_update(serializer)
#     return Response(serializer.data)
class ProcedureFillByUserPutViewSet(viewsets.ModelViewSet):
    serializer_class = UserProcedureSerializerPut
    queryset = UserProcedure.objects.all()
    lookup_field = 'id'
    # permission_classes = (permissions.IsAuthenticated,)

def list(self, request, *args, **kwargs):
    return super(ProcedureFillByUserPutViewSet, self).list(request, *args, **kwargs)
def get_queryset(self):
    query = self.request.query_params
    queryset = self.queryset
    if "user" in query.keys():
def put(self, request, *args, **kwargs):
    instance = self.get_object()
    instance.active = request.data.get("status")
    instance.save()

    serializer = self.get_serializer(instance)
    serializer.is_valid(raise_exception=True)
    self.perform_update(serializer)
    return Response(serializer.data)

class ProcedureFillByUserListView(APIView):
    serializer_get = UserProcedureSerializer
    serializer_post = UserProcedureSerializerPost

    def get(self, request, id=None, format=None):
        procedures = UserProcedure.objects.all()
        response = self.serializer_get(procedures, many=True)
        return Response(response.data)

    def post(self, request, format=None):
        serializer = UserProcedureSerializerPost(data=request.data)
        if serializer.is_valid():
            serializer.save()
            return Response(serializer.data, status=status.HTTP_201_CREATED)
        return Response(serializer.errors, status=status.HTTP_400_BAD_REQUEST)

class FileUploadForm(forms.Form):
    file_source = forms.FileField

class SignatureListViewSet(viewsets.ModelViewSet):
    serializer_class = SignatureSerializer
    queryset = Signature.objects.all()
    lookup_field = 'id'

    def list(self, request, *args, **kwargs):
        return super(SignatureListViewSet, self).list(request, *args, **kwargs)

class ProceduresListViewSet(viewsets.ModelViewSet):
    serializer_class = ProcedureSerializer
    queryset = Procedure.objects.all()
    lookup_field = 'id'

    def list(self, request, *args, **kwargs):
        return super(ProceduresListViewSet, self).list(request, *args, **kwargs)

    def get_queryset(self):
        query = self.request.query_params
        queryset = self.queryset
        if "user_type" in query.keys():
            queryset = queryset.filter(user_type=query.get("user_type")
        return queryset

class UserFilesUploadsViewSet(viewsets.ModelViewSet):
serializer_class = UserFilesUploadsSerializer
queryset = UserFilesUploads.objects.all()
lookup_field = 'id'

def pre_save(self, obj):
    obj.document = self.request.FILES.get('file')

def list(self, request, *args, **kwargs):
    return super(UserFilesUploadsViewSet, self).list(request, *args, **kwargs)

def get_queryset(self):
    query = self.request.query_params
    queryset = self.queryset
    queryset = queryset.filter(active=True)
    user = query.get("user")
    procedure = query.get("procedure")
    userProcedure = query.get("userProcedure")
    if user and (procedure is None) and (userProcedure is None):
        queryset = queryset.filter(user=query.get("user"))
    if procedure and (user is None) and (userProcedure is None):
        queryset = queryset.filter(procedure=query.get("procedure"))
    if procedure and user and (userProcedure is None):
        queryset = queryset.filter(procedure=query.get("procedure"),
                                  user=query.get("user"))
    if userProcedure and user and (procedure is None):
        queryset = queryset.filter(userProcedure=query.get("userProcedure"),
                                  user=query.get("user"))
    if userProcedure and procedure and user:
        queryset = queryset.filter(userProcedure=query.get("userProcedure"),
                                 procedure=query.get("procedure"),
                                 user=query.get("user"))
    return queryset

class FileTypeViewSet(viewsets.ModelViewSet):
    serializer_class = FileTypeSerializer
    queryset = FileType.objects.all()
    lookup_field = 'id'
    
    # permission_classes = permissions.IsAuthenticatedOrReadOnly,

    def list(self, request, *args, **kwargs):
        print(request.user)
        return super(FileTypeViewSet, self).list(request, *args, **kwargs)

class UserFileRequestViewSet(viewsets.ModelViewSet):
    serializer_class = UserFileRequestSerializer
    queryset = UserFileRequest.objects.all()
    lookup_field = 'id'

def pre_save(self, obj):
    obj.document = self.request.FILES.get('file')

def list(self, request, *args, **kwargs):

return super(UserFileRequestViewSet, self).list(request, *args, **kwargs)

def get_queryset(self):
    query = self.request.query_params
    queryset = self.queryset
    queryset = queryset.filter(active=True)
    if "user" in query.keys():
        queryset = queryset.filter(user=query.get("user"))
    return queryset

from django.contrib import admin

# Register your models here.
from app.models import *

class UserTypeAdmin(admin.ModelAdmin):
    pass

class UserAppAdmin(admin.ModelAdmin):
    pass

class MenuAdmin(admin.ModelAdmin):
    pass

class ProcedureAdmin(admin.ModelAdmin):
    pass

class UserProcedureAdmin(admin.ModelAdmin):
    pass

class SignatureAdmin(admin.ModelAdmin):
    pass

class UserFilesUploadsAdmin(admin.ModelAdmin):
    pass

class UserFileRequestAdmin(admin.ModelAdmin):
    pass

class FileTypeAdmin(admin.ModelAdmin):
    pass

class UserProcedureInitialAdmin(admin.ModelAdmin):
    pass

django.contrib.admin.site.register(UserType, UserTypeAdmin)
django.contrib.admin.site.register(UserApp, UserAppAdmin)
django.contrib.admin.site.register(Menu, MenuAdmin)
50. admin.site.register(Procedure, ProcedureAdmin)
51. admin.site.register(UserProcedure, UserProcedureAdmin)
52. admin.site.register(Signature, SignatureAdmin)
53. admin.site.register(UserFilesUploads, UserFilesUploadsAdmin)
54. admin.site.register(FileType, FileTypeAdmin)
55. admin.site.register(UserFileRequest, UserFileRequestAdmin)
56. admin.site.register(UserProcedureInitial, UserProcedureInitialAdmin)

[epaperworkBackEnd]/
[app]/
models.py

1. from __future__ import unicode_literals
2. 3. from django.contrib.auth.models import User
4. from django.db import models
5. import uuid
6. 7. 8. # Create your models here
9. 10. 11. class UserType(models.Model):
12.     user_type = models.CharField(max_length=30)
13.     description = models.CharField(max_length=200)
14. 15.     def __str__(self):
16.         return "%s" % self.user_type
17. 18. 19. class UserApp(models.Model):
20.     user = models.ForeignKey(User)
21.     phone = models.CharField(max_length=30, blank=True, null=True)
22.     active = models.BooleanField(default=False)
23.     user_type = models.ForeignKey(UserType, on_delete=models.CASCADE)
24. 25.     def __str__(self):
26.         return "%s %s" % (self.phone, self.active)
27. 28. 29. class Procedure(models.Model):
30.     name = models.CharField(max_length=200)
31.     created_at = models.DateTimeField(auto_now_add=True, blank=False)
32.     init_date = models.DateTimeField(auto_now_add=True, blank=False)
33.     end_date = models.DateTimeField(auto_now_add=True, blank=True)
34.     description = models.CharField(max_length=200)
35.     form_fields = models.TextField(verbose_name="JSON", blank=True)
36.     procedure_type = models.BooleanField(default=False)
37.     user_type = models.ForeignKey(UserType, on_delete=models.CASCADE)
38. 39.     def __str__(self):
40.         return "%s" % self.name
41. 42. 43. class UserProcedureInitial(models.Model):
44.     user = models.ForeignKey(UserApp, on_delete=models.CASCADE)
45.     procedure = models.ForeignKey(Procedure, on_delete=models.CASCADE)
46.     created_at = models.DateTimeField(auto_now_add=True, blank=False)
47.     modified_at = models.DateTimeField(auto_now_add=True, blank=False)
active = models.BooleanField(default=False)
status = models.IntegerField(default=0)
def __str__(self):
    return "%s" % self.procedure
class UserProcedure(models.Model):
    user = models.ForeignKey(UserApp, on_delete=models.CASCADE)
    user_procedure_initial = models.ForeignKey(UserProcedureInitial, on_delete=models.CASCADE, default=None, related_name='procedures')
    procedure = models.ForeignKey(Procedure, on_delete=models.CASCADE)
    full_form = models.TextField()
    step_order = models.IntegerField(default=1)
    hash = models.CharField(max_length=200)
    modified_at = models.DateTimeField(auto_now_add=True, blank=False)
    active = models.BooleanField(default=False)
    observations = models.CharField(max_length=1000, default="None")
    status = models.IntegerField(default=0)
def __str__(self):
    return "%s" % self.procedure
class Signature(models.Model):
    signature_path = models.CharField(max_length=200)
    created_at = models.DateTimeField(auto_now_add=True, blank=False)
    active = models.BooleanField(default=False)
    user = models.ForeignKey(UserApp, on_delete=models.CASCADE)
def __str__(self):
    return "%s %s" % (self.active, self.user.user.first_name)
class Menu(models.Model):
    url = models.CharField(max_length=50)
    display_name = models.CharField(max_length=50)
    user_type = models.ForeignKey(UserType, on_delete=models.CASCADE)
def __str__(self):
    return "%s %s" % (self.url, self.display_name)
class Image(models.Model):
    image = models.ImageField(upload_to='item_images')
    time_created = models.DateTimeField(auto_now_add=True)
def scramble_uploaded_filename(instance, filename):
    extension = filename.split(".")[1]
    return "{}{}".format(uuid.uuid4(), extension)
class UserFilesUploads(models.Model):
    field = models.CharField(blank=True, max_length=200)
    document = models.FileField(blank=True, default='', upload_to=scramble_uploaded_filename)
    user = models.ForeignKey(UserApp, on_delete=models.CASCADE)
    procedure = models.ForeignKey(Procedure, on_delete=models.CASCADE)
    userProcedure = models.ForeignKey(UserProcedure, on_delete=models.CASCADE, default=18)
    step = models.CharField(max_length=200)
115.     hash = models.CharField(blank=True, max_length=200)
116.     created_at = models.DateTimeField(auto_now_add=True, blank=False)
117.     modified_at = models.DateTimeField(auto_now_add=True, blank=False)
118.     active = models.BooleanField(default=False)
119.     
120.     def __str__(self):
121.         return "%s" % self.field
122.     
123.     class Filetype(models.Model):
124.         name = models.CharField(max_length=200)
125.         created_at = models.DateTimeField(auto_now_add=True, blank=False)
126.         modified_at = models.DateTimeField(auto_now_add=True, blank=False)
127.         active = models.BooleanField(default=False)
128.         
129.         def __str__(self):
130.             return "%s" % self.name
131.     
132.     class UserFileRequest(models.Model):
133.         OPTIONS = [(0, 'Pendiente'),
134.             (1, 'Aprobado'),
135.             (2, 'Rechazado'),
136.         )
137.         user = models.ForeignKey(UserApp, on_delete=models.CASCADE)
138.         file_type = models.ForeignKey(FileType, on_delete=models.CASCADE)
139.         document = models.CharField(max_length=200)
140.         document_file = models.FileField(blank=True, null=True, default='', upload_to=scramble_uploaded_filename)
141.         status = models.IntegerField(choices=OPTIONS, default=0)
142.         observations = models.CharField(max_length=1000, default="Ninguno")
143.         created_at = models.DateTimeField(auto_now_add=True, blank=False)
144.         modified_at = models.DateTimeField(auto_now_add=True, blank=False)
145.         active = models.BooleanField(default=False)
146.         
147.         def __str__(self):
148.             return "%s" % self.document

# [epaperworkBackEnd]/
#    [app]/
#       [templates]/
#          urls.py

1.     from django.conf.urls import url
2.     
3.     from app.views import SignatureRegister
4.     from . import views
5.     
6.     urlpatterns = [
7.         url(r'^$', views.index, name='index'),
8.         url(r'^signature_register/$', SignatureRegister.as_view(), name='signature_register'),
9.     ]
```python
1. from django.shortcuts import render
2. from django.http import HttpResponse
3. from django.views.generic import CreateView
4.
5.
6. # Create your views here.
7. from app.models import UserApp
8.
9.
10. def index(request):
    11.     return HttpResponse("ePaperwork App.")
12.
13.
14. class SignatureRegister(CreateView):
15.     template_name = 'signature_register.html'
16.     model = UserApp
17.     fields = '__all__'
18.     success_url = 'signature_register'

```

```python
from django.conf.urls import include, url
from django.contrib import admin
from django.conf.urls.static import static
from django.conf import settings

urlpatterns = [
    url(r'^app/', include('app.urls')),
    url(r'^api/', include('api.urls')),
    url(r'^admin/', admin.site.urls),
] + static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
```
Frontend Code.

```typescript
import {Component, ViewChild} from '@angular/core';
import {Nav, Platform} from 'ionic-angular';
import {StatusBar, SplashScreen} from 'ionic-native';

import {Page1} from '../pages/page1/page1';
import {Page2} from '../pages/page2/page2';
import {LoginPage} from '../pages/login/login';
import {Steps} from '../pages/fieldsBySteps/fieldsBySteps';
import {APIService} from '../providers/epapperwork-service';
import {Procedure} from '../pages/procedures/procedure';
import {ProceduresFilledByUsers} from '../pages/proceduresFilledByUsers/proceduresFilledByUsers';
import {ProceduresInProcess} from '../pages/proceduresInProcess/proceduresInProcess';
import {FileRequest} from '../pages/fileRequest/fileRequest';
import {FileRequestByUser} from '../pages/fileRequestByUser/fileRequestByUser';
import {FillProcedures} from '../pages/fillProcedures/fillProcedures';

@Component({
  templateUrl: 'app.html',
  providers: [APIService]
})
export class MyApp {
  @ViewChild(Nav) nav: Nav;
  rootPage: any = LoginPage;
  pages: Array<{title: string, component: any}>;

  constructor(public platform: Platform,
              public apiService: APIService) {
    this.initializeApp();
  }

  // setTimeout(function() {
  //   this.loadMenu(this.globalVars.getUsertType());
  //   console.log(this.globalVars.getUsertType())
  // }, 2000);

  loadMenu(user_type) {
    this.apiService.loadMenu(user_type)
      .then(data => {
        console.log(data)
        this.pages = data;
      });
  }

  initializeApp() {
    this.platform.ready().then(() => {
      StatusBar.styleDefault();
      SplashScreen.hide();
    });
  }

  openPage(page) {
```
this.apiService.getLoggedUser()
  .then(data => {
    console.log(data)
    if (page == "/edit_user/" ) {
      this.nav.setRoot(Page2)
    } else if (page == "/procedures_list/" ) {
      if(data.user_type.user_type == "adm"){
        this.nav.setRoot(ProceduresFilledByUsers, {
          userDataLogged: data
        })
      } else {
        this.nav.setRoot(Procedure, {
          userDataLogged: data
        })
      }
    } else if (page == "/procedures_in_process/" ) {
      this.nav.setRoot(ProceduresInProcess, {
        state: 0
      })
    } else if (page == "/procedures_finished/" ) {
      this.nav.setRoot(ProceduresInProcess, {
        state: 1
      })
    } else if (page == "/file_request/" ) {
      this.nav.setRoot(FileRequest, {
        userDataLogged: data
      })
    } else if (page == "/file_request_list/" ) {
      this.nav.setRoot(FileRequestByUser, {
        userDataLogged: data
      })
    } else if (page == "/close_session/" ) {
      this.nav.setRoot(LoginPage);
    }
  });

