UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

FIB

TRABAJO DE FIN DE MÁSTER

Máster Universitario en Ingeniería Informática

Desarrollo de una aplicación móvil de running sostenible (Eco-running)

Autor:
Ing. XU GUANGYI

Director:
Sr. ALBERT OLIVÁN GONZÁLEZ

Empresa:
Pacer Smart Beat

Ponente:
Dr. DANIEL JIMÉNEZ GONZÁLEZ

Barcelona, 30 de agosto del 2020

Resumen

En la actualidad podemos observar como la tecnología cada vez tiene un papel más protagonista en nuestra vida.

De la aparición de internet han aparición de internet, la tecnología digital ha evolucionado de manera significativa. En la última década han surgido tecnologías como la IA (Inteligencia artificial), *machine learning*, nano-tecnología, IOT (internet de las cosas) y etc. Estas tecnologías han provocado la aparición de nuevas tendencias sociales como *Smart City*, *Smart Home*, *Smart Transport* que están haciendo que nuestras vidas sean mucho más cómoda. Al fin y al cabo, la tecnología ha sido el motor del progreso de nuestra sociedad

El concepto de Pacer Smart Beat, nació por la curiosidad de dos emprendedores y a la vez aficionados al running, los cuales creían que no existía la aplicación "perfecta" para el runner actual. Estamos hablando de una tipología de corredor diferente, un corredor mucho más concienciado con la sostenibilidad, el medioambiente y la alimentación saludable, natural y sostenible.

Pacer Smart Beat, es una aplicación de running, donde el corredor puede registrar su actividad física como otras aplicaciones que existen en el mercado. Pero a más a más, la herramienta integra un market place sostenible, donde el usuario puede comprar cualquier producto de running respetuoso con el planeta. Desde ropa técnica, suplementación o una donación a entidades relacionadas con la sostenibilidad.

Otra de las funcionalidades que diferencian esta aplicación, con la del resto de aplicaciones existentes en el mercado como *Nike Running Club, Strava, Runnkeeper*, Pacer Smart Beat contiene un "buscador de runners", en el cual, puedes conectar con otros runners que están cerca del usuario, para aquellos runners que no les gusta correr solos.

En conclusión, el propósito principal de esta aplicación es cubrir las necesidades de esta tipología de corredores (Eco-runners) y mejorar la experiencia desde un punto de vista más global, donde no solamente es una aplicación para correr, sino que abarca mucho más que la sola experiencia deportiva.

Abstract

Today we can see how technology increasingly plays a leading role in our lives.

From the emergence of the internet to the emergence of the internet, digital technology has evolved significantly. In the last decade, technologies such as AI (Artificial Intelligence), machine learning, nano-technology, IOT (internet of things) and etc. have emerged. These technologies have caused the appearance of new social trends such as Smart City, Smart Home, Smart Transport that are making our lives much more comfortable. After all, technology has been the engine of progress in our society

The concept of Pacer Smart Beat was born out of the curiosity of two entrepreneurs and at the same time running fans, who believed that there was no "perfect" application for the current runner. We are talking about a different type of corridor, a corridor much more aware of sustainability, the environment and healthy, natural and sustainable food.

Pacer Smart Beat is a running application, where the runner can record his physical activity like other applications that exist on the market. But in addition, the tool integrates a sustainable market place, where the user can buy any running product that respects the planet. From technical clothing, supplements or a donation to entities related to sustainability.

Another of the functionalities that differentiate this application from the rest of the existing applications in the market such as Nike Running Club, Strava, Runnkeeper, Pacer Smart Beat contains a "runner search engine", in which you can connect with other runners who are close to the user, for those runners who don't like to run alone.

In conclusion, the main purpose of this application is to meet the needs of this type of runners (Eco-runners) and improve the experience from a more global point of view, where it is not only an application for running, but also covers much more than the single sporting experience.

Tabla de contenidos

Tabla de Ilustración

1. Contexto

1.1 Introducción

Hoy en día, con el desarrollo de tecnología en el campo de móviles inteligentes, podemos encontrar numerosas aplicaciones para móviles, de todas las temáticas, salud, educación, religión, ocio, entretenimiento, finanzas, hábitos…. A demás, los consumidores utilizamos estas aplicaciones en diferentes dispositivos, smartphone, smartwatch, tabletas... Sin embargo, por parte de las empresas y los departamentos de TI, tampoco se quieren gastar tiempo y dinero en desarrollar una aplicación con distintos códigos para cada plataforma. Afortunadamente existen plataformas de lenguaje hibrido que permiten el ahorro de tiempo y dinero, ya que conseguimos una aplicación para los dos principales sistemas operativos con un mismo lenguaje de código.

Bajo mi punto de vista, estas facilidades ayudan a que este tipo de plataformas creen una tendencia alcista en el sector del desarrollo de aplicaciones móviles. Donde cada vez, se ofrecen más soluciones, donde el usuario no requiere conocimientos de programación, y democratizan así, el sector del desarrollo app y web.

Una aplicación híbrida es una aplicación *web* creada con las habilidades informáticas de la *web* y luego empaquetada dentro de un *shell* de aplicación nativa real (android, ios, pwa, etc.). Con este enfoque, se obtiene acceso a todas las diferentes funcionalidades del dispositivo, lo que le permitirá ejecutar su código en todas las plataformas y llegando así, a todo el mercado, independientemente del sistema operativo que utilice su terminal.

Basándose en esta estrategia, la aplicación de Pacer Smart Beat está creada con **Ionic Framework + Angular Language + Capacitor Runtime + Firebase Database**.

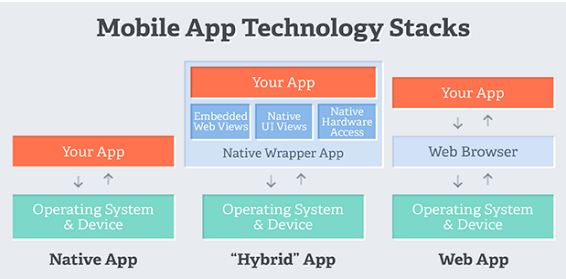


Ilustración 1. Stacks de tecnología móvil

1.2 Motivación

Eco-running es un concepto nuevo, dentro del sector de *running*. Como dice la palabra, esta tendencia fusiona el *running* con la sostenibilidad.

Como hemos definido antes, la inquietud de un *eco-runner* no se finaliza cuando acaba la actividad deportiva, sino que podemos decir que es su estilo de vida. La preocupación de utilizar ropa de *running* sostenible, alimentación de km 0, suplementación deportiva que mitigue la huella de carbono, o utilizar complementos de *running* con materiales reutilizados…

Si observamos el mercado, podemos decir, que actualmente, existen numerosas aplicaciones que nos ayudan a trackear nuestra actividad deportiva, pero cada una de ellas centradas en tipologías diferentes. Las más populares (al menos en el sector de *running*) se centran básicamente en el registro de actividad, mostrando las rutas recorridas, cálculo de calorías gastadas, el tiempo realizado, etc.

Pacer Smart Beat nació para ofrecer una solución aportando un valor añadido, para todos aquellos corredores que no solo quisieran registrar su actividad. Complementado así la experiencia esperada por un *eco-runner*

1.3 Oportunidad de negocio

Observando el mercado, podemos decir que la aplicación hibrida es una tendencia alcista a nivel tecnológico. A nivel social podemos asegurar que la sostenibilidad no solo es una tendencia, sino que debería ser una obligación, por el futuro de la humanidad. A nivel deportivo y salud, la alimentación saludable y la práctica deportiva como el running, ya estaban integradas en nuestra sociedad, pero debido a la pandemia que estamos viviendo estos meses, podemos decir, que aun más han ocupado nuestros hábitos diarios.

Como muestran en los datos, una la investigación reciente de (ESTUDIO DE GRUPO NN, 2019), “la participación en las carreras populares es un factor que también se ha incrementado en los últimos años. Así lo refleja este análisis, que desvela que el 45% de los españoles aficionados al running ha participado en alguna carrera en los últimos doce meses. Entre ellos, un 47% prioriza las carreras urbanas, un 6% se decanta por las de relevos y el 6%

por triatlones. Sin embargo, otro tipo de carreras como las maratones también están en la mente de los corredores, ya que el 22% opta por participar en competiciones de este tipo y el 40% considera afrontar este reto en el futuro."

Hacer deporte solo o acompañado es otra de las cuestiones de interés de esta investigación. Así, un 42% de los corredores de nuestro país prefiere disfrutar del deporte en soledad, mientras que un 40% lo hace en compañía de amigos o familiares. En cuanto a sus lugares favoritos, una mayoría de los encuestados elige la naturaleza y el aire libre, ya que un 35% de ellos menciona el campo mientras que el 30% prefiere correr por vías o caminos establecidos y el 10% por la playa. Por el contrario, hay quienes prefieren los espacios cerrados, puesto que un 12% practica el running en los gimnasios y el 6% en pistas cubiertas.

Una vez más, si analizamos los datos del mercado y los hábitos de nuestra sociedad, podemos llegar a la conclusión de que uno de los sectores con más proyección de futuro es el health-tech, debido a que la esperanza de vida actualmente es más larga y de mayor calidad.

1.4 Análisis de competencia

A nivel de marketing, antes de realizar un proyecto comercial, es indispensable que hagamos análisis e investigaciones a aquellas aplicaciones que brindan iguales o similares funcionalidades en el mercado que las que ofrecemos nosotros. Con estos análisis de competencia, podríamos fortalecer nuestras ventajas con funcionalidades más renovadas basándose en las que ya existen en el mercado, mientras tanto evitar los inconvenientes y posibles errores aprendiendo de los competidores. A continuación, en la Tabla 1 se listan y se definen las partes donde marcamos la diferencia y otras donde podríamos hacer posibles avances.

	PRECIO	CALENDARIO COMPETICIONES	REGISTRO DE ACTIVIDAD	ENTRENAMIENTOS PERSONALIZADOS	MARKET PLACE SOSTENIBLE	BUSCADOR DE RUNENRS	CONSEJOS DIARIOS	COMUNIDAD	MARCA DEPORTIVA
STRAVA	PREMIUM	NO	SI	SI	NO	NO	NO	MUY ALTA	NO
NIKE RUNNING	PREMIUM	NO	SI	NO	NO	NO	NO	MUY ALTA	NIKE
RUNTASTIC	PREMIUM	NO	SI	NO	NO	NO	NO	MUY ALTA	ADIDAS
RUNKEEP	PREMI	NO	SI	NO	NO	NO	NO	MUY	ASICS

ER	UM							ALTA	
ENDOMONDO	GRATIS	NO	SI	NO	NO	NO	NO	ALTA	UNDER ARMOUR
MAP MY RUN	GRATIS	NO	SI	NO	NO	NO	NO	ALTA	NO
PACER	GRATIS	NO	SI	NO	NO	NO	NO	BAJA	NO
RUNEA	PREMIUM	NO	NO	SI	NO	NO	NO	BAJA	NO
FREELETICS	PAGO	NO	SI	SI	NO	NO	NO	ALTA	NO
PUMA TRACK	PREMIUM	NO	SI	NO	NO	NO	NO	ALTA	PUMA
RUN OPINION	GRATIS	SI	NO	NO	NO	NO	NO	ALTA	NO
PACER SMART BEAT	PREMIUM	SI	SI	SI	SI	SI	SI	BAJA	NO

Tabla 1 Table de Análisis de Competencia

1.5 Requerimientos general del proyecto

Los requerimientos generales del proyecto se pueden categorizar de la siguiente forma:

- **Obtención de datos**: la aplicación deberá disponer de un sistema de almacenamiento usando una base de datos no relacional para acumular los datos de perfiles de usuarios.
- **Presentación de datos**: los datos almacenados en BD deberán poder procesarse y presentarse cuando los soliciten los usuarios, tanto para mostrar información personal como para realizar comunicaciones entre ellos. .
- **Sistema de autenticación**: la aplicación usará el sistema de autenticación que provee Firebase y permitirá que los usuarios se identifiquen. También deberá disponer de un sistema de supervisión para vigilar las sesiones de usuario.
- **Tiempo real**: como una aplicación que ofrece servicios para corredores, exigirá al sistema donde montan las funcionnalidades que ejecuten en tiempo real, intentando minimizar la latencia.
- **Multi-plataforma:** la aplicación híbrida obtiene datos para poder compilarse hacia cualquier sistema operativo con configuración minimizada de cada plataforma, por lo tanto, intentará ejecutarse en varias plataformas.

1.6 Actores Implicados

En este apartado se definen los actores principales de este proyecto, es decir, todas las personas que tienen algún tipo de relación con el proyecto.

1.6.1 La empresa

Pacer Smart Beat es una *start-up* que actualmente está naciendo dentro de la incubadora de UPC Emprèn de Terrassa. La compañía se encuentra en la vanguardia de la tecnología intentando realizar el sueño de crear un mundo mejor para los corredores persiguiendo la tendencia tecnológica y la sostenibilidad. Pacer Smart Beat está en un momento inicial donde la página web publicada, la comunicación con *testers* está en marcha, la aplicación se está implementando y el *marketing* sobre las redes sociales se actualiza semanalmente. Actualmente el objetivo principal es tener la aplicación (MVP) con la mayoría de las funcionalidades propuestas hasta que llegue a publicar una versión Beta para que luego participen los *testers*. Cuando tengamos los *feedbacks* de usuarios potenciales. Una vez llegado a ese punto la empresa solicitará inversión financiera con la ayuda de la incubadora de UPC.

1.6.2 El equipo de desarrollo

El equipo de desarrollo en la fase inicial del proyecto está formado por 3 personas con diferentes perfiles profesionales.

- Jefe de proyecto: responsable de la estrategia de *marketing*, la organización de comunicación, la planificación de tereas y la supervisión de todo el proyecto.
- Director de diseño: responsable del deseño gráfico y la implementación de la página web.
- Programador y *UX Designer*: responsable de la implementación de la aplicación entera desde el principio.

Mi rol es un perfil de programador junior, con la libertad de poder aportar nuevas ideas y nuevos procesos en la experiencia de *UX*. A medida que han pasado los meses he podido ganar más peso dentro del equipo.

2. Alcance del Proyecto

2.1 Objetivos

El objetivo principal del proyecto es ofrecer una aplicación para eco-runners, cubriendo todas las necesidades de este nuevo perfil de corredores. Desde un *Marketplace*, hasta un conector con otros *eco-runners*.

La aplicación deberá registrar la información de los usuarios incluyendo, sus estados de salud y sus hábitos de entrenamiento y así describir un perfil para cada usuario. Con ello, los datos acumulados en el base de datos, servirán para hacer análisis y planificaciones personales, establecer comunicaciones con otros corredores, actualizar las geolocalizaciones en tiempo real, etc.

Para llevar a cabo el desarrollo de la aplicación hemos definido los siguientes pasos y objetivos:

- **Definición del proyecto**: definir en qué consiste el proyecto con mayor exactitud posible y planificar cómo llevarlo a cabo en el tiempo establecido
- **Estudio sobre la tecnología a usar**: el aprendizaje y la familiarización con las nueva tecnología ha sido la parte más complicadas del proyecto, sobre todo la aplicación e implementación con un *framework* y un lenguaje de programación que nunca he utilizado. Entre ello, entender el concepto de aplicación híbrida, las relaciones entre el base de código y el proyecto nativo y la selección entre dos marco de *runtime* (Capacitor y Cordova).
- **Diseño de la arquitectura y la estructura del almacenamiento de datos**: La arquitectura de todo el proyecto es un trabajo significativo que ayuda mucho a desarrollar las funcionalidades complejas que se interactúan con otras funciones. Más específicamente, el diseño de la arquitectura de una aplicación híbrida consiste en la selección del marco de *runtime,* la instalación del *framework* de Angular, el diseño de la estructura del flujo de interfaces y la configuración para distintas plataformas. También está incluido la familiarización del funcionamiento del sistema de Cloud Firestore de Firebase para almacenar datos.

- **Implementación de diversas funcionalidades**: La etapa de implementación de la aplicación. Se realizará la base de código visualizando en la *web* y se compilarán en el entorno de Android, a fin de poder realizar un *test* funcional en un dispositivo Android.
- **Test funcional de toda la aplicación**: Etapa de ejecución de la aplicación. Se realizarán una serie de *tests* funcionales a fin de poner a prueba cada funcionalidad. Cuando se aprueben todas las funcionalidades, se acabará siendo un producto Beta de la empresa.
- **Evaluación personal**: Valoración de la aplicación y resumen de las posibles mejoras y errores durante la implementación para aplicar a los futuros proyectos.
- **Escritura de la memoria**: Realización de la memoria según normativa y requisitos a fin de presentar el proyecto en público
- **Presentación del proyecto**: Defensa del proyecto delante del tribunal.

  *****En la sección "2.3 Planificación" se puede consultar la duración estimada de la realización de dichos objetivos.

2.2 Descripción general

Se propone como Trabajo de Fin de Máster (TFM) el diseño e implementación de una aplicación híbrida, que contribuya a solucionar las necesidades de los llamados eco-runners.

La aplicación está creada para dar soluciones y respuestas a las cifras mencionadas anteriormente sobre la actividad deportiva como el running.

Gracias a esta aplicación, podremos acceder una Marketplace, donde todo corredor consciente con su salud y el planeta puede necesitar.

Esta aplicación también está pensada para aquellos corredores que no les guste salir a correr solos, pudiendo conectar con otros usuarios de la aplicación para compartir la actividad.

El registro de actividad, también es un parte fundamental de aplicación, donde el corredor puede observar y analizar todos sus datos generados en sus entrenamientos.

La historia que hay detrás del nombre de la empresa, tiene una explicación curiosa, ya que, en el mundo del *running*, *una libre* (Pacer), es una persona que guía al corredor marcando el ritmo controlado durante la competición, para alcanzar la meta a un ritmo establecido previamente según la preparación física de éste, sin frenazos ni acelerones (Pacer

Smart Beat), que pueden provocar la no finalización de la competición. Y, en definitiva, el equipo, se propone ser la libre de todo aquellos corredores o aficionados del running que se descarguen la app. Ser una herramienta de referencia que guie al corredor a sus éxitos deportivos, de una manera saludable, controlada y sobre todo respetando el medio ambiente.