<ion-menu id="menu" [content]="content">
  <ion-header>
    <ion-toolbar>
      <ion-title>Menu</ion-title>
    </ion-toolbar>
  </ion-header>
  <ion-content>
    <ion-list>
      <button menuClose ion-item *ngFor="let p of pages" (click)="openPage(p.url)"
        >
        {{p.display_name}}
      </button>
    </ion-list>
  </ion-content>
</ion-menu>
18. <!-- Disable swipe-to-go-back because it's poor UX to combine STGB with side menus -->
19. <ion-nav [root]="rootPage" #content swipeBackEnabled="false"></ion-nav>

```typescript
import { NgModule, ErrorHandler } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { IonicModule, IonicModule, IonicErrorHandler } from 'ionic-angular';
import { MyApp } from './app.component';
import { Page1 } from '../pages/page1/page1';
import { Page2 } from '../pages/page2/page2';
import { LoginPage } from '../pages/login/login';
import { Procedure } from '../pages/procedures/procedure';
import { Steps } from '../pages/steps/steps';
import { FillProcedures } from '../pages/fillProcedures/fillProcedures';
import { ProceduresFilledByUsers } from '../pages/proceduresFilledByUsers/proceduresFilledByUsers';
import { ProceduresByUsers } from '../pages/proceduresByUsers/proceduresByUsers';
import { StepsFilledByUser } from '../pages/stepsFilledByUser/stepsFilledByUser';
import { StepsResultByUser } from '../pages/stepsResultByUser/stepsResultByUser';
import { ProceduresInProcess } from '../pages/proceduresInProcess/proceduresInProcess';
import { StepsInProcessFilledByUser } from '../pages/stepsInProcessFilledByUser/stepsInProcessFilledByUser';
import { InfoHash } from '../pages/infoHash/infoHash';
import { FileRequest } from '../pages/fileRequest/fileRequest';
import { FileRequestByUser } from '../pages/fileRequestByUser/fileRequestByUser';
import { PdfViewer } from '../pages/pdfViewer/pdfViewer';
import { Storage } from '@ionic/storage';
import { Transfer, FileUploadOptions, TransferObject } from '@ionic-native/transfer';
import { File } from '@ionic-native/file';
import { PdfViewerComponent } from 'ng2-pdf-viewer';

@NgModule({
  declarations: [
    MyApp,
    Page1,
    Page2,
    LoginPage,
    Procedure,
    Steps,
    FieldsBySteps,
    FillProcedures,
    ProceduresFilledByUsers,
    ProceduresByUsers,
    StepsFilledByUser,
    StepsResultByUser,
    ProceduresInProcess,
    StepsInProcessFilledByUser,
    InfoHash,
    FileRequest,
    FileRequestByUser,
  ]
})
```
PdfViewer,
PdfViewerComponent
],
imports: [
  IonicModule.forRoot(MyApp),
  BrowserModule
],
bootstrap: [IonicApp],
entryComponents: [
  MyApp,
  Page1,
  Page2,
  LoginPage,
  Procedure,
  Steps,
  FieldsBySteps,
  FillProcedures,
  ProceduresFilledByUsers,
  ProceduresByUsers,
  StepsFilledByUser,
  StepsResultByUser,
  ProceduresInProcess,
  StepsInProcessFilledByUser,
  InfoHash,
  FileRequest,
  FileRequestByUser,
  PdfViewer
],
providers: [Storage, {
  provide: ErrorHandler,
  useClass: IonicErrorHandler,
}, Transfer, File]
})
export class AppModule {}
<ion-list>
  <ion-item *ngFor="let fields of textViewArray; let i = index;" display="block">
    <ion-label floating>{{fields.label}}</ion-label>
    <ion-input type="{{fields.type}}" required="{{fields.required}}" name="{{fields.id}}"></ion-input>
  </ion-item>
</ion-list>

<ion-list>
  <ion-item *ngFor="let fields of selectViewArray; let i = index;">
    <ion-label>{{selectViewArray[0].label}}</ion-label>
    <ion-select [(ngModel)]="selectViewArrayObj[fields.label]" [ngModelOptions]="{standalone: true}">
      <ion-option *ngFor="let options of fields.options">{{options.name}}</ion-option>
    </ion-select>
  </ion-item>
</ion-list>

<ion-list *ngIf="checkBoxViewArray[0]">
  <ion-list-header>>{{checkBoxViewArray[0].label}}</ion-list-header>
  <ion-item *ngFor="let fields of checkBoxViewArray[0].options; let i = index;">
    <ion-label>{{fields.text}}</ion-label>
    <ion-checkbox value="{{fields.value}}" [(ngModel)]="checkBoxViewArrayObj[fields.text]" [ngModelOptions]="{standalone: true}"></ion-checkbox>
  </ion-item>
</ion-list>

<ion-list *ngIf="radioViewArray[0]">
  <ion-list-header>>{{radioViewArray[0].label}}</ion-list-header>
  <ion-item *ngFor="let fields of radioViewArray[0].options; let i = index;">
    <ion-label>{{fields.text}}</ion-label>
    <ion-radio value="{{fields.value}}" [(ngModel)]="radioViewArrayObj[radioViewArray[0].label]" [ngModelOptions]="{standalone: true}"></ion-radio>
  </ion-item>
</ion-list>

<ion-list *ngIf="fileViewArray.length > 0">
  <ion-item *ngFor="let fields of fileViewArray; let i = index;">
    <span>{{ fields.label }}</span>(change)="fileChangeEvent($event)" accept=".pdf" name="{{fields.id}}"></ion-input>
  </ion-item>
</ion-list>

<button ion-button type="submit" block>Register</button>

</form>
```typescript
import {Component} from '@angular/core';
import {NavController, NavParams} from 'ionic-angular';
import {APIService} from '../../providers/epaperwork-service';
import {Util} from '../../providers/util';
import {AlertController} from 'ionic-angular';
import {Procedure} from '../procedures/procedure';
import {Http, Headers, RequestOptions} from '@angular/http';

declare var cordova: any;

@Component({
  selector: 'page-fields-by-steps',
  templateUrl: 'fieldsBySteps.html',
  providers: [APIService, Util]
})
export class FieldsBySteps {

  steps = FieldsBySteps;

  public textViewArrayObj: any;
  public selectViewArrayObj: any;
  public checkBoxViewArrayObj: any;
  public radioViewArrayObj: any;
  public fileViewArrayObj: any;
  public descriptionObject: any;

  public formFields: any;

  public textViewArray: any;
  public selectViewArray: any;
  public checkBoxViewArray: any;
  public radioViewArray: any;
  public fileViewArray: any;
  public procedureInitialByUser: any;
  public dataUserLogged: any;
  public order: any;
  public procedure_type: any;

  public todo: any;

  public filesToUpload: Array<File>;
  public filesArrayToUpload: any;

  private data: any;

  private step: any;
  private procedureId: any;

  constructor(public http: Http, 
              public navCtrl: NavController, 
              public apiService: APIService, 
              public util: Util, 
              public params: NavParams, 
              public alertCtrl: AlertController) {
```
this.http = http;
this.textViewArrayObj = {};
this.selectViewArrayObj = {};
this.checkBoxViewArrayObj = {};
this.radioViewArrayObj = {};
this.fileViewArrayObj = {};
this.formFields = [];
this.textViewArray = [];
this.selectViewArray = [];
this.checkBoxViewArray = [];
this.radioViewArray = [];
this.fileViewArray = [];
this.procedureInitialByUser = [];
this.todo = {};
this.filesToUpload = [];
this.filesArrayToUpload = [];
this.descriptionObject = [];
this.step = this.params.get('step');
this.procedureId = this.params.get('procedureId');
this.dataUserLogged = this.params.get('user');
this.order = this.params.get('order');
this.procedure_type = this.params.get('procedure_type');
this.loadFields(this.params.get('fields'));
this.loadInitialProceduresByUser();

loadInitialProceduresByUser() {
    this.apiService.getUserProcedureInitial(this.dataUserLogged.userAppId)
        .then(data => {
            for (let i = 0; i < data.length; i++) {
                if(data[i].procedure == this.procedureId || data[i].procedure == 0) {
                    this.procedureInitialByUser.push(data[i])
                }
            }
        })
}
loadFields(fields) {
    console.log(fields)
    let description = fields.description;
    if (description.length > 0) {
        this.descriptionObject.push(description);
    }
    for (let i = 0; i < fields.form_fields.length; i++) {
        this.formFields.push(fields.form_fields[i])
    }
```javascript
if (fields.form_fields[i].type == "text") {
    this.textViewArray.push(
        this.util.getTextView(fields.form_fields[i])
    )
} else if (fields.form_fields[i].type == "select") {
    this.selectViewArray.push(
        this.util.getSelectView(fields.form_fields[i])
    )
} else if (fields.form_fields[i].type == "checkbox") {
    this.checkBoxViewArray.push(
        this.util.getCheckBoxView(fields.form_fields[i])
    )
} else if (fields.form_fields[i].type == "radio") {
    this.radioViewArray.push(
        this.util.getRadioView(fields.form_fields[i])
    )
} else if (fields.form_fields[i].type == "file") {
    this.fileViewArray.push(
        this.util.getFileView(fields.form_fields[i])
    )
}
}

fileChangeEvent(fileInput: any) {
    this.filesToUpload = Array<File>(fileInput.target.files);
    console.log(this.filesToUpload)
    this.filesArrayToUpload.push(this.filesToUpload)
    console.log(this.filesArrayToUpload)
}

logForm() {
    this.util.showLoadingDialog()
    let saveArray = [];
    let checkBoxSelectedOptions = [];
    let radioSelectedOptions = [];
    console.log(this.checkBoxViewArrayObj)
    console.log(this.radioViewArrayObj)
    for (let j = 0; j < this.formFields.length; j++) {
        // console.log(this.formFields[j])
        if (this.formFields[j].type == "text") {
            this.formFields[j]["value"] = this.textViewArrayObj[this.formFields[j].id]
        } else if (this.formFields[j].type == "checkbox") {
            for (let c = 0; c < this.formFields[j].options.length; c++) {
                if (this.getSelectedDegree(this.checkBoxViewArrayObj[j], this.formFields[j].options[c].name)) {
                    checkBoxSelectedOptions.push(
                        this.formFields[j].options[c].name,
                        "value": this.formFields[j].options[c].value
                    )
                }
            }
        } else if (this.formFields[j].type == "radio") {
            for (let c = 0; c < this.formFields[j].options.length; c++) {
```
183.     if (this.formFields[j].options[c].value == this.radioViewArrayObj[this.formFields[j].label]) {
184.         radioSelectedOptions.push({
185.             "name": this.formFields[j].options[c].name,
186.             "value": this.formFields[j].options[c].value
187.         })
188.     }
189.  
190.     this.formFields[j]["value"] = radioSelectedOptions
191.  
192.     saveArray.push(this.formFields[j])
193.  
194. // ------------------------------------------------------
195.  
196.     let userLoggedId = this.dataUserLogged.userAppId;
197.  
198.     console.log(this.filesToUpload)
199.  
200.     var saveFile = {};
201.  
202.     let saveProcedureInitial = {
203.         "active": true,
204.         "status": 0,
205.         "user": userLoggedId,
206.         "procedure": this.procedureId
207.     }
208.  
209.     console.log(JSON.stringify(saveProcedureInitial))
210.  
211.     console.log(this.procedureId)
212.  
213.     let dataProcedureInitial = this.checkIfExistPreviousProcedure(this.procedureId);
214.  
215.     console.log(dataProcedureInitial)
216.     console.log(dataProcedureInitial.exist)
217.     console.log(this.procedure_type)
218.  
219.     if (dataProcedureInitial.exist) {
220.         this.saveStepByUser(userLoggedId, saveArray, dataProcedureInitial.id);
221.     } else {
222.         this.saveProcedureAndStepByUser(userLoggedId, saveArray, saveProcedureInitial)
223.     }
224.  
225.     saveStepByUser(userLoggedId, saveArray, dataUserProcedureInitialId)
226.  
227.  
228.  
229.  
230.  
231.     let saveStep = {
232.         "user": userLoggedId,
233.         "procedure": this.procedureId,
234.         "full_form": JSON.stringify(saveArray),
235.         "hash": this.util.getStepHash(saveArray),
236.         "step": this.step,
237.         "active": true,
238.         "step_order": this.order,
239.         "user_procedure_initial": dataUserProcedureInitialId
240.     }
var count = 0;

this.apiService.saveProcedure(saveStep)
  .then(data => {
    this.data = data;
    let filesArrayLength = this.filesArrayToUpload.length;
    console.log(filesArrayLength)
    if (filesArrayLength > 0) {
      if (this.data) {
        for (let i = 0; i < this.filesArrayToUpload.length; i++) {
          console.log(this.filesArrayToUpload[i])
          this.util.getFileHash(this.filesArrayToUpload[i])
            .then(dataHash => {
              let file: File = this.filesArrayToUpload[i][0];
              let formData: FormData = new FormData();
              formData.append('document', file, file.name);
              formData.append('field', this.fileViewArray[i].label);
              formData.append('step', this.step);
              formData.append('hash', dataHash);
              formData.append('active', true);
              formData.append('user', userLoggedId);
              formData.append('procedure', this.procedureId);
              formData.append('userProcedure', data.id);
              let headers = new Headers();
              headers.append('Accept', 'application/json');
              let options = new RequestOptions({headers: headers});
              this.apiService.saveFileForStep(formData, options)
                .then(result => {
                  count++;
                  if (count > this.filesArrayToUpload.length - 1) this.done(result);
                }, error => {
                  this.doneError(error)
                })
            }, error => {
              this.showAlertError(error)
            })
      } else if (filesArrayLength <= 0 && this.procedure_type) {
        this.showErrorModal();
      }
    } else if (filesArrayLength <= 0 && this.procedure_type) {
        this.showErrorModal();
      } else {
let formData: FormData = new FormData();
formData.append('document', null, null);
formData.append('field', null);
formData.append('step', this.step);
formData.append('hash', '');
formData.append('active', true);
formData.append('user', userLoggedId);
formData.append('procedure', this.procedureId);
let headers = new Headers();
headers.append('Accept', 'application/json');
let options = new RequestOptions({headers: headers});
this.apiService.saveFileForStep(formData, options)
  .then(result => {
    count++;
    if (count > this.filesArrayToUpload.length - 1) this.done(result);
    }, error => {
      this.doneError(error)
    })
  }
else {
  this.done(data)
}
}, error => {
  this.showAlertError(error);
}
};
saveProcedureAndStepByUser(userLoggedId, saveArray, saveProcedureIn initial) {
  var count = 0;
  this.apiService.saveUserProcedureInitial(saveProcedureInitial)
    .then(dataUserProcedureInitial => {
      let saveStep = {
        "user": userLoggedId,
        "procedure": this.procedureId,
        "full_form": JSON.stringify(saveArray),
        "hash": this.util.getStepHash(saveArray),
        "step": this.step,
        "active": true,
        "step_order": this.order,
        "user_procedure_initial": dataUserProcedureInitial.
      id
    }
  }
  this.apiService.saveProcedure(saveStep)
    .then(data => {
      this.data = data
      let filesArrayLength = this.filesArrayToUpload.length;
      console.log(filesArrayLength)
      if (filesArrayLength > 0) {

console.log("entra al primer if > 0")

if (this.data) {
  load.length; i++) {
    console.log(this.filesArrayToUpload[i])
    this.util.getFileHash(this.filesArrayToUpload[i][0]).then(dataHash => {
      let file: File = this.filesArrayToUpload[i][0];
      formData.append('document', file, file.name);
      formData.append('field', this.isFileViewArray[i].label);
      formData.append('step', this.step);
      formData.append('hash', dataHash);
      formData.append('active', true);
      formData.append('user', userLoggedId);
      formData.append('procedure', this.procureId);
      headers.append('Accept', 'application/json');
      let options = new RequestOptions(headers);
      this.apiService.saveFileForStep(formData, options).then(result => {
        count++;
        if (count > this.filesArrayToUpload.length - 1) this.done(result);
      }, error => {
        this.doneError(error)
      })
    })
  }
}