Finalmente, el idioma de esta aplicación es el español, en la primera versión, ya que el mercado al que hemos planeado lanzar la versión Beta es España.

A medida que la empresa escale y el público objetivo salga de nuestras fronteras, se traducirá a los idiomas necesarios para ello.

2.3 Planificación temporal

La planificación corresponde a la fase inicial del proyecto de la empresa. Desde el punto de vista de la empresa, se han preparado conscientemente los siguientes pasos para que todo el proyecto se avance en el tiempo establecido:

- Fase inicial. (marzo' 20)
  - Ideación: fase de conceptualización de la idea y comprobación las soluciones existentes del mercado.
  - Análisis: estudio de mercado y del consumidor ideal de la aplicación.
  - Construcción del equipo: búsqueda del equipo humano más idóneo para lleva a cabo un MVP y alineado con los objetivos de la empresa.

- Fase de construcción. (mazo-octubre '20)

Construcción de la primera versión de aplicación. Dentro del MVP se desarrollan las funcionalidades básicas para recibir el feedback del consumidor lo más rápido posible. Las funcionalidades desarrolladas son: registro a la app, menú principal, buscador de runners, Marketplace de marcas de running sostenibles y registro de actividad deportiva.
Mientas tanto el equipo de diseño y UX, diseñan una interface atractiva y fácil de utilizar para el consumidor, siguiendo pautas establecidas en aplicaciones móviles.

- Fase de cocración. (noviembre '20)

Una vez finalizada la primera versión de la aplicación, es el momento de recibir el feedback real del consumidor. Se deja testar a un grupo muy concreto de usuarios la versión beta de la

app. Con ello se intenta iterar la aplicación y co-crear de nuevo las funcionalidades construidas en ella. Así como la valoración de nuevas ideas propuestas por los usuarios.

- Fase de iteración. (diciembre '20)

Una vez analizado la reacción de los usuarios a la aplicación, se realizan los cambios necesarios para realizar la segunda versión de la aplicación.

*Esta fase se repetirá tantas veces fueran necesarias para realizar una aplicación que cubriera completamente las necesidades del usuario final.

- Fase final. (enero '21)

Lanzamiento al mercado de la aplicación. Mediante herramientas de marketing y un presupuesto destinado a ello, la aplicación estará disponible en todas las plataformas de descarga y para todos los móviles.

2.4 Diagrama de Gantt

El diagrama de Gantt muestra las diferentes fases del proyecto y las tareas principales de cada una de ellas. La implementación de la aplicación corresponde a la fase de construcción, el bloque azul mostrado en la *Ilustración 2*.

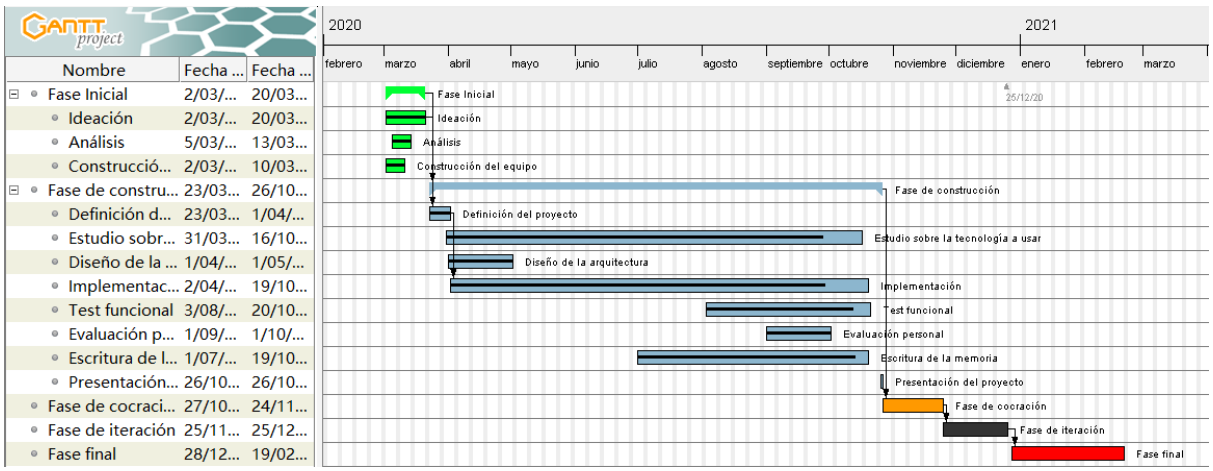


Ilustración 2. Diagrama de Gantt

2.5 Posibles obstáculos

Los principales obstáculos que se pueden encontrar durante el proyecto son:

- Comunicación y coordinación con el resto de miembros del proyecto

El hecho de no poder trabajar en el mismo espacio con el resto de miembros debido a la situación de pandemia puede dificultar la comunicación y coordenación, por lo que puede resultar retrasos o no sincronización entre la implementación de funcionalidad y el diseño.

- Posibles incompatibilidad entre *softwares* de distintas versiones

  La realización de una aplicación híbrida exige la alta compatibilidad entre distintas capas y los *frameworks* dentro de ellas, lo que puede resultar la falla de algunas funcionalidades colaboradas con tercera plataforma debido a la incompatbilidad de versión.

- Implementación de terceras

  El coste de aprendizaje en utilizar códigos implementados por terceras personas puede ser muy elevado debido a la carencia de documentación oficial, o otras documentaciones y comunidades poca detalladas.

- Prueba en un entorno real

  La falta de dispositivo físico correspondiente a varias plataformas y el entorno de simulador poco estable pueden dificultar la prueba de funcionalidades en un entorno real. La interfaz también puede variar incluso en los dispositivos de la misma plataforma.

3. Análisis de requerimientos

3.1 Requerimientos funcionales.

Las funcionalidades de la primera versión de la app (MVP) serán:

- **Sistema de autenticación**

  La aplicación debe registrar a nuevos usuarios y almacenar sus datos personalizados para construir sus perfiles de salud.

- **Configuración de Google Maps**

  Con la ayuda de Google Maps APIs, la aplicación debe poder demostrar las localizaciones de los usuarios y actualizarlas a tiempo real para conectar con corredores cercanos. También pueden visualizar todos los usuarios en toda España. Se ha contado con la privacidad del usuario, y en todo momento se puede decidir estar visible o no para otros usuarios.

- ***Chat* privado**

  La aplicación debe poder enviar mensajes privados a los corredores cercanos con los que quieras conectar. Los mensajes se almacenan separadamente del mapa y se pueden borrar.

- **Comercio electrónico**

  La aplicación ofrece un *marketplace* de productos relacionados con el running como pueden ser alimentación, suplementación, ropa técnica, etc. Todo ellos con la colaboración con otras empresas para ofrecer descuentos especiales a los usuarios.

3.2 Requerimientos no funcionales

Los requerimientos no funcionales son aspectos sobre la propia aplicación y las condiciones con las que se han de realizar las funciones. Dicho de otra forma, son las características de calidad que hay que tener en cuenta en el momento de diseñar la aplicación.

3.2.1 Disponibilidad

La aplicación está pensada para estar disponible las 24 horas del día. Mientras la aplicación está funcionando no hay limitación de tiempo ya que debe poder permitir a los usuarios salir a correr en cualquier momento y la base de datos está preparada para ir almacenando y actualizando la información del usuario, tanto la localización como de la comunicación privada entre los runners.

3.2.2 Escalabilidad

Es uno de los requerimientos no funcionales más importantes. Con escalabilidad nos referimos a nivel funcional. La aplicación está hecha para poder crecer con nuestros usuarios recibiendo *feedbacks* y mejorando las funcionalidades que se ofrecen. Cabe recordar que se trata de un producto propio de la empresa que se quiere comercializar en el futuro, ha de ser un producto ampliable para poder satisfacer a las necesidades nuevas de los usuarios sin modificar la base que se ha desarrollado hasta ahora. Co-creación con el usuario.

El hecho de ser una aplicación híbrida, la alta escalabilidad es una de las ventajas principales que posee ya que solamente tiene una base de código para diversas plataformas en el mercado, hace más fácil su modificación y adaptación con nuevas funcionalidades.

3.2.3 Usabilidad

Ha de ser una aplicación muy fácil de utilizar y el coste de aprendizaje para los usuarios ha de ser muy bajo, ya que no tienen obligación ni paciencia de tener conocimiento fuera de la normalidad y el hábito para usar una aplicación. Es un concepto que la empresa tiene en cuenta al ahora de desarrollar cualquier características de la app así se simplifica al máximo todos los productos a fin de ser amigable al usuario. Un buen ejemplo será que el menú principal que ha diseñado es similar a la interfaz de *smart watch*, según el hábito de los gestos al mover el dado en la pantalla.

3.3 Tecnologías y herramientas utilizadas

3.3.1 Frontend: Ionic / Angular

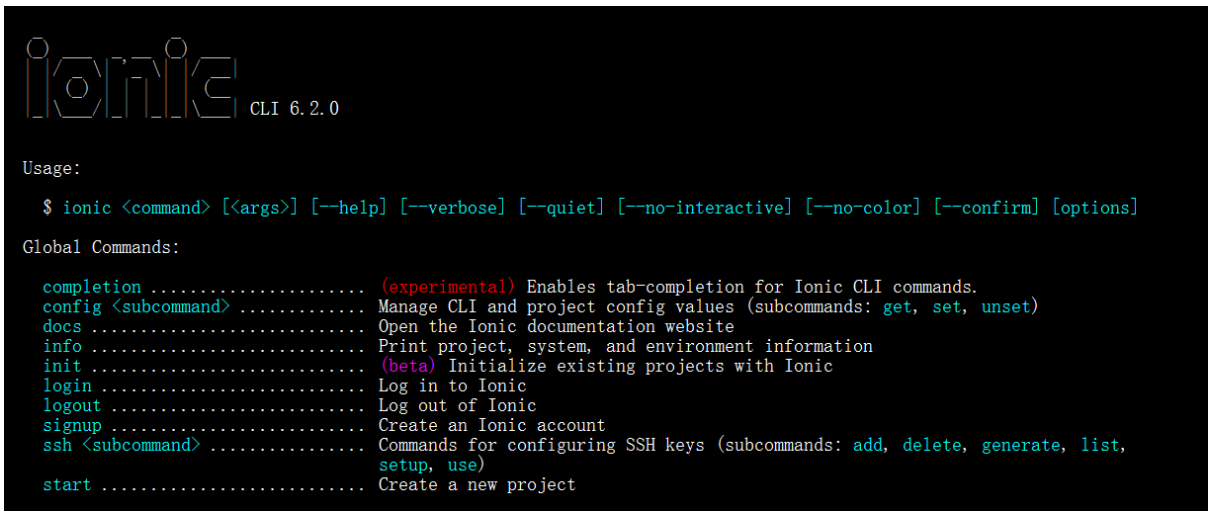


Ilustración 3. Interfaz de Ionic CLI

Según la definición de (Drifty, Inc, 2016), *Ionic Framework* es un conjunto de herramientas de interfaz de usuario de código abierto para crear aplicaciones móviles y de escritorio de alta calidad y rendimiento utilizando tecnologías *web (HTML, CSS y JavaScript)* con integraciones para marcos populares como Angular y React. Su interfaz de comando la podemos visualizar en la *Ilustración 3*.

Como una herramienta de *frontend*, sus beneficios y características son los siguientes:

- Una base de código, que se ejecuta en todos dispositivos con todas plataformas
- Un enfoque en el rendimiento
- Diseño limpio, sencillo y funcional
- Nativo y web optimizado
- Actualmente es compatible con Javascript, Angular, React, Vue

Según la definición de (Angular Team, 2019), Angular es un marco de diseño de aplicaciones y una plataforma de desarrollo para crear aplicaciones de una sola página eficientes y sofisticadas. Forma una buena combinación con Ionic, entre sus beneficios y características están:

- *Cross platform* (Progressive Web Apps, Native, Desktop-installed apps)
- Alta velocidad y rendimiento
- Cree rápidamente vistas de interfaz de usuario con una sintaxis de plantilla sencilla y potente

3.3.2 Backend: Firebase

Ilustración 4. Interfaz de Firebase

Según la definición de (Firebase, Inc., 2016), Firebase es una plataforma en la nube, constituida por un grupo de herramientas que hacen más fácil el desarrollo de aplicaciones para los dispositivos móviles y web. Firebase proporciona funciones como estadísticas, bases de datos, informes de fallas y mensajería, de manera más eficiente. Utiliza la infraestructura de Google y se escala automáticamente, incluso para las apps más grandes.

Firebase es compatible con JavaScript (tanto en el Frontend como con Node), iOS (Swift y Objective C), Android (Java), Python, C++ y Unity.

- **Sistema de autenticación**

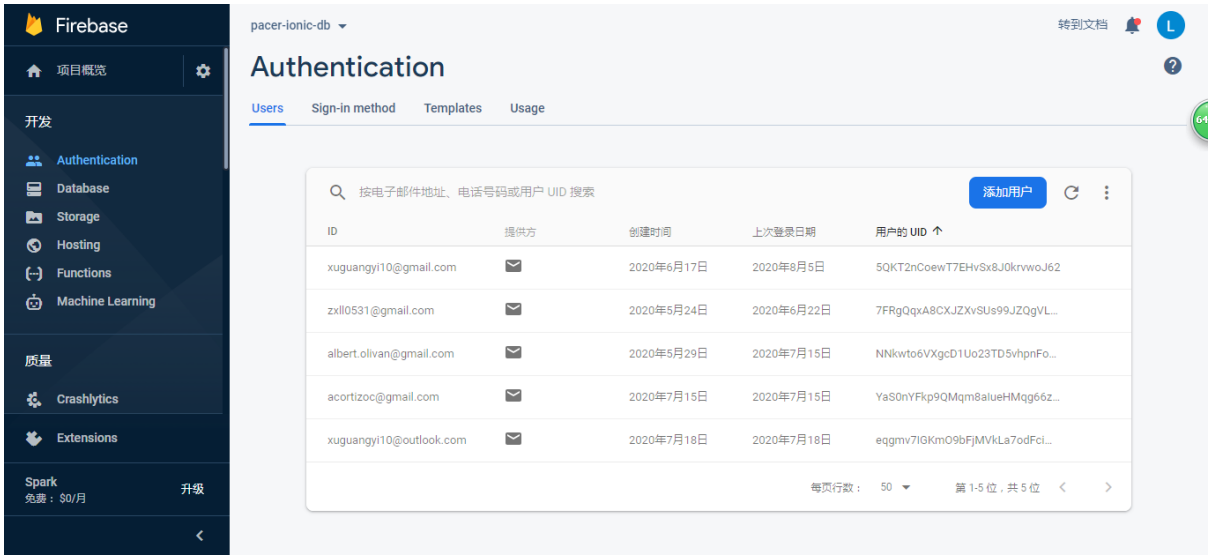


Ilustración 5. Interfaz del sistema de autenticación de Firebase

Para tener un mejor control y servicio de los usuarios, claramente el paso de crear cuenta/iniciar sesión es imprescindible para las mayorías aplicaciones. Según la presentación de (Firebase Auth, 2016), el sistema de autenticación de Firebase proporciona servicios de *backend*, *SDKs* con mucha variedad y bibliotecas de *IU* ya elaborados para autenticar los usuarios de la aplicación. Además de permitir la manera tradicional de autenticación mediante correo y contraseñas, también a los proveedores de identidad federada populares, como Google, Facebook y Twitter, y mucho más. Es fácil empezar a configurar desde cero ya que Firebase aprovecha estándares de la industria como *OAuth 2.0* y *OpenID Connect*, los cuales algunos programadores ya conocemos en otros proyectos o estudios.

- Cloud Firestore Database

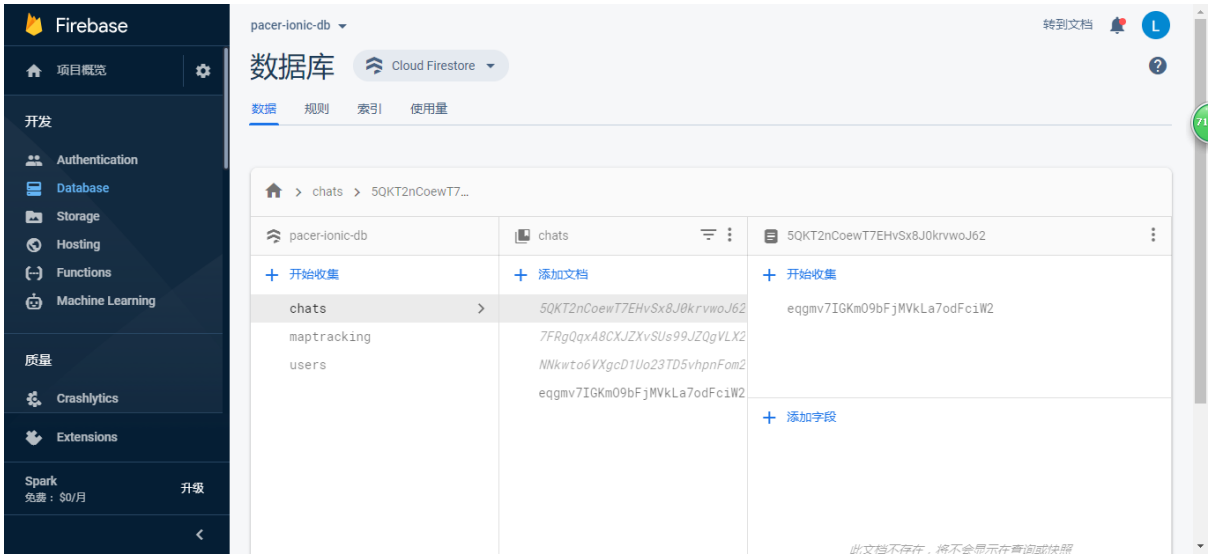


Ilustración 6. Consola de Cloud Firestore

Respecto al base de datos, actualmente existen dos versiones de base de datos en Firebase: Realtime Database y Cloud Firestore Database.