} else if (filesArrayLength <= 0 && this.procedure_type) {
  console.log("entra al primer if < 0")
  let formData: FormData = new FormData();
formData.append('document', null, null);
formData.append('field', null);
formData.append('step', this.step);
formData.append('hash', "");
formData.append('active', true);
formData.append('user', userLoggedId);
formData.append('procedure', this.procedureId);
formData.append('userProcedure', data.id);
let headers = new Headers();
headers.append('Accept', 'application/json');
let options = new RequestOptions({headers: headers});
this.apiService.saveFileForStep(formData, options)
  .then(result => {
    count++;
    if (count > this.filesArrayToUpload.length - 1) this.done(result);
    }, error => {
      this.doneError(error)
      });
    } else {
      this.done(data)
    }
    }, error => {
      this.showAlertError(error)
    });
  }
  this.showAlertError(error)
});
done(result) {
  this.util.hideLoadingDialog();
  this.showAlertSuccess(result)
}
doneError(error) {
  this.util.hideLoadingDialog();
  this.showAlertError(error)
}
checkIfExistPreviousProcedure(procedureId) {
  let response = null;
  console.log(this.procedureInitialByUser)
  if (this.procedureInitialByUser.length > 0) {
    console.log(“entra if length”)
    for (let i = 0; i < this.procedureInitialByUser.length; i++) {
      console.log(“entra for”)
      if(procedureId == this.procedureInitialByUser[i].proced
```javascript
console.log("entra if")
let pendingProcedure = this.procedureInitialByUser[i];
console.log(pendingProcedure)
if (pendingProcedure.status == 0) {
    console.log("entra if Status")
    if (pendingProcedure.procedure == procedureId) {
        console.log("entra if procedure")
        response = {
            "exist": true,
            "id": pendingProcedure.id
        }
    } else {
        response = {
            "exist": false,
            "id": 0
        }
    }
} else {
    console.log("entra else 1")
    response = {
        "exist": false,
        "id": 0
    }
} else {
    console.log("entra else 3 hay procedimiento pero no del mismo tipo")
    response = {
        "exist": false,
        "id": 0
    }
} else {
    console.log("entra else 2")
    response = {
        "exist": false,
        "id": 0
    }
} return response;

getSelectedDegree(obj, attr) {
    return obj[attr]
}
```

handler: () => {
  this.navCtrl.setRoot(Procedure, {
    userDataLogged: data
  });
}

this.navCtrl.setRoot(Procedure, {
  userDataLogged: data
});

alert.present();

showAlertSuccess(error) {
  this.apiService.getLoggedUser()
    .then(data => {
      let alert = this.alertCtrl.create({
        title: 'Data registered successfully',
        buttons: [
          {
            text: 'Ok',
            role: 'ok',
            handler: () => {
              this.navCtrl.setRoot(Procedure, {
                userDataLogged: data
              });
            }
          }
        ]
      });
      alert.present();
    });
  alert.present();
};

showAlertSuccess(error) {
  this.apiService.getLoggedUser()
    .then(data => {
      let alert = this.alertCtrl.create({
        title: 'Data registered successfully',
        buttons: [
          {
            text: 'Ok',
            role: 'ok',
            handler: () => {
              this.navCtrl.setRoot(Procedure, {
                userDataLogged: data
              });
            }
          }
        ]
      });
      alert.present();
    });
  alert.present();
}
<ion-item *ngFor="let fields of selectViewArray; let i = index;"
    (ngModel)="fields.value.id">
    <ion-label>Gender</ion-label>
    <ion-select [(ngModel)]="fields.value.id">
        <ion-option *ngFor="let options of fields.options" value="f" selected="true">
            {{options.name}}
        </ion-option>
    </ion-select>
</ion-item>

<ion-list *ngIf="checkBoxViewArray[0]">
    <ion-list-header>
        {{checkBoxViewArray[0].label}}
    </ion-list-header>
    <ion-item *ngFor="let fields of checkBoxViewArray[0].options; let i = index;">
        <ion-label>{{fields.text}}</ion-label>
        <ion-checkbox [checked]="getChecked(fields)" disabled readonly></ion-checkbox>
    </ion-item>
</ion-list>

<ion-list *ngIf="radioViewArray[0]" radio-group>
    <ion-list-header>
        {{radioViewArray[0].label}}
    </ion-list-header>
    <ion-item *ngFor="let fields of radioViewArray[0].options; let i = index;">
        <ion-label>{{fields.text}}</ion-label>
        <ion-radio [checked]="getSelected(fields)" disabled readonly></ion-radio>
    </ion-item>
</ion-list>

<ion-list *ngIf="fileViewArray.field">
    <ion-list-header>
        Archivos
    </ion-list-header>
    <ion-item-divider *ngFor="let fileFields of filesByUser; let i = index;">
        <h2 class="list-item-left" item-left (click)="viewFile($event, fileFields)">
            {{fileFields.field}}
        </h2>
    </ion-item-divider>
</ion-list>

<ion-row *ngIf="procedureItem.length > 0">
    <ion-list *ngIf="procedureItem[0].procedure.procedure_type">
        <ion-list-header>
            Attach File
        </ion-list-header>
        <ion-item-divider *ngFor="let fields of procedureItem; let i = index;">
            <input type="file" required
                (change)="fileChangeEvent($event)" accept=".pdf"
```typescript
import {Component} from '@angular/core';
import {NavController, NavParams} from 'ionic-angular';
import {APIService} from '../../providers/epaperwork-service';
import {Util} from '../../providers/util';
import {ParallelHasher} from 'ts-md5/dist/parallel_hasher';
import {StepsResultByUser} from '../stepsResultByUser/stepsResultByUser';
```

9. import {AlertController} from 'ionic-angular';
10. import {PdfViewer} from '../pdfViewer/pdfViewer';
11. import {Http, Headers, RequestOptions} from '@angular/http';
12.
13. @Component({
14.   selector: 'page-page1',
15.   templateUrl: 'fillProcedures.html',
16.   providers: [APIService, Util]
17. })
18.
19. export class FillProcedures {
20.   public userProcedures: any;
21.   procedure = FillProcedures;
22.   public fieldsList: any;
23.   steps = FillProcedures;
24.   public textViewArrayObj: any;
25.   public selectViewArrayObj: any;
26.   public checkBoxViewArrayObj: any;
27.   public radioViewArrayObj: any;
28.   public fileViewArrayObj: any;
29.   public formFields: any;
30.   public textViewArray: any;
31.   public selectViewArray: any;
32.   public checkBoxViewArray: any;
33.   public radioViewArray: any;
34.   public fileViewArray: any;
35.   public stepsByProcedure: any;
36.   public stepsFilledByUserByProcedure: any;
37.   public updateProcedureData: any;
38.   public todo: any;
39.   public filesToUpload: Array<File>;
40.   public filesByUser: any;
41.   private data: any;
42.   private step: any;
43.   private userId: any;
44.   private procedureId: any;
45.   private full_form: any;
46.   private idProcedure: any;
47.   private procedureItem: any;
48.   private idUserProcedure: any;
49.   private procedureIdByStep: any;
50.   private userType: any;
51.   private userProcedureId: any;
52.   private procedureStatus: any;
53.
54.   constructor(
55.     public navCtrl: NavController,
56.     public apiService: APIService,
57.     public util: Util,
58.     public params: NavParams,
59.     public alertCtrl: AlertController) {
60.
```javascript
this.textViewArrayObj = {};
this.selectViewArrayObj = {};
this.checkBoxViewArrayObj = {};
this.radioViewArrayObj = {};
this.fileViewArrayObj = {};
this.formFields = [];
this.selectViewArray = [];
this.checkBoxViewArray = [];
this.radioViewArray = [];
this.fileViewArray = [];
this.filesByUser = [];
this.todo = {};
this.stepsByProcedure = [];
this.stepsFilledByUserByProcedure = [];
this.procedureStatus = 0;
this.filesToUpload = [];
this.updateProcedureData = {};

this.step = this.params.get('steps');
this.userId = this.params.get('userId');
this.procedureId = this.params.get('procedureId');
this.full_form = this.params.get('full_form');
this.idProcedure = this.params.get('idProcedure');
this.procedureItem = this.params.get('procedure');
this.procedureIdByStep = this.params.get('procedureIdByStep');
this.idUserProcedure = this.params.get('idUserProcedure');
this.userType = this.params.get('userType');
this.userProcedureId = this.params.get('userProcedureId');

console.log(this.step);
console.log(this.userId);
console.log(this.procedureId);
console.log(this.full_form);
console.log(this.idProcedure);
console.log(this.procedureItem);
console.log(this.idUserProcedure);
console.log(this.procedureIdByStep);
console.log(this.userType);
console.log(this.userProcedureId);
this.loadFields(this.step)
this.loadFilesArray(this.userId, this.procedureIdByStep, this.idUserProcedure)
this.getStepsByProcedure();
this.getPreviousStepsRegistered();
this.checkStateProcedure();
```
getStepsByProcedure() {
    this.apiService.getProcedureById(this.procedureIdByStep)
    .then(procedure => {
        for (let i = 0; i < procedure.form_fields.form.length; i++) {
            this.stepsByProcedure.push({
                "name": procedure.form_fields.form[i].name
            })
        }
    })
}

getPreviousStepsRegistered() {
    this.apiService.loadUserStepsByProcedure(this.userId, this.procedureId)
    .then(stepByUser => {
        console.log(stepByUser)
        for (let i = 0; i < stepByUser.length; i++) {
            this.stepsFilledByUserByProcedure.push({
                "name": stepByUser[i].step
            })
        }
    })
}

checkStateProcedure() {
    this.apiService.getUserProcedureInitialByProcedure(this.userId, this.procedureId)
    .then(dataUserProcedureInitial => {
        // procedureInitial status
        // 0 pendiente
        // 1 terminado
        if (dataUserProcedureInitial.length > 0) {
            this.updateProcedureData = {
                active: true,
                procedure: this.procedureId,
                user: this.userId
            }
        }
    })
}

getUserType() {
    return this.userType == "std";
}

fileChangeEvent(fileInput: any) {
    console.log(fileInput)
    this.filesToUpload = <Array<File>> fileInput.target.files;
}

acceptStep() {
    console.log(this.step)
    console.log(this.procedureId)
    console.log(this.userId)
    console.log(this.full_form)
    console.log(this.idProcedure)
    console.log(this.fileViewArrayObj)
    console.log(this.filesToUpload)
    if (!this.procedureItem[0].procedure.procedure_type) {

```javascript
this.showAcceptDialog();
}
else {
  this.showAcceptUploadFile();
}
}
}

rejectStep() {
  console.log(this.step)
  console.log(this.procedureId)
  console.log(this.user)
  console.log(this.fullForm)
  console.log(this.idProcedure)
  console.log(this.fileViewArrayObj)
  console.log(this.filesToUpload)
  this.showRejectDialog();
}

showAcceptDialog() {
  let alert = this.alertCtrl.create({
    title: 'Warning!',
    message: '¿Are you sure you want to accept this step?',
    buttons: [
      {
        text: 'Cancel',
        role: 'cancel',
        handler: () => {
          console.log('Cancel clicked');
        }
      },
      {
        text: 'Ok',
        handler: () => {
          this.util.showLoadingDialog();
          console.log(this.getProcedureState());
          if (this.stepsByProcedure.length == 1) {
            this.approveStepAndUpdateProcedureStatus().then(result => {
              this.util.hideLoadingDialog();
              this.showSuccessDialog();
            }, error => {
              this.util.showLoadingDialog();
              this.util.showErrorAlertError('An unexpected error occurred', 'Error getting data');
            })
          } else if (this.stepsByProcedure.length > 1) {
            if (this.getProcedureState() == 1) {
              this.approveStepAndUpdateProcedureStatus().then(result => {
                this.util.hideLoadingDialog();
              }, error => {
                this.util.showLoadingDialog();
              })
            } else if (this.getProcedureState() == 0) {
```
```
this.approveSingleStep()
  .then(result => {
    this.util.hideLoadingDialog();
    this.showSuccessDialog();
  }, error => {
    this.util.hideLoadingDialog();
    this.util.showAlertError('An unexpected error occurred', 'Error getting data');
  });
}

showAcceptUploadFile() {
  let alert = this.alertCtrl.create({
    title: 'Warning!
    message: '¿Are you sure you want to accept this step?',
    buttons: [
      {
        text: 'Cancel',
        role: 'cancel',
        handler: () => {
          console.log('Cancel clicked');
        }
      },
      {
        text: 'Ok',
        handler: () => {
          this.util.showLoadingDialog()
          console.log(this.getProcedureState())
          if (this.stepsByProcedure.length == 1) {
            console.log("entrata con tamaño = 1")
          }
          this.approveStepAndUpdateProcedureStatusWithFile()
            .then(result => {
              this.util.hideLoadingDialog();
              this.showSuccessDialog();
            }, error => {
              this.util.hideLoadingDialog();
              this.util.showAlertError('An unexpected error occurred', 'Error getting data');
            });
        }
      }]
    };
    alert.present();
  }
}

showAcceptUploadFile() {
  let alert = this.alertCtrl.create({
    title: 'Warning!',
    message: '¿Are you sure you want to accept this step?',
    buttons: [
      {
        text: 'Cancel',
        role: 'cancel',
        handler: () => {
          console.log('Cancel clicked');
        }
      },
      {
        text: 'Ok',
        handler: () => {
          this.util.showLoadingDialog()
          console.log(this.getProcedureState())
          if (this.stepsByProcedure.length == 1) {
            console.log("entrata con tamaño = 1")
          }
          this.approveStepAndUpdateProcedureStatusWithFile()
            .then(result => {
              this.util.hideLoadingDialog();
              this.showSuccessDialog();
            }, error => {
              this.util.hideLoadingDialog();
              this.util.showAlertError('An unexpected error occurred', 'Error getting data');
            });
        }
      }]
    };
    alert.present();
  }
}

showAcceptUploadFile() {
  let alert = this.alertCtrl.create({
    title: 'Warning!
    message: '¿Are you sure you want to accept this step?',
    buttons: [
      {
        text: 'Cancel',
        role: 'cancel',
        handler: () => {
          console.log('Cancel clicked');
        }
      },
      {
        text: 'Ok',
        handler: () => {
          this.util.showLoadingDialog()
          console.log(this.getProcedureState())
          if (this.stepsByProcedure.length == 1) {
            console.log("entrata con tamaño = 1")
          }
          this.approveStepAndUpdateProcedureStatusWithFile()
            .then(result => {
              this.util.hideLoadingDialog();
              this.showSuccessDialog();
            }, error => {
              this.util hideLoadingDialog();
            });
        }
      }]
    };
    alert.present();
  }
}
this.util.showAlertError('An unexpected error occurred', 'Error getting data');
})
} else if (this.getProcedureState() == 0) {

this.approveSingleStepWithFile()
  .then(result => {
    this.util.hideLoadingDialog();

  }, error => {
    this.util.showErrorDialog();
  });

this.util.showAlertError('An unexpected error occurred', 'Error getting data');
}

} } } } } } alert.present();

showRejectDialog() {
  let alert = this.alertCtrl.create({
    title: 'Reject?',
    message: '¿Are you sure you want to reject this step?',
    inputs: [
      { name: 'observations',
        placeholder: 'Observations'
      }
    ],
    buttons: [
      { text: 'Cancel',
        role: 'cancel',
        handler: data => {
          console.log('Cancel clicked');
        }
      }, {
        text: 'Reject',
        handler: data => {
          let sendData = {
            idProcedure: this.userProcedureId,
            observations: data.observations
          }
          console.log(this.getProcedureState())
          this.util.showLoadingDialog()
          this.rejectSimpleStep(sendData)
          .then(result => {
            this.util.hideLoadingDialog();
            this.showSuccessDialog();
          }, error => {
            this.util.showErrorDialog();
            this.util.showAlertError('An unexpected error occurred', 'Error getting data');
          })
        }
      }]},
```javascript
// -----------------------------------------------------------
getProcedureState() {
  if (this.checkIfExists(this.full_form)) {
    this.stepsFilledByUserByProcedure.push({
      "name": this.full_form
    });
  }
  console.log(this.stepsFilledByUserByProcedure)
  for (let i = 0; i < this.stepsByProcedure.length; i++) {
    for (let j = 0; j < this.stepsFilledByUserByProcedure.length; j++) {
      if (this.stepsByProcedure[i].name != this.stepsFilledByUserByProcedure[j].name) {
        this.procedureStatus = 0;
        break
      } else {
        this.procedureStatus = 1;
      }
    }
  }
  return this.procedureStatus;
}