- **Cloud Firestore** es la base de datos más reciente de Firebase para el desarrollo de apps para dispositivos móviles. Aprovecha lo mejor de Realtime Database con un modelo de datos nuevo y más intuitivo. Con Cloud Firestore también se pueden realizar consultas más ricas y rápidas, y el escalamiento se ajusta a un nivel más alto que Realtime Database.
- **Realtime Database** es la base de datos original de Firebase. Es una solución eficiente y de baja latencia destinada a las apps para dispositivos móviles que necesitan estados sincronizados entre los clientes en tiempo real.

Con la recomendación de (Elige una base de datos: Cloud Firestore o Realtime Database, 2016), al final hemos elegido Cloud Firestore Database para mi proyecto con los siguientes motivos:

- Necesito realizar interacciones complejas con los datos que hemos coleccionado, como las funciones de chatear o las de comercio electrónico que se implementará en el futuro.
- Prefiero un modelo de datos con colecciones estructuradas y documentos listados.
- Necesito que tenga capacidades de consultas sofisticadas en datos locales cuando el usuario no tiene conexión.
- La aplicación enviará un flujo de actualizaciones relativamente grandes.

- Aunque en la parte de buscador de runners se exige una disponibilidad bastante cr fica, la cual es la ventaja de realtime database, veo que el firestore tampoco pierde mucho en este campo.

En la secci ón “5.1 Arquitectura de base de datos” se puede consultar m ás detalles sobre Cloud Firestore Database y las interacciones dentro de él.

➢ Almacenamiento

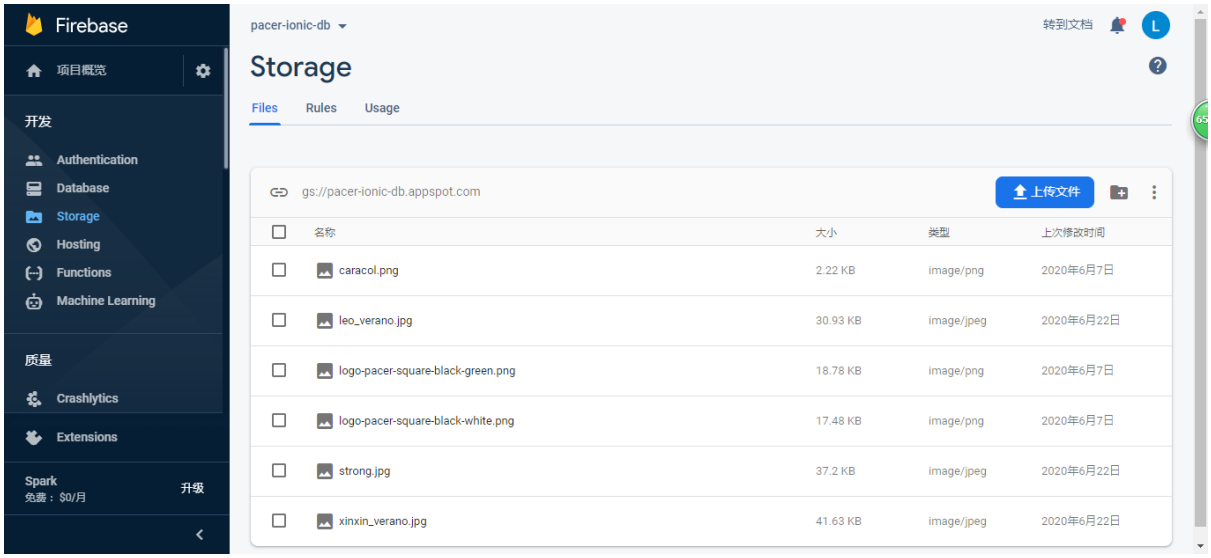


Ilustración 7. Interfaz del almacenamiento de Firebase

Cloud Storage de Firebase es un servicio de almacenamiento con el que los desarrolladores podemos almacenar y entregar los contenidos generado por usuarios, como audios, videos o im ágenes. En mi caso, se usa para guardar los avatares que van subiendo y cambiando los usuarios.

➢ Cloud Messaging

Ilustración 8. Interfaz de Cloud Messaging

FCM (Firebase Cloud Messaging) es un servicio de Firebase con el que se puede publicar notificaciones personalizadas. En la configuración de notificación, aparte de editar los títulos y contenidos de una publicidad, también se puede definir el tipo de notificación, arreglar el tiempo de envío (puede ser una vez o un ciclo de un tiempo específico), seleccionar a los usuarios específicos quienes reciban la notificación, etc. Además, para enviar una notificación a un dispositivo, se requiere un *FCM token* que explicaré cómo se consigue más adelante.

3.3.3 Runtime: Capacitor

Runtime system, o en español sistema en tiempo de ejecución, es un tipo de software que provee el entorno donde ejecuta el programa. *Runtime* puede administrar la memoria de la aplicación, ofrecer la forma de acceder a las variables, interactuar con el sistema operativo, etc.

Existen dos tipos de *native runtime* integrados en *Ionic Framework*: Cordova y Capacitor. Reconozco que al inicio del proyecto y también durante el proceso, me encontré con muchas ocasiones donde no sabía cómo implementar las funcionalidades concretas integradas con Capacitor, mientras tanto usé plugins de Cordova. Suele suceder el problemas de compatibilidad y conflictos. Para aclarar las dudas, en este apartado explicaré específicamente las diferencias entre ellos y sus relaciones con la ayuda del artículo de (Max Lynch (CEO y co-fundador de Ionic), 2019).

Según el resumen de (Cordova Team, 2015), *Apache Cordova* es un marco de desarrollo de aplicaciones móviles de código abierto que transforma *HTML / CSS / JS* estándar en aplicaciones nativas completas. Su equipo lo creó en el año 2008. Proporciona una *API* de JavaScript para acceder a la funcionalidad nativa del dispositivo, como la cámara o el acelerómetro. Cordova contiene las herramientas de compilación necesarias para empaquetar aplicaciones web para iOS, Android y Windows Phone.

(Capacior:Cross-Platform native runtime for web apps, 2020) es un *runtime* de aplicaciones multiplataforma de código abierto que permite que las aplicaciones basadas en web se ejecuten de forma nativa en iOS, Android, Electron y la web. El equipo de Ionic lanzó la primera versión de capacitor en mayo del año 2019. Con Ionic Capacitor, se puede agregar

funcionalidad nativa a nuestras aplicaciones fácilmente utilizando una *API* de complemento simple para Java en Android, JavaScript para la web y Swift en iOS. Podemos decir que Capacitor es un sucesor espiritual de *Apache Cordova y Adobe PhoneGap*.

Al hacer la comparación entre los dos, se puede detectar dos puntos esenciales que marcan la diferencia: gestión de *Native Project* y gestión de *Plugin*.

- *Native Project*

En comparación con Cordova, Capacitor nos ofrece un mejor "tratamiento" a proyectos nativos que están generados desde la base de código que escribimos con javascript (o typescript), es decir, cuando necesitamos interactuar con el proyecto generado por cierta plataforma, podemos trabajar con el código nativo de Android o IOS directamente sin hacer falta cambiar la base de código. Obviamente esta separación nos facilita mucho cuando un equipo de móvil trabaja con otro equipo de *web*. No existirá conflictos cuando quieren hacer modificaciones sobre el mismo proyecto ya que cada equipo puede trabajar en paralelo.

Sin embargo, si es un proyecto individual, nos exige la habilidad experta tanto en código de *web* con javascript(o typescript) como en código de cada plataforma(Android, IOS, Electron)

Normalmente un proyecto usando Cordova tiene una capa abstracta que es el fichero *config.xml* donde guardan todas las configuraciones(*widget*) para la plataforma de Android y de IOS. También es donde podemos modificar el nombre de la aplicación. En cambio, un proyecto usando Capacitor no tiene esa capa, nos exige utilizar el *IDE* (entorno de desarrollo integrado) específico de cada plataforma, Xcode para IOS y Android Studio para Android. Así nos dará más control y visibilidad de los proyectos nativos mientras tanto nos exige la habilidad de las dos partes.

- *Plugin*

Los plugins en Capacitor permiten que JavaScript interactúe directamente con las API nativas. Y cuando uno quiere crear su propio plugin para un uso especial, Capacitor tiene unos comandos del plugin generación bastantes sencillos. Genera automáticamente *hooks* de JavaScript en el servidor de cliente, por lo que la mayoría de los plugins solo necesitan crear uno de Java para Android y otro de Swift / Obj-C para iOS.

Sin embargo, aunque el equipo de Ionic ha declarado que la mayoría de los plugins de Cordova van a seguir siendo compatibles dentro del marco Capacitor 2.0, es inevitable que mezclemos los plugins de ambos marcos porque algunos de Cordova que faltan la compatibilidad son bastante necesarios para algunos proyectos y Capacitor nos ofrecen sus propios plugins, por ejemplo, cordova-plugin-fcm para *push* notificación, cordova-plugin-firebase para configurar la conexión con Firebase y etc según (Known Incompatible Cordova Plugins - Capacitor, 2020) .