checkIfExists(full_form) {
  for (let i = 0; i < this.stepsFilledByUserByProcedure.length; i++) {
    if (this.stepsFilledByUserByProcedure[i].name == full_form)
      return true
  }
  return false
}

approveSingleStep() {
  return new Promise(resolve => {
    this.apiService.acceptStepByUser(this.idProcedure)
    .then(res => {
      console.log(res)
      if (res) {
        this.showSuccessDialog();
      }
    }, error => {
      resolve({
        status: 400,
        error: "An unexpected error occurred",
        data: "Error getting data"
      })
    });
  });
}

rejectSimpleStep(sendData) {
  console.log("rejectSimpleStep")
  console.log(sendData)
```
```javascript
return new Promise(resolve => {
  this.apiService.rejectStepByUser(sendData)
    .then(res => {
      console.log(res)
      if (res) {
        this.showSuccessDialog();
      }
    }, error => {
      resolve({
        status: 400,
        error: "An unexpected error occurred",
        data: "Error getting data"
      })
    });
});

approveStepAndUpdateProcedureStatus() {
  this.updateProcedureData.status = 1;
  return new Promise(resolve => {
    this.apiService.updateProcedureInitialState(this.userProcedureId, this.updateProcedureData)
      .then(updateResponse => {
        this.apiService.acceptStepByUser(this.idProcedure)
          .then(res => {
            console.log(res)
            if (res) {
              this.showSuccessDialog();
            }
          }, error => {
            resolve({
              status: 400,
              error: "An unexpected error occurred",
              data: "Error getting data"
            })
          });
    }, error => {
      resolve({
        status: 400,
        error: "An unexpected error occurred",
        data: "Error getting data"
      })
    });
  }, error => {
    resolve({
      status: 400,
      error: "An unexpected error occurred",
      data: "Error getting data"
    })
  });
}

rejectStepAndUpdateProcedureStatus(sendData) {
  this.updateProcedureData.status = 1;
  return new Promise(resolve => {
    this.apiService.updateProcedureInitialState(this.userProcedureId, this.updateProcedureData)
      .then(updateResponse => {
        this.apiService.rejectStepByUser(sendData)
          .then(res => {
            resolve(res)
          }, error => {
            resolve({
              status: 400,
              error: "An unexpected error occurred",
```
data: "Error getting data"

approveSingleStepWithFile()
{
  return new Promise(resolve => {
    this.apiService.acceptStepByUser(this.idUserProcedure)
      .then(res => {
        if (res) {
          this.getFileHash(this.filesToUpload[0])
            .then(dataHash => {
                let file: File = this.filesToUpload[0];
                let formData: FormData = new FormData();
                if (file) {
                  formData.append('document', file, file.name);
                  formData.append('field', this.full_form);
                } else {
                  formData.append('document', null, null);
                  formData.append('field', null);
                }
                formData.append('step', this.full_form);
                formData.append('hash', dataHash);
                formData.append('active', true);
                formData.append('user', this.userId);
                formData.append('procedure', this.procedureId);
                formData.append('userProcedure', this.idUserProcedure);
                let headers = new Headers();
                headers.append('Accept', 'application/json');
                let options = new RequestOptions({headers: headers});
                console.log(this.filesByUser[0])
                console.log(this.filesByUser[0].id)
                console.log(formData)
                console.log(JSON.stringify(formData))
                this.apiService.updateFileByUser(this.filesByUser[0].id, formData, options)
                  .then(resFile => {
                    console.log(resFile)
                    if (resFile) {
                      console.log(resFile)
                      if (resFile) {
                        this.showSuccessDialog();
                      } else {
                        this.showErrorDialog();
                      }
                    }
                  })
                console.log(this.filesByUser[0])
                console.log(this.filesByUser[0].id)
                console.log(formData)
                console.log(JSON.stringify(formData))
                this.apiService.updateFileByUser(this.filesByUser[0].id, formData, options)
                  .then(resFile => {
                    console.log(resFile)
                    if (resFile) {
                      console.log(resFile)
                      if (resFile) {
                        this.showSuccessDialog();
                      } else {
                        this.showErrorDialog();
                      }
                    }
                  })
              })
        }
      }
  });
}
```javascript
} else {
    resolve({
        status: 400,
        error: "An unexpected error occurred",
        data: "Error getting data"
    });
}

else {
    resolve({
        status: 400,
        error: "An unexpected error occurred",
        data: "Error getting data"
    });
}

approveStepAndUpdateProcedureStatusWithFile() {
    this.updateProcedureData.status = 1;
    return new Promise(resolve => {
        this.apiService.updateProcedureInitialState(this.userProcedureId, this.updateProcedureData)
            .then(updateResponse => {
                console.log(updateResponse)
                if (updateResponse) {
                    this.apiService.acceptStepByUser(this.idProcedure)
                        .then(res => {
                            if (res) {
                                this.getFileHash(this.filesToUpload[0])
                                    .then(dataHash => {
                                        let file: File = this.filesToUpload[0];
                                        let formData: FormData = new FormData();
                                        if (file) {
                                            formData.append('document', file, file.name);
                                        } else {
                                            formData.append('field', this.full_form);
                                        }
                                    })
                            }
                        })
                }
            })
    });
```
formData.append('docu
ent', null, null);
formData.append('field'
, null);
formData.append('step', thi
s.full_form);
formData.append('hash', dat
aHash);
formData.append('active', t
rue);
formData.append('user', thi
s.userId);
formData.append('procedure'
, this.procedureId);
formData.append('userProced
ure', this.idUserProcedure);
let headers = new Headers();
headers.append('Accept', 'a
pplication/json');
let options = new RequestOp
\tions({headers: headers});
console.log(this.filesByUse
r[0])
console.log(this.filesByUse
r[0].id)
console.log(formData)
console.log(JSON.stringify(
formData))
this.apiService.updateFileByUser(this.filesByUser[0].id, formData, options)
.then(resFile => {
  console.log(resFile
})
  if (resFile) {
    this.showSucces
sdialog();
  } else {
    }, error => {
      resolve({
        status: 400,
        error: "An unex
pected error occurred",
        data: "Error ge
 Booting"}
    })
})
else {
    })
else {
    resolve({
      status: 400,
      error: "An unexpected error occ
urred",
      data: "Error getting data"
    })
})
else {
  resolve({
    status: 400,
    error: "An unexpected error occurred",
    data: "Error getting data"
655.           })
656.       }
657.     }, error => {
658.         resolve({
659.           status: 400,
660.           error: "An unexpected error occurred",
661.           data: "Error getting data"
662.         })
663.     });
664.   })
665. }
666.
667.   // ----------------------------------
668.
669.   getFileHash(file) {
670.     console.log(file)
671.     if (file) {
672.       return new Promise(resolve => {
673.         let hasher = new ParallelHasher('.../assets/libs/md5_worker.js');
674.         return hasher.hash(file)
675.           .then(function (result) {
676.             resolve(result);
677.           });
678.       })
679.     } else {
680.       return new Promise(function (resolve, reject) {
681.         // Save Data
682.         resolve("";
683.       });
684.     }
685.     }
686.
687.   showSuccessDialog() {
688.     this.util.hideLoadingDialog();
689.     let alert = this.alertCtrl.create({
690.       title: 'Successfully entered data!',
691.       buttons: [
692.         { text: 'OK',
693.         handler: () => {
694.           console.log('Cancel clicked');
695.           this.redirectToStepsFilledByUser()
696.         }
697.       }
698.     ]}
699.     )
700.     alert.present();
701.   }
702.
703.   redirectToStepsFilledByUser() {
704.     console.log(this.procedureId)
705.     this.navCtrl.push(StepsResultByUser, {
706.       procedureId: this.procedureId,
707.       userId: this.userId,
708.       idProcedure: this.idProcedure
709.     });
710.   }
711.
712.   loadFilesArray(userId, procedureId, idUserProcedure) {
713.     console.log(userId)
714.     console.log(procedureId)
715.     console.log(idUserProcedure)
716.     this.apiService.getFilesByUserProcedureUser(userId, procedureId
717.          , idUserProcedure)
718.       .then(data => {
console.log(data)

console.log(data.length)

this.filesByUser = data

console.log(this.filesByUser)

if (data.length > 1) {
  console.log("entra 1")
  this.filesByUser = data
} else if (data.length <= 0 || data.length <=1){
  console.log("entra 2")
  this.filesByUser = [data]
}

viewFile(event, file) {
  this.navCtrl.push(PdfViewer, {
    file: file
  });
}

loadFields(fields) {
  for (let i = 0; i < fields.length; i++) {
    this.formFields.push(fields[i])
    if (fields[i].type == "text") {
      this.textViewArray.push(
        this.util.getTextView(fields[i])
      )
    } else if (fields[i].type == "select") {
      this.selectViewArray.push(
        this.util.getSelectView(fields[i])
      )
    } else if (fields[i].type == "checkbox") {
      this.checkBoxViewArray.push(
        this.util.getCheckBoxView(fields[i])
      )
    } else if (fields[i].type == "radio") {
      this.radioViewArray.push(
        this.util.getRadioView(fields[i])
      )
    } else if (fields[i].type == "file") {
      this.fileViewArray.push(
        this.util.getFileView(fields[i])
      )
    }
  }
}

getChecked(fields) {
  if (this.checkBoxViewArray[0].value) {
    for (let i = 0; i < this.checkBoxViewArray[0].value.length; i++) {
      if (fields.text == this.checkBoxViewArray[0].value[i].text) {
        return true;
      }
    }
  }
}

getSelected(fields) {
  if (this.radioViewArray[0].value) {
    for (let i = 0; i < this.radioViewArray[0].value.length; i++) {
      // 
    }
  }
}
if (fields.text == this.radioViewArray[0].value[i].text) {
  return true;
}
}
Observation: {{file.observation}}
49.  console.log(this.params.get('stepInfo'))
50.  console.log(this.params.get('filesInfo'))
51.  }
52.  }
53.  getDateString(date) {
54.    let fullDate = new Date(date);
55.    let twoDigitMonth = fullDate.getMonth() + "";
56.    if (twoDigitMonth.length == 1)
57.      twoDigitMonth = "0" + twoDigitMonth;
58.    let twoDigitDate = fullDate.getDate() + "";
59.    if (twoDigitDate.length == 1)
60.      twoDigitDate = "0" + twoDigitDate;
61.    return twoDigitDate + "/" + twoDigitMonth + "/" + fullDate.getFullYear();
62.  }
63.  }

```
[epaperwork]/
  [src]/
  [pages]/
    [login]/
      login.html

1.  <ion-header>
2.  <ion-navbar>
3.  <ion-title>
4.    e-Paperwork
5.  </ion-title>
6.  </ion-navbar>
7.  </ion-header>
8.  
9.  <ion-content class="ion-md-log-in">
10. </ion-content>
```

```
[epaperwork]/
  [src]/
  [pages]/
    [login]/
      login.ts

1.  import {Component} from '@angular/core';
2.  
3.  import {NavController} from 'ionic-angular';
4.  import {Procedure} from '../procedures/procedure';
```
import {ProceduresFilledByUsers} from "../proceduresFilledByUsers/proceduresFilledByUsers";
import {APIService} from '../../providers/epapperwork-service';
import {Util} from '../../providers/util';

@Component({
  selector: 'page-login',
  templateUrl: 'login.html',
  providers: [APIService, Util]
})

export class LoginPage {
  rootPage: any = LoginPage;
  login = {
    username: 'Staff1',
    password: 'upcadm1n'
  };
  private data: any;
  private userDataLogged: any;
  constructor(public navCtrl: NavController,
               public apiService: APIService,
               public util: Util) {
    this.data = {};
    this.userDataLogged = {};
  }

  doLogin() {
    this.util.showLoadingDialog();
    this.apiService.doLogin(this.login)
       .then(data => {
        this.data = data;
        if (this.data.status == 200) {
          this.util.hideLoadingDialog();
          this.userDataLogged = this.data.data;
          if (this.userDataLogged.user_type.user_type == 'adm') {
            this.navCtrl.setRoot(ProceduresFilledByUsers, {
              userDataLogged: this.userDataLogged
            });
          } else {
            this.navCtrl.setRoot(Procedure, {
              userDataLogged: this.userDataLogged
            });
          }
        } else {
          this.util.hideLoadingDialog();
          this.util.showAlertError('User not found!', 'Fail to Login');
        }
      }, error => {
        this.util.hideLoadingDialog();
        this.util.showAlertError('User not found!', 'Fail to Login');
      });
  }
}
1. `<ion-header>`
2.  
3. `<ion-navbar>`
4.  
5. `<ion-title>`
6.  
7.  
8.  
9. `<ion-title>`
10.  
11.  
12.  
13.  
14.  
15.  
16.  
17.  
18.  
19.  
20.  
21.  
22.  
23.  

```
import {Component} from '@angular/core';

import {NavController, NavParams, LoadingController} from 'ionic-angular';

import {APIService} from '../../providers/epapperwork-service';

@Component({
  selector: 'page-pdviewer',
  templateUrl: 'pdfViewer.html',
  providers: [APIService]
})
export class PdfViewer {
  rootPage: any = PdfViewer;
  menu: any;
  file: any;
  loadingPopup: any;
  constructor(public navCtrl: NavController,
              public apiService: APIService,
              public loadingCtrl: LoadingController, 
              public params: NavParams) {
    this.loadingPopup = this.loadingCtrl.create({
      content: 'Downloading...'
    })
  }
```
110

```javascript
28. });
29. this.loadingPopup.present();
30. this.file = this.params.get('file');
31. this.loadingPopup.dismiss();
32. }
33. }
34. 
35. }
```

```
├── [src]/
├── [pages]/
│   └── [procedure]/
│       procedure.html
```

```html
1. <ion-header>
2. <ion-navbar>
3. <button ion-button menuToggle>
4. <ion-icon name="menu"></ion-icon>
5. </button>
6. <ion-title>Paperwork ({{ getUserLoggedData() }})</ion-title>
7. </ion-navbar>
8. </ion-header>
9. 
10. <ion-content>
11. <ion-refresher (ionRefresh)="doRefresh($event)">
12. <ion-refresher-content pullingIcon="arrow-dropdown"
13. pullingText="Pull to refresh"
14. refreshingSpinner="circles"
15. refreshingText="Refreshing...">
16. </ion-refresher-content>
17. </ion-refresher>
18. 
19. <ion-list>
20. <ion-list-header>
21. Requests to Download
22. </ion-list-header>
23. <ion-item-
24. divider *ngFor="let procedure of proceduresFileIn; let i = index;">
25. <h2 class="list-item-left" item-left
26. (click)="getSteps($event, procedure.form_fields, procedure.id, procedure.procedure_type)"
27. }
28. </h2>
29. <button class="border-less" ion-button item-
30. right="" (click)="showInformation(procedure.description)"">Info
31. </button>
32. </ion-item-divider>
33. </ion-list>
34. 
35. <ion-list>
36. <ion-list-header>
37. Procedures available for {{ getUserLoggedData() }}
38. </ion-list-header>
39. <ion-item-
40. divider *ngFor="let procedure of proceduresFileOut; let i = index;">
41. <h2 class="list-item-left" item-left
42. (click)="getSteps($event, procedure.form_fields, procedure.id, procedure.procedure_type)"
43. }
44. </h2>
45. <button class="border-less" ion-button item-
46. right="" (click)="showInformation(procedure.description)"">Info
47. </button>
```
```typescript
1. import {Component} from '@angular/core';
2. 3. import {NavController, NavParams, MenuController} from 'ionic-angular';
4. 5. import {APIService} from '../../providers/epapperwork-service';
6. import {Util} from '../../providers/util';
7. import {Steps} from '../steps/steps';
8. 9. 10. @Component({
11.   selector: 'page-procedure',
12.   templateUrl: 'procedure.html',
13.   providers: [APIService, Util]
14. })
15. 16. export class Procedure {
17.   18.     public proceduresFileIn: any;
19.     public proceduresFileOut: any;
20.     public menu: any;
21.     procedure = Procedure;
22. 23.     public userLoggedData: any;
24. 25.   constructor(public navCtrl: NavController,
26.                 public apiService: APIService,
27.                 public util: Util,
28.                 public params: NavParams,
29.                 public menuCtrl: MenuController) {
30.       this.userLoggedData = this.params.get('userDataLogged');
31.       this.proceduresFileIn = [];
32.       this.proceduresFileOut = [];
33.       this.menu = menuCtrl;
34.       this.menu.enable(true, 'menu')
35.       this.loadProcedures(this.userLoggedData.user_type.id);
36.   }
37. 38.   getUserLoggedData() {
39.     return this.userLoggedData.user_type.description;
40.   }
41. 42.   doRefresh(refresher){
43.     this.proceduresFileIn = [];
44.     this.proceduresFileOut = [];
45.     this.apiService.loadProcedures(this.userLoggedData.user_type.id)
46.       .then(data => {
47.         for (let i = 0; i < data.length; i++) {
48.           if (data[i].procedure_type == true) {
49.             ...
50.           }
51.         }
52.       })
53.     })
54.   }  ```
```
54.   this.proceduresFileIn.push(data[i])
55.   refresher.complete();
56. } else if (data[i].procedure_type == false) {
57.   this.proceduresFileOut.push(data[i])
58.   refresher.complete();
59. }
60. }
61. });
62. }
63. }
64. loadProcedures(id) {
65.   this.util.showLoadingDialog();
66.   this.apiService.loadProcedures(id)
67.   .then(data => {
68.     this.util.hideLoadingDialog();
69.     for (let i = 0; i < data.length; i++) {
70.       if (data[i].procedure_type == true) {
71.         this.proceduresFileIn.push(data[i])
72.       } else if (data[i].procedure_type == false) {
73.         this.proceduresFileOut.push(data[i])
74.       }
75.     }
76.   });
77. }
78. showInformation(description) {
79.   this.util.showToast(description)
80. }
81. }
82. getSteps(event, steps, procedureId, procedure_type) {
83.   this.navCtrl.push(Steps, {
84.     item: steps,
85.     procedureId: procedureId,
86.     userDataLogged: this.userLoggedData,
87.     procedure_type: procedure_type
88.   });
89. }
90. }
91. }
```
import {Component} from '@angular/core';
import {NavController, NavParams} from 'ionic-angular';
import {APIService} from '../providers/epapperwork-service';
import {Util} from '../providers/util';
import {StepsFilledByUser} from '../stepsFilledByUser/stepsFilledByUser';

@Component({
  selector: 'page-proceduresByUsers',
  templateUrl: 'proceduresByUsers.html',
  providers: [APIService, Util]
})
export class ProceduresByUsers {

  proceduresByUsers = ProceduresByUsers;

  public userLoggedData: any;
  public procedures: any;

  proceduresByUserList: any;
  proceduresByUserResult: any;

  constructor(public navCtrl: NavController,
             public apiService: APIService,
             public util: Util,
             public params: NavParams) {

    this.userLoggedData = this.params.get('userDataLogged');

    this.procedures = this.params.get('procedures');

    this.proceduresByUserList = [];
    this.proceduresByUserResult = [];

    this.getProceduresByUser(this.procedures.user.id)
  }

  getProceduresByUser(user) {

    this.util.showLoadingDialog();
  }
48. this.apiService.getUserProcedureInitial(user)
49.   .then(proceduresByUser => {
50.     console.log(proceduresByUser)
51.   
52.     this.util.hideLoadingDialog();
53.   
54.     for (let i = 0; i < proceduresByUser.length; i++) {
55.       this.proceduresByUserList.push(proceduresByUser[i])
56.     
57.   }, error => {
58.     this.util.hideLoadingDialog();
59.     this.util.showAlertError('An unexpected error occurred', 'Error getting data');
60.   });
61. 
62.   showInformation(description) {
63.     this.util.showToast(description)
64.   }
65. 
66.   getItemBackGroundColor(status) {
67.     if (status == 1) {
68.       return "approved-row";
69.     } else if (status == 2) {
70.       return "rejected-row";
71.     }
72.   }
73. 
74.   getStepsByUserProcedure(event, steps, procedureId, procedureSteps) {
75.     console.log(steps)
76.     console.log(procedureId)
77.     console.log(procedureSteps)
78.     console.log(procedureSteps[0].id)
79.   
80.     this.navCtrl.push(StepsFilledByUser, {
81.       userId: this.procedures.user.id,
82.       item: steps,
83.       procedureId: procedureId,
84.       idProcedure: steps.id,
85.       procedureSteps: procedureSteps,
86.       procedureIdByStep: procedureSteps[0].procedure.id,
87.       userProcedureId: procedureSteps[0].id
88.     });
89.   })
90. 
91. 
92. 
93. 
94. 
95. 

[epaperwork]/
  ├── [src]/
  │   └── [pages]/
  │       └── [proceduresFilledByUser]/
  │          ── proceduresFilledByUser.html

1. <ion-header>
2.   <ion-navbar>
3.     <button ion-button menuToggle>
4.       <ion-icon name="menu"></ion-icon>
5.     </button>
6.   </ion-title>Paperwork List</ion-title>
7.   </ion-navbar>
8. </ion-header>
<ion-content>
    <div padding>
        <ion-segment [(ngModel)="listType"]>
            <ion-segment-button value="professors">
                Professors
            </ion-segment-button>
            <ion-segment-button value="students">
                Students
            </ion-segment-button>
        </ion-segment>
    </div>
    <div [ngSwitch="listType"]>
        <ion-list *ngSwitchCase="'professors'">
            <ion-item *ngFor="let procedureProfessor of resultsProfessorsFilledProcedures; let i = index;" (click)="loadProcedures(procedureProfessor)">
                <h2 class="list-item-left">
                    {{procedureProfessor.user.user.first_name}}
                </h2>
            </ion-item>
        </ion-list>
        <ion-list *ngSwitchCase="'students'">
            <ion-item *ngFor="let procedureStudent of resultsStudentsFilledProcedures; let i = index;" (click)="loadProcedures(procedureStudent)">
                <h2 class="list-item-left">
                    {{procedureStudent.user.user.first_name}}
                </h2>
            </ion-item>
        </ion-list>
    </div>
</ion-content>

[epaperwork]/
    [src]/
        [pages]/
            [proceduresFilledByUser]/
                proceduresFilledByUser.ts

import {Component} from '@angular/core';
import {NavController, NavParams} from 'ionic-angular';
import {APIService} from './providers/epaperwork-service';
import {Util} from './providers/util';
import {ProceduresByUsers} from './proceduresByUsers/proceduresByUsers';

@Component({
    selector: 'page-proceduresFilledByUsers',
    templateUrl: 'proceduresFilledByUsers.html',
    providers: [APIService, Util]
})

export class ProceduresFilledByUsers {
  public studentsFilledProcedures: any;
  public resultsStudentsFilledProcedures: any;
  public professorsFilledProcedures: any;
  public resultsProfessorsFilledProcedures: any;

  proceduresFilledByUsers = ProceduresFilledByUsers;

  public userLoggedData: any;

  constructor(public navCtrl: NavController,
               public apiService: APIService,
               public util: Util,
               public params: NavParams) {
    this.userLoggedData = this.params.get('userDataLogged');
    this.studentsFilledProcedures = [];
    this.resultsStudentsFilledProcedures = [];
    this.professorsFilledProcedures = [];
    this.resultsProfessorsFilledProcedures = [];

    this.loadUserFillByUserProcedures();
  }

  getUserLoggedData() {
    return this.userLoggedData.user_type.description;
  }

  loadUserFillByUserProcedures() {
    this.util.showLoadingDialog();
    this.apiService.getUserFillByUserProcedures().then(data => {
      console.log(data)
      this.util.hideLoadingDialog();
      for (let i = 0; i < data.length; i++) {
        if (data[i].user.user_type.user_type == "std" && data[i].status == 0) {
          this.studentsFilledProcedures.push(data[i]);
          if (!ProceduresFilledByUsers.checkIfExists(this.resultsStudentsFilledProcedures, data[i])) {
            this.resultsStudentsFilledProcedures.push(data[i]);
          }
        } else if (data[i].user.user_type.user_type == "prf" && data[i].status == 0) {
          this.professorsFilledProcedures.push(data[i]);
          if (!ProceduresFilledByUsers.checkIfExists(this.resultsProfessorsFilledProcedures, data[i])) {
            this.resultsProfessorsFilledProcedures.push(data[i]);
          }
        }
      }
    });
  }
}
this.resultsProfessorsFilledProcedures.push(data[i]);
}
}
}, error => {
  this.util.hideLoadingDialog();
  this.util.showAlertError('An unexpected error occurred', 'Error getting data');
});
}
}

static checkIfExists(array, obj) {
  let i = array.length;
  while (i--) {
    if (array[i].user.user.id === obj.user.user.id) {
      return true;
    }
  }
  return false;
}

loadProcedures(procedures) {
  this.navCtrl.push(ProceduresByUsers, {
    procedures: procedures
  });
}

<ion-header>
  <ion-navbar>
    <button ion-button menuToggle>
      <ion-icon name="menu"></ion-icon>
    </button>
    <ion-title>{{getTittle()}} Procedures</ion-title>
  </ion-navbar>
</ion-header>

<ion-content>
  <ion-list>
    <ion-item-divider *ngFor="let procedure of procedures; let i = index;"
      [ngClass]="getItemBackGroundColor(procedure.status)"
    >
      <h2 class="list-item-left" item-left
        (click)="getSteps($event, procedure.steps, procedure.id, procedure.procedures)"
      >
        {{procedure.procedures[0].procedure.name}}
      </h2>
    </ion-item>
    <button *ngIf="checkProcedureType(procedure.procedure_type)"
      ion-buttons ion-button icon-only ion-button item-right
        color="royal" (click)="downloadFile($event, procedure.steps)""}>
  </ion-list>
</ion-content>
<ion-icon name="cloud-download"></ion-icon>

<ion-item-divider>
</ion-item>

<ion-list>
  <ion-item>
    <ion-button class="border-less" ion-button item-right=""
      (click)="showInformation(procedure.procedures[0].procedure.description)"
      >Info
    </ion-button>
  </ion-item>
  <ion-item divider>
  </ion-item>
</ion-list>

import {Component} from '@angular/core';
import {NavController, NavParams} from 'ionic-angular';
import {APIService} from '../../providers/epapperwork-service';
import {Util} from '../../providers/util';
import {StepsInProcessFilledByUser} from '../stepsInProcessFilledByUser/stepsInProcessFilledByUser';
import {ToastController} from 'ionic-angular';
import {MenuController} from 'ionic-angular';
import {Transfer, TransferObject} from '@ionic-native/transfer';
import {File} from '@ionic-native/file';