3.3.4 Google Maps Platform

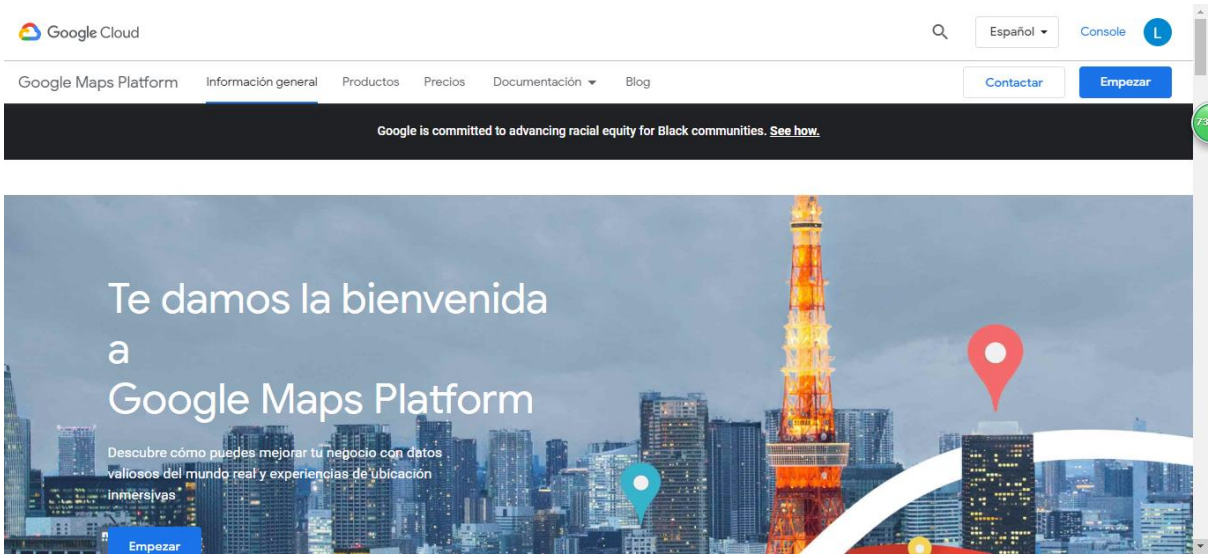


Ilustración 9. Interfaz de Google Maps Platform

Google Maps Platform es una rama de *Google Cloud*, servicios de cloud computing con mucha variedad, ofreciendo servicios de geolocalización y muchos APIs relativos. Con la ayuda de (APIs de geolocalización, 2011), no solamente se puede conectar a *Google Map*, sino también nos permite visualizar las informaciones de un sitio específico, indicar las ubicaciones de los usuarios, calcular las distancias entre ellos y etc.

3.3.5 Android Studio

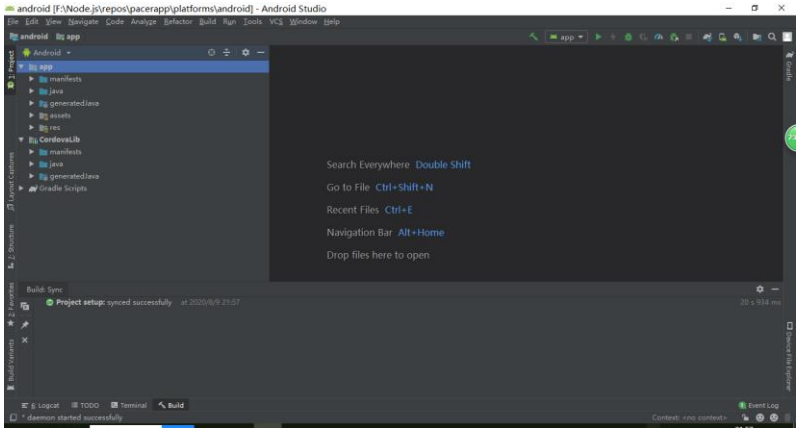


Ilustración 10. Interfaz de Android Studio

Como ya estamos familiarizados, Android Studio es el entorno de desarrollo integrado (*IDE*) oficial para el desarrollo de aplicaciones para Android, cuya interfaz la podemos ver en la *Ilustración 10*. Aquí lo utilizamos para ejecutar el proyecto conectando un dispositivo Android. Para conseguir eso, hay que recorrer unos pasos con el objetivo de convertir la *web app* en una aplicación nativa con la plataforma Android. Según las instrucciones de (Ionic Team, 2019), los pasos de implementación son los siguientes:

Para un proyecto capacitor:

1) Si es la primera vez implementarlo a Android, ejecuta: *ionic capacitor add android / npx cap add android*
2) Si no es la primera vez implementarlo a Android pero han hecho gran cambios, hay que sincronizar el proyecto ejecutando: *ionic capacitor sync android / npx cap sync android*, o *ionic capacitor copy android / npx cap copy android* si no hay cambio de plugins.
3) Ahora en el comando abre el Android Studio con *npx cap open android*. En Android Studio el proyecto ya está implementado, sólo hay que cliquear el botón *Run* para ejecutarlo en un simulador o un dispositivo conectado.

Para un proyecto cordova:

1) Si es la primera vez implementarlo a Android, ejecuta: *ionic cordova prepare android*
2) Si no es la primera vez implementarlo a Android pero han hecho gran cambios, hay que sincronizar el proyecto ejecutando: *ionic cordova run android -l*

Según las instrucciones de (Ionic Team, 2019), también podemos activar la opción Live Reload con la que se detectan los cambios que hemos hecho en el codebase y se sincronizan entre el web y la aplicación en los dispisitivos que conectan al ordenador.

4. Identificación y estimación de coste

Una vez que el proyecto pase la prueba funcional y entra en la siguiente etapa, es necesario determinar si el proyecto es factible mediante el estudio de su impacto económico.

4.1 Recursos humanos

Actualmente el proyecto se encuentra en la fase de construcción (también es la realización del TFM), con la ayuda de la incubadora de UPC se está buscando inversión, los tres miembros del equipo de momento no tenemos ingresos, unido a factores más inciertos, como la incorporación de recién llegados, el resultado de la inversión e incluso el impacto de la pandemia, hace que en esta fase el coste no se puede asumir.

Sin embargo, una vez que el proyecto entre en la fase final como se esperaba, se implementará el presupuesto económico a la vista de tres años que se ha elaborado, incluyendo los puestos previstos y el ingreso salarial correspondiente a cada puesto, que se utiliza aquí como referencia para el presupuesto de recursos humanos.

Areas	Personas	Salario mesual	Bonus mesual	IRPF (impuestos sobre la renta)	Coste empresa mesual	Incremento año 1	Incremento año 2	Incremento año 3
GENERAL	CEO	60,000	10000	21000	91000	0.16	0.2	0.25
GENERAL	CMO	12000		0	12000			
MARKETING	Product Manager	18000	1500	5850	25350			
TI	Product Manager	18000	1500	5850	25350			
TI	Desarrollador	12000	2000	4200	18200			
MARKETING/TI	Bancario	5400			5400			
OPERACIONES	Ventas	15000	1000	4800	20800		0.5	
Total		140400	16000	41700	198100	0.16	0.7	0.25

CEO	Año 1	Año 2	Año 3
Salario mesual previsto en función de incremento de beneficios	1,213	1.517	1,896

4.2 Recursos no humanos

Dentro de recursos no humanos podemos dividir entre dos categorías diferenciadas, *hardware* y *software*. Tendremos en cuenta el tiempo de vida tanto del software como del hardware y la amortización que se hará

Hardware

A la hora de realizar la fase de construcción del proyecto serán necesarios una serie de *hardwares* teniendo en cuenta que el tiempo de amortización son 3 años podemos calcular los costes siguientes

Producto	Precio unidad (€)	Unidades	Precio total (€)	Vida útil (años)	Amortización (€)
Ordenador de Lenovo Core i5 4210U/4GB/450G	500	1	500	3	100
Ratón Miniso	7	1	7	3	1.4
Dispositivo Android OnePlus 7+	400	1	400	3	80
Total			907		181.4

Software

A la hora de realizar la fase de construcción del proyecto serán necesarios una serie de *softwares* pero ninguno de ellos genera costes de momento, también teniendo en cuenta que el tiempo de amortización son 5 años podemos calcular los costes siguientes

Producto	Precio unidad (€)	Unidades	Precio total (€)	Vida útil (años)	Amortización
Windows 10 OS	0	1	0	5	0
Visual Studio Editor	0	1	0	5	0
Android Studio	0	1	0	5	0
Bitbucket	0	1	0	5	0
Firebase	0	1	0	5	0
Google Cloud Platform	0	1	0	5	0
Total			0		0

Total

Finalmente obtenemos la siguiente tabla calculando los gastos del *hardware* y del *software*:

Tipos	Precio (€)
hardware	907
software	0
Total	907

5. Sostenibilidad y compromiso social

5.1 Económica

Si bien existen en el mercado muchas aplicaciones deportivas y más específicamente, orientadas al entrenamiento de *running*, el proyecto busca desarrollar un *software* propio de acuerdo con el concepto de *eco-running*, que se coincide a la filosofía de la empresa, de manera de brindar la multi-funcionalidad y participación alta del usuario que quiere diferenciarse de otros proyectos. Por eso es necesario seguir desarrollando nuevos *softwares* para este tipo de proyectos basándose en los existentes en el mercado. Pero esto no significa que algunas funciones o módulos de las aplicaciones existentes no se puedan reutilizar, sino que ha de agregrse nuevas características. El riesgo potencial puede ser que los competidores ofrezcan el mismo tipo de productos a precios más bajos o con una mayor lealtad de los usuarios.

5.2 Social

Desde la perspectiva de mi propio desarrollo, el proyecto me ha brindado no solo la oportunidad de practicar los conocimientos adquiridos en el proceso de aprendizaje, sino también la de adquirir los nuevos conocimientos, como el uso de nuevos *frameworks* y lenguajes de programación basados en el desarrollo de Android. El producto aparecerá en el mercado como un *software* multifuncional de entrenamiento de *running*, que también puede definir sus funciones a nivel social y, al mismo tiempo, fortalecer el sentido de identidad entre grupos específicos. A nivel de marketing, el *software* ha ganado una base de clientes clara, y el objetivo es probar los requerimientos funcionales y proporcionar *feedbacks* en la siguiente fase del proyecto, por ejemplo, la precisión del geolocalización del mapa, la frecuencia de actualización del estado del usuario, etc.

5.3 Medioambiental

Así como el concepto *ecorunner* que se ha enfatizado en la motivación, la visión del proyecto en términos de *marketing* siempre ha estado estrechamente vinculado al desarrollo sostenible del medio ambiente. Se apuesta por crear un ambiente deportivo nacional mejorando la salud de las personas y el entorno ecológico. La mejor manifiesto es la sección *Marketplace* de la aplicación donde el proyecto establece conexiones con muchas empresas que desarrollan productos sostenibles, por ejemplo, *carefood* para alimentación ecológica,

becrit pra batidos de proteína saludables y etc., entre ellos, el *running public* es conocido por diseñar y fabricar ropa deportiva ecológica.

6. Entorno de Trabajo

- Node.js (v12.16.1)

  Node.js es un entorno de ejecución que permite escribir JavaScript a lado del servidor. Además de utilizarse para servicios web, el nodo se utiliza a menudo para crear herramientas de desarrollo, como *Ionic CLI*.

- npm (v6.13.4)

  Npm es el administrador de paquetes para node.js. Permite a los desarrolladores instalar, compartir y empaquetar módulos de node. Ionic se puede instalar con npm, junto con varias de sus dependencias.

- Ionic CLI (v6.2.0)

  Asegurando de haber instalado bien el Node.js, se instala el Ionic *CLI (**C**ommand-**L**ine **I**nterface)* con npm en el terminal con el siguiente comando:

  $ npm install -g @ionic/cli

  La opción -g significa instalar globalmente.

  Después, en la interfaz de comando de Ionic, ya se puede crear un proyecto usando: *$ ionic start myApp*. Finalmente, para correr el proyecto, entra en el directorio del proyecto y ejecuta: *$ ionic serve*. Pasando unos momentos, la interfaz de la aplicación se abrirá en tu navegador con la dirección localhost: 8100.

- Paquete @ionic/angular

  El paquete @ionic/angular combina las herramientas principales de Ionic y las APIs que se adaptan a los desarrolladores de Angular. Es significativo para realizar la interfaz de ionic con el idioma angular. Para instalarlo también ha de usar el npm: *$ npm install -g @ionic/angular*

- *Visual Studio Code*

  Un editor de código es claramente necesario. *Visual Studio Code* es un editor de texto gratuito que incluye baterías y creado por *Microsoft*.

- Android Studio

  El *test* de prueba a las funcionalidades se realiza en un dispositivo Android (Oneplus 7+), por lo tanto para que la base de código se compile al entorno de Android y se ejecute, se necesita el *IDE* para la plataforma Android, Android Studio.

- Navegador Chrome

La base de código se escribe con idioma *Typescript* y al iniciar el servidor de Ionic se ejecuta en la página *web*. Técnicamente se puede usar cualquier navegador, en el caso del proyecto, se usa Chrome de Google.

7. Diseño

7.1 Arquitectura de aplicación híbrida

Como se muestra en la *Ilustración 11*, una aplicación híbrida consiste en las siguientes capas:

- *Frontend Framework*
- *UI control*
- *Native Runtime*
- *Native Platform*
- *Cloud Functions or Database*

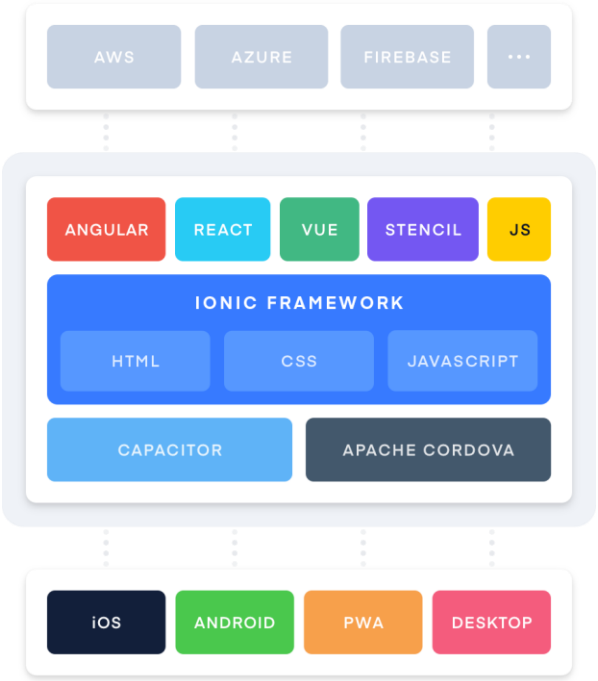


Ilustración 11. Arquitectura de una aplicación híbrida

7.1.2 Interacciones entre capas

Usuario abre la aplicación	**Dispositivo**	**Capacitor**		**Web aplicación**	
		Contenedor de Capacitor aplicación	La vista de web con Javascript API	Angular	Ionic
	el dispositivo carga el contenedor de la Capacitor aplicación	carga nueva vista de web	carga el fichero index.html	El bootstraps de Angular determina la vista por defecto	Los componentes de Ionic están reproducidos por la IU
	Dispositivo	Capacitor	Javascript	Angular	Ionic detecta

Ejemplo: usuario pide que active la cámara

	activa la aplicación de cámara (puede pedir permiso)	pide al dispositivo que active la aplicación de cámara	API pide al Capacitor por la función de cámara	llama Capacitor Cámara Javascript API	que la solicitud. Llama Angular	
	Cierra la aplicación de cámara y retorna fotos →	Capacitor recibe y transmite fotos →	Javascript API transmite fotos →	Angular recibe fotos y reproduce la vista →	Los componentes de Ionic están reproducidos por la IU	

Tabla 2. Interacción entre capas de aplicación híbrida

Como se muestra en la *Tabla 2*, el conjunto central de una aplicación híbrida se puede dividir en tres partes.

El dispositivo instalado con plataforma nativa es la parte que se comunica directamente con el usuario, cuando el usuario abre la aplicación, el *framework* Capacitor empieza a cargar la vista de web y transmitir la orden hacía el *framework* de Angular y Ionic para que ellos preparen las interfeces de usuario. El método de realizar la comunicación entre plataforma y Capacitor se varía, por ejemplo, un nativo ios se compila usando Xcode mientras un nativo Android se compila en Android Studio.

Respecto a la capa de base de datos, se interactúa con la web aplicación y participa en la parte de Angular, almacenando datos que se reproduce cuando sea necesario.

7.1.3 Flujo de funcionamiento

En la fase de construcción actual, el flujo de funcionamiento diseñado según interfaces es lo que se muestra en la *Ilustración 12*.