@Component({
  selector: 'page-proceduresInProcess',
  templateUrl: 'proceduresInProcess.html',
  providers: [APIService, Util]
})
export class ProceduresInProcess {
  public procedures: any;
  public proceduresArray: any;
  public userId: any;
  public menu: any;

  procedure = ProceduresInProcess;

  public state: any;
  public fileTransfer: TransferObject;

  public arrayProcedures: any;
  public finishProcedure: any;
  public inProcessProcedure: any;

  constructor(public navCtrl: NavController,
              public apiService: APIService,
              public util: Util,
              public params: NavParams,
              public toastCtrl: ToastController,
              public menuCtrl: MenuController,
              public transfer: Transfer,
              public file: File) {
    this.finishProcedure = []
  }
}

```javascript
46. this.inProcessProcedure = []
47. this.arrayProcedures =[];
48. this.state = this.params.get('state');
49. this.fileTransfer = this.transfer.create();
50. this.proceduresArray =[];
51. this.procedures =[];
52. this.menu = menuCtrl;
53. this.menu.enable(true, 'menu')
54. this.loadProcedures();
55. this.menu.enable(true, 'menu')
56. loadProcedures() {
57. this.util.showLoadingDialog();
58. this.apiService.getLoggedUser()
59. .then(dataUser => {
60. console.log(dataUser)
61. this.userId = dataUser.userAppId;  
62. if (this.state == 0){
63. this.apiService.getInProcessProcedures(this.userId)
64. .then(data => {
65. this.util.hideLoadingDialog();
66. console.log(data)
67. for(let i = 0; i<data.length; i++){
68. if(data[i].procedures.length > 0){
69. if(data[i].status == 0 || data[i].status = 2){
70. this.procedures.push(data[i])
71.  }
72.  }
73. }, error => {
74. this.util.hideLoadingDialog();
75. this.util.showError('An unexpected error occurred', 'Error getting data');
76.  });
77. } else if (this.state == 1) {
78. this.apiService.getFinishProcedures(this.userId)
79. .then(data => {
80. this.util.hideLoadingDialog();
81. for(let i = 0; i<data.length; i++){
82. if(data[i].procedures.length > 0){
83. this.procedures.push(data[i])
84.  }
85.  }, error => {
86. this.util.hideLoadingDialog();
87. this.util.showError('An unexpected error occurred', 'Error getting data');
88.  });
89. }, error => {
90. this.util.hideLoadingDialog();
91. this.util.showError('An unexpected error occurred', 'Error getting data');
92.  });
93.  }
94.  }
95. }  
96. })
97. })
98. })
99. })
100. })
101. })
102. })
103. })
104. })
105. })
106. })
107. }
```
getInProcessProcedures(userId) {
  this.apiService.getInProcessProcedures(userId)
    .then(data => {
      console.log(data)
      return data
    })
}

getFinishProcedures(userId) {
  this.apiService.getFinishProcedures(userId)
    .then(data => {
      console.log(data)
      return data
    }, error => {
      this.util.hideLoadingDialog();
      this.util.showAlertError('An unexpected error occurred', 'Error getting data');
    });
}

validateFinishProcedure(itemProcedure) {
  let isFinish: any;
  let procedureSteps = itemProcedure.form_fields.form;
  let steps = itemProcedure.steps;
  for (let i = 0; i < procedureSteps.length; i++) {
    for (let k = 0; k < steps.length; k++) {
      if (procedureSteps[i].name == steps[k].step && steps[k].status == 1) {
        isFinish = true;
      } else {
        isFinish = false;
        break;
      }
    }
  }
  return isFinish;
}

validateInProcessProcedure(itemProcedure) {
  let isInProcess: any;
  let procedureSteps = itemProcedure.form_fields.form;
  let steps = itemProcedure.steps;
  for (let i = 0; i < procedureSteps.length; i++) {
    for (let k = 0; k < steps.length; k++) {
      if (procedureSteps[i].name == steps[k].step && steps[k].status == 1) {
        isInProcess = false;
      } else {
        isInProcess = true;
        break;
      }
    }
  }
  return isInProcess;
}

getProcedureBgColor() {
  if (this.state != 0) {
    return 
  }
}
    return 0

    checkProcedureType(procedure_type) {
        if (!procedure_type) {
            return false;
        } else if (procedure_type) {
            return true;
        }
    }

    downloadFile(event, steps) {
        console.log(steps)
        let userProcedureId = steps[0].id;
        this.apiService.getFilesByUserProcedureData(this.userId, userProcedureId)
            .then(data => {
                console.log(data)
                const url = data[0].document;
                console.log(url)
                console.log(this.file)
                this.fileTransfer.download(url, this.file.dataDirectory + 'file.pdf').then((entry) => {
                    console.log('download complete: ' + entry.toURL());
                }, (error) => {
                    console.log(error)
                });
            }, error => {
                this.util.hideLoadingDialog();
                this.util.showAlertError('An unexpected error occurred', 'Error getting data');
            });

        getItemBackGroundColor(status) {
            if (status == 1) {
                return "approved-row";
            } else if (status == 2) {
                return "rejected-row";
            }
        }

        getTittle() {
            if (this.state == 0) {
                return "Pending "
            } else if (this.state != 0) {
                return "Finished 
            }
        }

        showInformation(description) {
            console.log(description)
            console.log(this.procedures)
        }

        let toast = this.toastCtrl.create({
message: description,
duration: 5000

})
tost.present();
}

getSteps(event, steps, procedureId, procedure, procedureSteps) {
  this.navCtrl.push(StepsInProcessFilledByUser, {
    userId: this.userId,
    item: steps,
    procedureId: procedureId,
    idProcedure: procedureId,
    proceduresArray: this.proceduresArray,
    proceduresInProcess: steps,
    procedureSteps: procedureSteps,
    procedureIdByStep: procedure.procedures[0].procedure.id,
    userProcedureId: procedure.procedures[0].id
  });
// if (this.state != 0) {
//  this.navCtrl.push(StepsInProcessFilledByUser, {
//    userId: this.userId,
//    item: steps,
//    procedureId: procedureId,
//    idProcedure: steps.id,
//    proceduresArray: this.proceduresArray,
//    proceduresInProcess: this.proceduresInProcess
//  });
// }
}

1. <ion-header>
2.  <ion-navbar>
3.   <button ion-button menuToggle>
4.     <ion-icon name="menu"></ion-icon>
5.   </button>
6.   <ion-title>List of Steps</ion-title>
7.  </ion-navbar>
8. </ion-header>
9. 
10. <ion-content>
11.  <ion-list>
12.   <ion-item-
13.     divider *ngFor="let steps of stepsList.form; let i = index;">
14.       <h2 class="list-item-left" item-left
15.         (click)="getFields($event, i, stepsList.form, steps.name, steps.order)">
16.         {{steps.name}}
17.       </h2>
18.     </ion-item-divider>
19.  </ion-list>
20. </ion-content>
import {Component} from '@angular/core';

import {NavController, NavParams, AlertController} from 'ionic-angular';

import {FieldsBySteps} from '../fieldsBySteps/fieldsBySteps';

import {APIService} from '../../providers/epaperwork-service';

import {Util} from '../../providers/util';

@Component({
  selector: 'page-steps',
  templateUrl: 'steps.html',
  providers: [APIService, Util]
})

export class Steps {

  public stepsList: any;
  public procedureId: any;
  public userDataLogged: any;
  public procedure_type: any;
  public hashes: any;
  public fileHashes: any;
  public stepsFilledByUserByProcedure: any;

  steps = Steps;

  constructor(public navCtrl: NavController,
              public apiService: APIService,
              public util: Util,
              public params: NavParams,
              public alertCtrl: AlertController) {

    this.stepsList = this.params.get('item');
    this.procedureId = this.params.get('procedureId');
    this.userDataLogged = this.params.get('userDataLogged');
    this.procedure_type = this.params.get('procedure_type');

    this.hashes = [];
    this.fileHashes = [];

    this.stepsFilledByUserByProcedure = [];

    this.getPreviousStepsRegistered();
  }

  getPreviousStepsRegistered() {
    this.util.showLoadingDialog()
    this.apiService.loadUserStepsByProcedure(this.userDataLogged.userAppId,
                                              this.procedureId)
      .then(stepByUser => {
        console.log(stepByUser)
        if (stepByUser.length > 0) {
this.apiService.getUserProcedureInitialByProcedure(this.userDataLogged.userAppId, this.procedureId)
  .then(dataInitial => {
    console.log(dataInitial)
    this.util.hideLoadingDialog()
    for (let i = 0; i < dataInitial.length; i++) {
      if (dataInitial[i].status == 0) {
        for (let i = 0; i < stepByUser.length; i++) {
          console.log(stepByUser[i])
          this.stepsFilledByUserByProcedure.push({
            "name": stepByUser[i].step,
            "status": stepByUser[i].status,
            "order": stepByUser[i].step_order
          })
        }
      } else {
        this.util.hideLoadingDialog()
      }
    }
    checkIfIsAvailableForNextStep(order) {
      // return this.checkPreviousSteps(order);
      // return this.checkActualStep(order)
      // return this.checkLatestSteps(order)
      console.log(this.stepsList)
      console.log(this.stepsFilledByUserByProcedure)
      if (this.stepsList.form.length == 1) {
        if (this.stepsFilledByUserByProcedure.length > 0) {
          let previousObject = null;
          console.log(this.stepsFilledByUserByProcedure)
          for (let i = 0; i < this.stepsFilledByUserByProcedure.length; i++) {
            if (this.stepsFilledByUserByProcedure[i].status == 0) {
              previousObject = this.stepsFilledByUserByProcedure[i];
              break;
            }
          }
          if (previousObject) {
            if (previousObject.status == 0) {
              return {
                code: false,
message: "To perform this step, previous steps must be accepted"
}
} else {
  return {
    code: true,
    message: ""
  }
}
else {
  if (order == 1) {
    return {
      code: true,
      message: "Is Unique Step."
    }
  } else if (order > 1) {
    return {
      code: false,
      message: "To perform this step, previous steps must be accepted"
    }
  }
}
checkStep(order) {
  // valida el paso previo
  let previousStep = order - 1;
  if (previousStep == 0) {
    if(this.checkStateOfStep(order).code){
      return this.checkStateOfStep(order)
    } else {
      return this.checkStateOfStep(order)
    }
  } else if (previousStep > 0){
    return this.checkStateOfStep(order)
  }
}
checkStateOfStep(order) {
  let lastRegisteredStep = 0;
  if (this.stepsFilledByUserByProcedure.length > 0) {
    let sortArray = this.stepsFilledByUserByProcedure.sort(function (a, b) {
      return a.order - b.order;
    });
  }
```javascript
console.log(sortArray)

lastRegisteredStep = sortArray[sortArray.length - 1].order;

let currentObject = null;
for (let i = 0; i < sortArray.length; i++) {
  if (sortArray[i].order == order) {
    currentObject = sortArray[i];
  }
}

console.log(currentObject)

if(currentObject){
  if(currentObject.status == 0){
    console.log("exsiste y esta pendiente")
    return {
      code: false,
      message: "This step is already waiting to be reviewed."
    }
  } else if(currentObject.status == 1){
    console.log("exsiste pero esta aprobado")
    return {
      code: false,
      message: "This step was already approved."
    }
  } else if(currentObject.status == 2){
    console.log("existe pero esta rechazado")
    return {
      code: true,
      message: ""
    }
  } else if( (order - lastRegisteredStep ) > 1 ) {
    console.log("la diferencia")
    return {
      code: false,
      message: "To perform this step, previous steps must be accepted."
    }
  } else {
    console.log("no existe un step de este tipo previamente registrado")
    return {
      code: true,
      message: ""
    }
  }
}
else {
  console.log(lastRegisteredStep);
  console.log("Is First step, empty previous");
  return {
    code: true,
    message: ""
  }
}
showAlreadyRegisteredStepToast(message) {
  this.util.showToast(message)
```
getFields(event, position, steps, step, order) {
    console.log(this.checkIfIsAvailableForNextStep(order))
    if (this.checkIfIsAvailableForNextStep(order).code) {
        let fields = [];
        for (let i = 0; i < steps.length; i++) {
            if (i == position) {
                fields = steps[i];
            }
        }
    }
    this.navCtrl.push(FieldsBySteps, {
        item: position,
        fields: fields,
        step: step,
        procedureId: this.procedureId,
        user: this.userDataLogged,
        order: order,
        procedure_type: this.procedure_type
    });
} else {
    this.util.showToast(this.checkIfIsAvailableForNextStep(order).message)
}
}

1.  <ion-header>
2.    <ion-navbar>
3.      <button ion-button menuToggle>.
4.      <ion-icon name="menu"></ion-icon>
5.    </button>
6.    <ion-title>Steps</ion-title>
7.  </ion-navbar>
8.  </ion-header>
9.  
10.  <ion-content>
11.    <ion-list>
12.      <ion-item divider ngFor="let steps of stepsList; let i = index;" 
13.          [ngClass]="getItemBackGroundColor(steps.status)"
14.          class="list-item-left" item-left
15.          (click)="getFields($event, steps.full_form, steps.step, steps.
16.            status, steps.id)"
17.          >
18.          {{steps.step}}
19.        </ion-item>
20.      
21.      <button class="border-less" ion-button item-
22.        right (click)="showHashInformation(steps.step, steps)">Info</button>
23.    </ion-item-divider>
24.  </ion-list>
25.  
26.  </ion-content>
import {Component} from '@angular/core';

import {NavController, NavParams, ViewController} from 'ionic-angular';

import {APIService} from '../../providers/epapperwork-service';
import {Util} from '../../providers/util';

import {FillProcedures} from '../fillProcedures/fillProcedures';
import {InfoHash} from '../infoHash/infoHash';

@Component({
  selector: 'page-stepsFilledByUser',
  templateUrl: 'stepsFilledByUser.html',
  providers: [APIService, Util]
})
export class StepsFilledByUser {

  public stepsList: any;
  public procedureId: any;
  public userId: any;
  public idProcedure: any;
  public userType: any;
  public procedureIdByStep: any;
  public userProcedureId: any;
  public hashes: any;
  public fileHashes: any;

  stepsFilledByUser = StepsFilledByUser;

  constructor(public navCtrl: NavController, 
              public apiService: APIService, 
              public util: Util, 
              public params: NavParams) {

    // this.stepsList = [this.params.get('item')];
    this.stepsList = this.params.get('procedureSteps');

    console.log(this.stepsList)
    this.procedureId = this.params.get('procedureId');
    this.userId = this.params.get('userId');
    this.idProcedure = this.params.get('idProcedure');
    this.procedureIdByStep = this.params.get('procedureIdByStep');
    this.userProcedureId = this.params.get('userProcedureId');
    console.log("steps filed by user")
    console.log(this.stepsList)
    console.log(this.procedureId)

    console.log(this.procedureId)
    console.log(this.userProcedureId)
    console.log(this.idProcedure)
    console.log(this.procedureIdByStep)

    this.hashes = [];
    this.fileHashes = [];
  
    this.getHashesByStep();
  }
}
getUserType();

getHashesByStep() {
  for (let i = 0; i < this.stepsList.length; i++) {
    this.hashes.push({
      procedureId: this.stepsList[i].procedure.id,
      user: this.stepsList[i].user.id,
      id: this.stepsList[i].id,
      step: this.stepsList[i].step,
      hash: this.stepsList[i].hash,
      created_at: this.stepsList[i].created_at
    });
  }
}

getItemBackGroundColor(status) {
  if (status == 1) {
    return "approved-row";
  } else if (status == 2) {
    return "rejected-row";
  }
}

showHashInformation(stepName, steps) {
  let stepInfo = "";
  let sendData = {};
  for (let i = 0; i < this.hashes.length; i++) {
    if (this.hashes[i].step == stepName && steps.id == this.hashes[i].id) {
      stepInfo = this.hashes[i]
      this.apiService.getFilesByUserProcedureUser(this.hashes[i].user, this.hashes[i].procedureId, this.hashes[i].id)
      .then(dataFileHashByUser => {
        sendData = {
          "stepInfo": [stepInfo],
          "filesInfo": dataFileHashByUser.length > 0 ? dataFileHashByUser : [],
          "observation": [{"observation": steps.observations}
        ]
      });
      this.navCtrl.push(InfoHash, sendData);
    }, error => {
      this.util.hideLoadingDialog();
      this.util.showAlertError('An unexpected error occurred', 'Error getting data');
    };
    }
  }
}

getUserType() {
  this.apiService.getLoggedUser()
  .then(data => {
    this.userType = data.user_type.user_type
  });
}

getFields(event, steps, full_form, status, idUserProcedure) {
  if (status == 0) {
    this.navCtrl.push(FillProcedures, {
      procedure: this.stepsList,
import {Component} from '@angular/core';
import {NavController, NavParams} from 'ionic-angular';
import {APIService} from '../../providers/epaperwork-service';
import {Util} from '../../providers/util';
import {FillProcedures} from '../fillProcedures/fillProcedures';
import {Transfer, FileUploadOptions, TransferObject} from '@ionic-native/transfer';
import {File} from '@ionic-native/file';
import {InfoHash} from '../infoHash/infoHash';

@Component({
  selector: 'page-stepsInProcessFilledByUser',
  templateUrl: 'stepsInProcessFilledByUser.html',
  providers: [APIService, Util]
})
export class StepsInProcessFilledByUser {
  public stepsList: any;
  public procedureId: any;
  public userId: any;
  public procedureIdByStep: any;
  public idProcedure: any;
  public userType: any;
  public userProcedureId: any;
  public fileTransfer: TransferObject;
  public proceduresArray: any;
  public hashes: any;
  public fileHashes: any;

  public stepsFilledByUser = StepsInProcessFilledByUser;

  constructor(public navCtrl: NavController,
              public apiService: APIService,
              public util: Util,
              public params: NavParams,
              public transfer: Transfer,
              public file: File) {
    this.fileTransfer = this.transfer.create();
    this.stepsList = this.params.get('procedureSteps');
    this.procedureIdByStep = this.params.get('procedureIdByStep');
    console.log(this.stepsList)
    this.procedureId = this.params.get('procedureId');
    this.userId = this.params.get('userId');
    this.idProcedure = this.params.get('idProcedure');
    this.userProcedureId = this.params.get('userProcedureId');
    console.log(this.idProcedure)
```javascript
60. console.log("steps filed by user")
61. console.log(this.stepsList)
62. console.log(this.procedureId)
63. this.getUserType();
64. this.hashes = [];
65. this.fileHashes = [];
66. this.getHashesByStep();
67. }
68. getHashesByStep() {
69.   for (let i = 0; i < this.stepsList.length; i++) {
70.     this.hashes.push({
71.       procedureId: this.stepsList[i].procedure.id,
72.       user: this.stepsList[i].user.id,
73.       id: this.stepsList[i].id,
74.       step: this.stepsList[i].step,
75.       hash: this.stepsList[i].hash,
76.       created_at: this.stepsList[i].created_at
77.     });
78.   }
79. }
80. checkProcedureType(procedure_type) {
81.   if (!procedure_type) {
82.     return false;
83.   } else if (procedure_type) {
84.     return true;
85.   }
86. }
87. getItemBackGroundColor(status) {
88.   if (status == 1) {
89.     return "approved-row";
90.   } else if (status == 2) {
91.     return "rejected-row";
92.   }
93. }
94. showHashInformation(stepName, steps) {
95.   console.log(stepName)
96.   console.log(steps)
97.   let stepInfo = ""
98.   for (let i = 0; i < this.hashes.length; i++) {
99.     if (this.hashes[i].step == stepName & & steps.id == this.hashes[i].id) {
100.       stepInfo = this.hashes[i]
101.   }
102.   }
103.   this.apiService.getFilesByUserProcedureUser(this.hashes[i].user, this.hashes[i].procedureId, this.hashes[i].id)
104.       .then(dataFileHash => {
105.         console.log(dataFileHash)
106.         this.navCtrl.push(InfoHash,
107.           "stepInfo": [stepInfo],
108.           "filesInfo": dataFileHash.length > 0 ? data
109.           FileHash : []),
```
getUserType() {
  this.apiService.getLoggedUser()
  .then(data => {
    this.userType = data.user_type.user_type
    }, error => {
      this.util.hideLoadingDialog();
      this.util.showAlertError('An unexpected error occurred', 'Error getting data');
    });
}
this.util.showToast('This step was already processed!')
}

```typescript
import {Component} from '@angular/core';
import {NavController, NavParams, ToastController} from 'ionic-angular';
import {APIService} from '../../providers/epapperwork-service';
import {FieldsBySteps} from '../fieldsBySteps/fieldsBySteps';
import {AlertController} from 'ionic-angular/index';
import {FillProcedures} from '../fillProcedures/fillProcedures';
import {FieldsBySteps} from '../fieldsBySteps/fieldsBySteps';

@Component({
  selector: 'page-stepsResultByUser',
  templateUrl: 'stepsResultByUser.html',
  providers: [APIService]
})
export class StepsResultByUser {
  public stepsList: any;
  public procedureId: any;
  public userId: any;
  public idProcedure: any;
```
    public hashes: any;
    public fileHashes: any;

stepsResultByUser = StepsResultByUser;

constructor(public navCtrl: NavController,
            public apiService: APIService,
            public params: NavParams,
            public alertCtrl: AlertController,
            private toastCtrl: ToastController) {

    console.log("============================")
    console.log("STEPS RESULT")

    // this.stepslist = [this.params.get('item')];
    // console.log(this.stepslist)
    this.procedureId = this.params.get('procedureId');
    this.userId = this.params.get('userId');
    this.idProcedure = this.params.get('idProcedure');

    this.getStepsByUserAndProcedure(this.userId, this.procedureId, this.idProcedure);

    console.log(this.stepslist)
    // this.userId = this.params.get('userId');
    // this.idProcedure = this.params.get('idProcedure');

    // console.log("steps filed by user")
    // console.log(this.stepslist)
    // console.log(this.procedureId)

    this.hashes = [];
    this.fileHashes = [];

    this.getHashesByStep();
    this.getFileHashesByStep(this.procedureId);

    } getStepsByUserAndProcedure(userId, procedureId, idProcedure){
        this.apiService.loadUserProcedures(userId)
            .then(data => {
                for (let i = 0; i < data.length; i++) {
                    if(data[i].id == idProcedure){
                        console.log(data[i])
                        this.stepsList = [data[i]];
                    }
                }
            });

    } getHashesByStep() {
        this.apiService.loadUserProcedures(this.userId)
            .then(dataUser => {
                for (let obj of dataUser) {
                    this.hashes.push({
                        step: obj.step,
                        hash: obj.hash,
                        created_at: obj.created_at
                    });
                }
            });

    } getFileHashesByStep(procedureId) {

}
```javascript
this.apiService.getFilesByUserProcedure(this.userId, procedureId)
  .then(dataUser => {
    for (let obj of dataUser) {
      this.fileHashes.push(
        {
          step: obj.step,
          field: obj.field,
          hash: obj.hash,
          created_at: obj.created_at
        }));
  });

getItemBackGroundColor(status) {
  if (status == 1) {
    return "approved-row";
  } else if (status == 2) {
    return "rejected-row";
  }
}

showHashInformation(stepName) {
  console.log(stepName)
  let alert = this.alertCtrl.create(
    { title: 'Hashes',
      subTitle: this.getAlertList(stepName),
      buttons: ['Ok']
    });
  alert.present();
}

getAlertList(stepName) {
  console.log(stepName)
  console.log(this.hashes)
  let items = "";
  for (let i = 0; i < this.hashes.length; i++) {
    if (this.hashes[i].step == stepName) {
      items = items + "<div>
      this.hashes[i].step + "<br/>
      "<small>" + "Fecha: " + StepsResultByUser.getDateString(this.hashes[i].created_at) + "<br/>
      "<small>" + this.hashes[i].hash + "</small>" +
      "</div>" +
      this.getFileHTMLHasshes() +
      "<hr/>");
    } else {
      items = items + "No registered data";
    }
  }
  return items;
}

getFileHTMLHasshes() {
  let fileHash = this.fileHashes[0]
  return "<div>"
  "Archivos" + "<br/>" +
  "<small>" + fileHash.field + ": " + "<small>" + "<br/>" +
  "<small>" + fileHash.hash + "</small>" +
  "</div>";
}

static getDateString(date) {
  let fullDate = new Date(date);
  console.log(fullDate);
```

let twoDigitMonth = fullDate.getMonth() + "";

if (twoDigitMonth.length == 1)
    twoDigitMonth = "0" + twoDigitMonth;

let twoDigitDate = fullDate.getDate() + "";

if (twoDigitDate.length == 1)
    twoDigitDate = "0" + twoDigitDate;

return twoDigitDate + "/" + twoDigitMonth + "/" + fullDate.getFullYear();

getFields(event, steps, full_form, status) {
    if (status == 0) {
        this.navCtrl.push(FillProcedures, {
            userId: this.userId,
            steps: steps,
            procedureId: this.procedureId,
            full_form: full_form,
            idProcedure: this.idProcedure
        });
    } else {
        let toast = this.toastCtrl.create({
            message: 'This step was already processed!',
            duration: 3000,
            position: 'bottom'
        });
        toast.onDidDismiss(() => {
            console.log('Dismissed toast');
        });
        toast.present();
    }
}

import {Injectable} from '@angular/core';
import {Http, Headers, RequestOptions, Response} from '@angular/http';
import {LoadingController, NavController} from 'ionic-angular/index';
import {Observable} from 'rxjs/Observable';
import {AlertController} from 'ionic-angular';
import {Storage} from '@ionic/storage';
import 'rxjs/add/operator/map';
import 'rxjs/Rx';

@Injectable()
export class APIService {
    public loading: any;
    public userDataObj: any;
    private dataMenu: any;
    private dataProceduresInitial: any;
    private dataProcedures: any;
    private dataUserFillByUserProcedures: any;
    private dataUserProcedures: any;
    private dataUserByUsername: any;
    private dataUserAppById: any;
private dataUserFilesUploads: any;
private dataUserFilesUploadsProcedure: any;
private fileType: any;
private fileRequestByUser: any;
private dataUserFilesUploadsFilesByUser: any;

private data: {};
response: any;
private protocol: any;
private host: any;
private port: any;
private api: any;
private BASE_URL: any;
private flag: any;

public dataUser: any;
private customer: any;
private contract: any;

constructor(public http: Http,
public loadingCtrl: LoadingController,
public alertCtrl: AlertController,
public storage: Storage) {

this.loading = this.loadingCtrl;
this.http = http;
this.data = {};
this.userDataObj = {};

this.protocol = "http://";
this.host = "104.236.76.253";
// this.host = "127.0.0.1";
this.port = ":8000/";
this.api = "/api/";

this.BASE_URL = this.protocol + this.host + this.port + this.api;
this.flag = false;
}

doLogin(userData) {
return new Promise(resolve => {

this.http.post(this.BASE_URL + "token-auth/", userData)
.map(res => res.json())
.subscribe(character => {

this.http.get(this.BASE_URL + "usersAppAuth/" + userData.userName + "/")
.map(res => res.json())
.subscribe(dataUserByUsername => {

this.http.get(this.BASE_URL + "usersApp/" + dataUserByUsername.id + "/")
.map(res => res.json())
.subscribe(dataUser => {

this.dataUserAppById = dataUser;
this.userDataObj = {
id: dataUserByUsername.id,
userAppId: dataUser.id,
token: this.data,
email: dataUserByUsername.email,
first_name: dataUserByUserName.first_name,
last_name: dataUserByUserName.last_name,
phone: dataUserByUserName.phone,
user_type: dataUser.user_type
}
this.setLoggedUser(this.userDataObj)
resolve({
  status: 200,
  error: "",
  data: this.userDataObj
});
}, error => {
  resolve({
    status: 400,
    error: "Usuario o Contraseña incorrectos."
  });
});
}
}, error => {
  resolve({
    status: 400,
    error: "No se encontró el usuario Específico."
  });
});
}

getUserProcedureInitial(idUser) {
  if (this.dataProceduresInitial) {
    return Promise.resolve(this.dataProceduresInitial);
  }
  return new Promise(resolve => {
    this.http.get(this.BASE_URL + "userProcedureInitialViewSet/" + "?user=" + idUser)
      .map(res => res.json())
      .subscribe(data => {
        this.dataProceduresInitial = data;
        resolve(this.dataProceduresInitial);
      });
  });
}
getUserProcedureInitialByProcedure(idUser, idProcedure) {
  if (this.dataProceduresInitial) {
    return Promise.resolve(this.dataProceduresInitial);
  }
  return new Promise(resolve => {
    this.http.get(this.BASE_URL + "userProcedureInitialViewSet/" + "?user=" +idUser + "&procedure=" + idProcedure)
      .map(res => res.json())
      .subscribe(data => {
        this.dataProceduresInitial = data;
        resolve(this.dataProceduresInitial);
      });
  });
}

saveUserProcedureInitial(data) {
  if (data) {
    return this.http.post(this.BASE_URL + "userProcedureInitialViewSet/", data)
      .map(res => res.json())
      .toPromise();
  }
}

getInProcessProcedures(user){
  if (this.dataProcedures) {
    return Promise.resolve(this.dataProcedures);
  }
  return new Promise(resolve => {
    this.http.get(this.BASE_URL + "userProcedureInitialViewSet/" + "?user=" + user)
      .map(res => res.json())
      .subscribe(data => {
        this.dataProcedures = data;
        resolve(this.dataProcedures);
      });
  });
}

getFinishProcedures(user){
  if (this.dataProcedures) {
    return Promise.resolve(this.dataProcedures);
  }
  return new Promise(resolve => {
    this.http.get(this.BASE_URL + "userProcedureInitialViewSet/" + "?user=" + user + "&status=1")
      .map(res => res.json())
      .subscribe(data => {
        this.dataProcedures = data;
        resolve(this.dataProcedures);
      });
  });
}

updateProcedureInitialState(idProcedureInitial, data){
  console.log("-------------------")
  console.log(this.BASE_URL + "userProcedureInitialViewSet/" + idProcedureInitial + "/")
  console.log(data)
  console.log("-------------------")
  if (idProcedureInitial) {
    return this.http.put(this.BASE_URL + "userProcedureInitialViewSet/" + idProcedureInitial + "/", data)
      .map(res => res.json())
206.       .toPromise()
207.     }
208. }
209.
210. getProcedureById(idProcedure){
211.   if (this.dataProcedures) {
212.     return Promise.resolve(this.dataProcedures);
213.   }
214.   return new Promise(resolve => {
215.     this.http.get(this.BASE_URL + "procedures/" + idProcedure + "/")
216.       .map(res => res.json())
217.       .subscribe(data => {
218.         this.dataProcedures = data;
219.         resolve(this.dataProcedures);
220.       });
221.   });
222. }
223.
224. loadProcedures(id) {
225.   this.dataProcedures = null;
226.   if (this.dataProcedures) {
227.     return Promise.resolve(this.dataProcedures);
228.   }
229.   return new Promise(resolve => {
230.     this.http.get(this.BASE_URL + "procedures/?user_type=" + id)
231.       .map(res => res.json())
232.       .subscribe(data => {
233.         this.dataProcedures = data;
234.         resolve(this.dataProcedures);
235.       });
236.   });
237. }
238.
239. loadMenu(user_type) {
240.   if (this.dataMenu) {
241.     return Promise.resolve(this.dataMenu);
242.   }
243.   return new Promise(resolve => {
244.     this.http.get(this.BASE_URL + "menu/" + "?user_type=" + user_type)
245.       .map(res => res.json())
246.       .subscribe(data => {
247.         this.dataMenu = data;
248.         resolve(this.dataMenu);
249.       });
250.   });
251. }
252. }
253.
254. saveProcedure(data) {
255.   if (data) {
256.     return this.http.post(this.BASE_URL + "usersProcedures/", data)
257.       .map(res => res.json())
258.       .toPromise()
259.   }
260. }
261.
262. saveFileForStep(formData, options) {
263.   if (formData) {
264.     return this.http.post(this.BASE_URL + "userFilesUploads/", formData, options)
265.       .map(res => res.json())
266.   }
loadUserProcedures(userId) {
  if (this.dataUserProcedures) {
    return Promise.resolve(this.dataUserProcedures);
  }
  return new Promise(resolve => {
    this.http.get(this.BASE_URL + "userFillByUserProcedures/" + "?user=" + userId)
      .map(res => res.json())
      .subscribe(data => {
        this.dataUserProcedures = data;
        resolve(this.dataUserProcedures);
      });
  });
}

loadUserStepsByProcedure(idUser, idProcedure){
  if (this.dataUserProcedures) {
    return Promise.resolve(this.dataUserProcedures);
  }
  return new Promise(resolve => {
    this.http.get(this.BASE_URL + "userFillByUserProcedures/" + "?procedure=" + idProcedure + ":&user=" + idUser)
      .map(res => res.json())
      .subscribe(data => {
        this.dataUserProcedures = data;
        resolve(this.dataUserProcedures);
      });
  });
}

getUserFillByUserProcedures() {
  if (this.dataUserFillByUserProcedures) {
    return Promise.resolve(this.dataUserFillByUserProcedures);
  }
  return new Promise(resolve => {
    let dataArray = [];
    this.http.get(this.BASE_URL + "userFillByUserProcedures/")
      .map(res => res.json())
      .subscribe(dataFilledProcedures => {
        this.dataUserFillByUserProcedures = dataFilledProcedures;
        resolve(this.dataUserFillByUserProcedures)
      });
  });
}

getFilesByUserProcedure(userId, procedureId) {
  console.log(userId)
  console.log(procedureId)
  if (this.dataUserFilesUploads) {
    return Promise.resolve(this.dataUserFilesUploads);
  }
  return new Promise(resolve => {
    this.http.get(this.BASE_URL + "userFilesUploads/?user=" + userId + ":&procedure=" + procedureId)
      .map(res => res.json())
      .subscribe(dataFilledProcedures => {
        console.log(dataFilledProcedures)
        this.dataUserFilesUploads = dataFilledProcedures;
        resolve(this.dataUserFilesUploads)
      });
  });
}
327. }};  
328. };
329. }
330. 
331. getFilesByUserProcedureUser(userId, procedureId, idUserProcedure) {
332.   this.dataUserFilesUploadsFilesByUser = null
333.   let urlFiles = this.BASE_URL + "userFilesUploads/?userProcedure=" + idUserProcedure + "&procedure=" + procedureId + "&user=" + userId;
334.   console.log(urlFiles)
335.   if (this.dataUserFilesUploadsFilesByUser) {
336.     return Promise.resolve(this.dataUserFilesUploadsFilesByUser);
337.   }
338.   return new Promise(resolve => {
339.     this.http.get(urlFiles)
340.       .map(res => res.json())
341.       .subscribe(dataFilledProcedures => {
342.         console.log(dataFilledProcedures)
343.         this.dataUserFilesUploadsFilesByUser = dataFilledProcedures;
344.         resolve(this.dataUserFilesUploadsFilesByUser)
345.       });
346.   });
347. }
348. 
349. getFilesByUserProcedureData(userId, userProcedureId) {
350.   if (this.dataUserFilesUploadsProcedure) {
351.     return Promise.resolve(this.dataUserFilesUploadsProcedure);
352.   }
353.   return new Promise(resolve => {
354.     this.http.get(this.BASE_URL + "userFilesUploads/?user=" + userId + "&userProcedure=" + userProcedureId)
355.       .map(res => res.json())
356.       .subscribe(dataFilledProcedures => {
357.         console.log(dataFilledProcedures)
358.         this.dataUserFilesUploadsProcedure = dataFilledProcedures;
359.         resolve(this.dataUserFilesUploadsProcedure)
360.       });
361.   });
362. }
363. 
364. getUserById(id) {
365.   if (this.dataUser) {
366.     return Promise.resolve(this.dataUser);
367.   }
368.   return new Promise(resolve => {
369.     this.http.get(this.BASE_URL + "usersApp/" + id + "/")
370.       .map(res => res.json())
371.       .subscribe(dataUser => {
372.         this.dataUserAppById = dataUser;
373.         this.userDataObj = {
374.           id: dataUser.id,
375.           email: dataUser.user.email,
376.           first_name: dataUser.user.first_name,
377.           last_name: dataUser.user.last_name,
378.           phone: dataUser.phone,
379.           user_type: dataUser.user_type
380.         }
381.         this.dataUser = this.userDataObj;
382.         resolve(this.dataUser)
383.       });
384.   });
385.   }};
acceptStepByUser(idProcedure) {
  if (idProcedure) {
    return this.http.put(this.BASE_URL + "userFillByUserProcedu resPut/" + idProcedure + "/", {status: 1})
      .map(res => res.json())
      .toPromise()
  }
}

rejectStepByUser(data) {
  if (data) {
    let sendData = {
      status: 2,
      observations: data.observations
    }
    return this.http.put(this.BASE_URL + "userFillByUserProcedu resPut/" + data.idProcedure + "/", sendData)
      .map(res => res.json())
      .toPromise()
  }
}

getFileTypes() {
  if (this.fileType) {
    return Promise.resolve(this.fileType);
  } else {
    return new Promise(resolve => {
      this.http.get(this.BASE_URL + "fileType/")
        .map(res => res.json())
        .subscribe(data => {
          this.fileType = data;
          resolve(this.fileType);
        });
    });
  }
}

updateFileByUser(idFile, data, options) {
  console.log(idFile)
  console.log(data)
  console.log(options)
  if (idFile) {
    return this.http.put(this.BASE_URL + "userFilesUploads/" + idFile + "/", data, options)
      .map(res => res.json())
      .toPromise()
  }
}

sendFileRequest(formData) {
  if (formData) {
    return this.http.post(this.BASE_URL + "userFileRequest/", formData)
      .map(res => res.json())
      .toPromise()
  }
}

getFileRequestByUser(user) {
  if (this.fileRequestByUser) {
    return Promise.resolve(this.fileRequestByUser);
  }
}
```typescript
return new Promise(resolve => {
    this.http.get(this.BASE_URL + "userFileRequest/?user=" + user)
      .map(res => res.json())
      .subscribe(data => {
        this.fileRequestByUser = data;
        resolve(this.fileRequestByUser);
      });
  });

showLoader() {
    this.loading.create({
      content: 'Please wait...
    });
    this.loading.present();
}

hideLoader() {
    this.loading.dismiss();
}

showAlert(error) {
    let alert = this.alertCtrl.create({
      title: 'Usuario no encontrado!',
      subTitle: 'Error al iniciar sesión',
      buttons: ['OK']
    });
    alert.present();
}
```

```
```javascript
content: 'Loading'
}
showToasts(message)
let toast = this.toastCtrl.create({
  message: message,
  duration: 3000,
  position: 'bottom'
});
toast.present();
showAlertError(title, subTitle)
let alert = this.alertCtrl.create({
  title: title,
  subTitle: subTitle,
  buttons: ['OK']
});
alert.present();
getFileHash(file)
console.log(file)
if (file) {
  return new Promise(resolve => {
    let hasher = new ParallelHasher('../../assets/libs/md5_worker.js');
      return hasher.hash(file) .then(function (result) {
        resolve(result);
      });
  });
} else {
  return new Promise(function (resolve, reject) {
    // Save Data
    resolve('');
  });
}
getStepHash(todo: any) {
  var JSONData = JSON.stringify(todo);
  var uri = "data:application/json;charset=UTF-8," + encodeURIComponent(JSONData);
  console.log(uri);
  console.log(Md5.hashStr(uri));
  return Md5.hashStr(uri);
}
showLoadingDialog() {
  this.loadingPopup.present();
}
hideLoadingDialog() {
  this.loadingPopup.dismiss();
}
getDateString(date) {
  let fullDate = new Date(date);
  console.log(fullDate);
  let twoDigitMonth = fullDate.getMonth() + ";
  if (twoDigitMonth.length == 1)
```
```javascript
89.   twoDigitMonth = "0" + twoDigitMonth;
90.   let twoDigitDate = fullDate.getDate() + "";
91.   if (twoDigitDate.length == 1)
92.       twoDigitDate = "0" + twoDigitDate;
93.   return twoDigitDate + "/" + twoDigitMonth + "/" + fullDate.getFullYear();
94. }
95. 
96. // TEXT VIEW ===================================
97. getTextView(componentObj) {
98.   return {
99.     id: componentObj.id,
100.    label: componentObj.label,
101.   readonly: componentObj.readonly ? "readonly" : "",
102.   required: componentObj.required ? "true" : "false",
103.   type: (componentObj.id == "e_mail") ? "email" : "text",
104.   value: componentObj.value ? componentObj.value : ""
105. }
106. }
107. 
108. // SELECT ======================================
109. getSelectView(componentObj) {
110.   let optionArray = [];
111.   for (let i = 0; i < componentObj.options.length; i++) {
112.       optionArray.push({
113.           value: componentObj.options[i].value,
114.           name: componentObj.options[i].name
115.       })
116.   }
117.   return {
118.     id: componentObj.id,
119.     label: componentObj.label,
120.   readonly: componentObj.readonly ? "readonly" : "",
121.   required: componentObj.required ? "true" : "false",
122.   options: optionArray
123. }
124. }
125. 
126. // CHECKBOX ===================================
127. getCheckBoxView(componentObj) {
128.   let checkBoxArr = [];
129.   let checkBoxArrValue = [];
130.   for (let i = 0; i < componentObj.options.length; i++) {
131.       checkBoxArr.push({
132.           text: componentObj.options[i].name,
133.           checked: false
134.       })
135.   }
136.   if(componentObj.value){
137.       for (let i = 0; i < componentObj.value.length; i++) {
138.           checkBoxArrValue.push({
139.               text: componentObj.options[i].name,
140.               checked: false
141.           })
142.       }
143.   }
144.   return {
145.     id: componentObj.id,
146.     label: componentObj.label,
147.   readonly: componentObj.readonly ? "readonly" : "",
148.   required: componentObj.required ? "true" : "false",
149.   options: checkBoxArr
150.   }
151. }```
getRadioView(componentObj) {
    let radioArr = [];
    let radioArrValue = [];
    for (let i = 0; i < componentObj.options.length; i++) {
        radioArr.push({
            text: componentObj.options[i].name,
            value: componentObj.options[i].value,
            checked: false
        })
    }

    if (componentObj.value){
        for (let i = 0; i < componentObj.value.length; i++) {
            radioArrValue.push({
                text: componentObj.options[i].name,
                checked: false
            })
        }
    }

    return {
        id: componentObj.id,
        label: componentObj.label,
        readonly: componentObj.readonly ? "readonly": "",
        required: componentObj.required ? "true": "false",
        options: radioArr,
        value: radioArrValue
    }
}

getFileView(componentObj) {
    return {
        id: componentObj.id,
        label: componentObj.label,
        readonly: componentObj.readonly ? "readonly": "",
        required: componentObj.required ? "true": "false",
        options: checkBoxArr,
        value: checkBoxArrValue
    }
}

// RADIO ---------------------------------------------------------------

// FILES ---------------------------------------------------------------

Used packages
[epaperwork]/
<table>
<thead>
<tr>
<th>package.json</th>
</tr>
</thead>
<tbody>
<tr>
<td>{</td>
</tr>
</tbody>
</table>
|     "name": "ionic-hello-world",
|     "author": "Ionic Framework",
|     "homepage": "http://ionicframework.com/",
|     "private": true,
|     "scripts": { |
|         "clean": "ionic-app-scripts clean", |
|     } |
| } |
"build": "ionic-app-scripts build",
"ionic:build": "ionic-app-scripts build",
"ionic:serve": "ionic-app-scripts serve"
},
"dependencies": {
  "@angular/common": "2.2.1",
  "@angular/compiler": "2.2.1",
  "@angular/compiler-cli": "2.2.1",
  "@angular/core": "2.2.1",
  "@angular/forms": "2.2.1",
  "@angular/http": "2.2.1",
  "@angular/platform-browser": "2.2.1",
  "@angular/platform-browser-dynamic": "2.2.1",
  "@angular/platform-server": "2.2.1",
  "@ionic-native/core": "^3.10.3",
  "@ionic-native/file": "^3.10.3",
  "@ionic-native/transfer": "^3.10.3",
  "@ionic/storage": "1.1.6",
  "ionic-angular": "2.0.0",
  "ionic-native": "2.4.1",
  "ionicons": "3.0.0",
  "md5-file": "^3.1.1",
  "ng2-file-uploader": "^0.1.4",
  "ng2-pdf-viewer": "^1.0.2",
  "rxjs": "5.0.0-beta.12",
  "sw-toolbox": "3.4.0",
  "zone.js": "0.6.26"
},
"devDependencies": {
  "@ionic/app-scripts": "1.0.0",
  "typescript": "2.0.9",
  "webpack": "^2.4.1",
  "webpack-dev-server": "^2.4.2"
},
"cordovaPlugins": [
  "cordova-plugin-whitelist",
  "cordova-plugin-console",
  "cordova-plugin-device",
  "cordova-plugin-splashscreen",
  "cordova-plugin-statusbar",
  "ionic-plugin-keyboard"
],
"cordovaPlatforms": [],
"description": "ePaperwork: An Ionic project"}