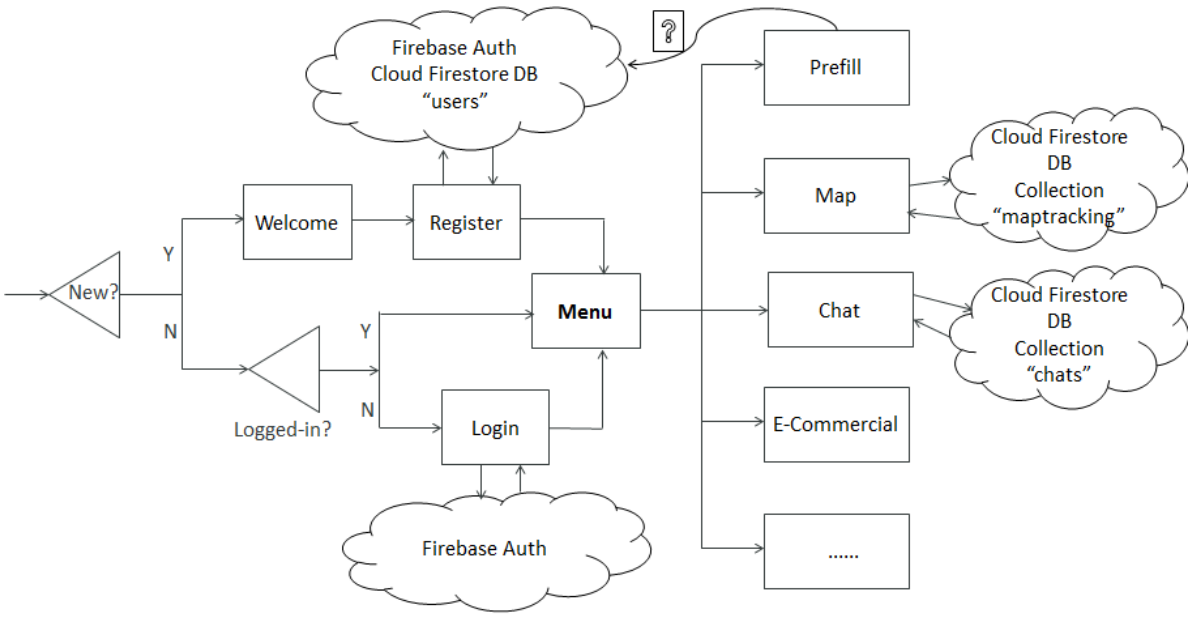


Ilustración 12. Flujo de funcionamientos

7.1.4 Modelo de casos de uso

El modelo de casos de uso es una representación de las funciones principales del sistema. Cada caso de uso refleja una secuencia de interacciones que realiza entre un actor del sistema y la aplicación. En el contexto del proyecto, existen dos actores:

- Pacer Smart Beat User (Pacer user): Es el usuario de la aplicación.
- Receptor de datos: Es el servicio encargado de recibir los datos que intruduce el usuario o insertarlos en la base de datos, como *AngularFire* o *Firebase Cloud Function*.

A continuación, en la *Ilustración 13*,se muestran los casos de uso de la aplicación para el actor Pacer user. Posteriormente, se entra en detalle para cada uno de ellos.

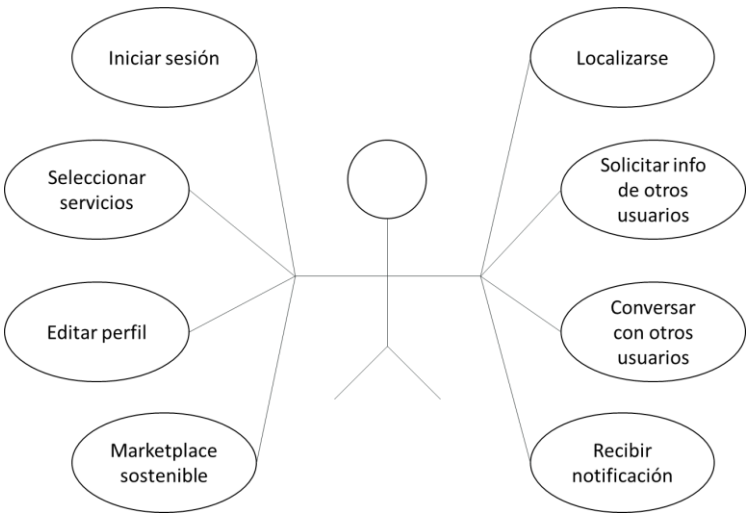


Ilustración 13. Casos de uso de Pacer user

Iniciar Sesión	
Descripción	Para acceder los servicios que ofrece la aplicación, el usuario tiene que disponer de una cuenta propia de Pacer Smart Beat. En el caso contrario, se quedará en la interfaz de información y bienvenida.
Actores	Pacer user
Precondición	
Postcondición	La aplicación está lista para ser utilizada.
Flujo normal de eventos	
1. El Pacer user entra en una interfaz de bienvenida al abrir la aplicación por la primera vez, donde puede iniciar la sesión o hacer el registro. 2. Dispone de dos maneras de crear la cuenta. Una es conectarse con una tercera aplicación como Facebook, Google o Apple para el sistema de ios. La otra forma es crear la propia cuenta de Pacer Smart Beat rellenando los datos en el formulario. 3. La aplicación comprueba que todos los datos introducidos están válidos y envía la información personal al sistema de autenticación de Firebase para que cree la cuenta. Mientras tanto, el Pacer user accederá a la interfaz de espera de validación de correo electrónico. 4. El Pacer user valida su correo electrónico en un enlace que envía el sistema. 5. La aplicación muestra el menú principal con el nombre del Pacer user. 6. El Pacer user puede ver la pantalla principal de la aplicación sin ningún error.	
Excepciones	
1. En el caso de que el Pacer user olvida su contraseña de la cuenta, puede entrar en la interfaz de recuperación de la contraseña. El sistema le enviará un *mail* a su correo vinculado. El Pacer user puede establecer una nueva contraseña vía el enlace del *mail*. 2. En el caso de que el Pacer user introduce su correo eletrónico o su contraseña de la cuanta incorrectamente, la aplicación le salta un aviso para que los corrija. 3. En el caso de que el Pacer user introduce los datos inválidos durante el proceso de registro, la aplicación no le deja crear la cuenta avisándole dónde está el error, por ejemplo, la dirección del correo electrónico tiene que ser de forma válida y la contraseña ha de contener una letra mayúscula, una letra minúscula y un número mientras mantenga la longitud entre 8 y 16 caracteres.	

Seleccionar los servicios	
Descripción	En el menú principal de la aplicación, el usuario puede seleccionar los servicios que le interesan. Dispone de un diseño similar a *smartwatch*, el usuario puede rodar el menú circular para ver más opciones.
Actores	Pacer user
Precondición	El usuario tiene que identificarse
Postcondición	-
Flujo normal de eventos	
1. El sistema coge el nombre del Pacer user que ha introducido al hacer el registro y lo pone en la cabecera del menú, indicando que el usuario ya está identificado y puede usar todos los servicios. 2. El Pacer user puede girar el menú circular bidireccionalmente para ver más servicios porque en la pantalla solo se muestra la mitad del círculo. Al estar seleccionado un servicio, la esfera correspondiente se cambia el color por un instante y el usuario entra a otra interfaz.	
Excepciones	
-	

Editar perfil	
Descripción	En esta sección la aplicación proporciona un formulario orientado al campo de entrenamiento *running* y de la salud, con el cual el usuario puede completar su perfil como un *ecorunner*. Con todos los datos rellenados, nuestros expertos pueden analizar el estado de salud del usuario y recomendarle una planificación personalizada.
Actores	Pacer user
Precondición	El usuario tiene que identificarse
Postcondición	-
Flujo normal de eventos	
1. El Pacer user rellena el formulario en función de su estado actual y real. 2. La aplicación coge los datos personalizados y almacenarlos en la base de datos 3. Próximamente, unido con la funcionalidad de consulta, la aplicación ofrecerá servicios personalizados sobre la planificación de entrenamiento para el Pacer user, colaborada con los expertos en este campo.	

Excepciones
-

Marketplace sostenible	
Descripción	En la sección de *Pacer shop*, el usuario puede encontrar varios tipos de productos sostenibles y visitar las páginas web de las empresas que venten los productos. Como una empresa inspirada por el concepto de sosteniblidad, tenemos colaboración con otras empresas o proyectos que comparten las mismas ideas. Si el usuario realiza las compras desde la aplicación Pacer, puede conseguir los descuentos correpondientes.
Actores	Pacer user
Precondición	El usuario tiene que identificarse
Postcondición	-
Flujo normal de eventos	
1. El Pacer user entra la tienda online y mira las categorías. Al hacer clic en una categoría encuentra los productos de este tipo. 2. Si al Pacer user le interesa un producto y quiere ver más información, puede hacer clic en la marca del producto y la aplicación le redireciona a la página *web* de la empresa donde el Pacer user puede terminar la compra.	
Excepciones	
-	

Localizarse	
Descripción	En la sección de buscador de *runners*, el usuario puede localizarse en el mapa que se integra con el Google Maps API. En comparación con el mapa de Google, es un mapa más "limpio", sin nombres de tienda o de edificio ni el mode de satélite o panorámico, ya que la aplicación quiere crear un ambiente más concentrado a terminar la carrera. Dispone de un botón de cambiar la visibilidad para el propio usuario.
Actores	Pacer user
Precondición	El usuario tiene que identificarse
Postcondición	-

Flujo normal de eventos
1. Una vez que el Pacer user entra en el mapa de buscador de *runner*, activa el API de geolocalización. 2. El API de geolocalización empieza a localizar el usuario y muestra su ubicación con un marcador. 3. La aplicación dispone de tres tipos de marcador para distinguir la identidad de usuarios y sus estados de visibilidad. 4. Automáticamente al entrar en el mapa está visible el Pacer user, si en el momento no quiere estar acompañado, al hacer clic en el icono su estado de visibilidad cambia a falso. Al hacer clic otra vez, vuelve a estar visible. Durante el proceso de cambios, el marcador del Pacer user cambia en función del estado. 5. Cuando el usuario cambia su estado de visibilidad con el icono, el sistema busca el campo de visibilidad en la base de datos de Cloud Firestore y examina su valor.
Excepciones
1. En el paso 5 del Flujo Normal, el cambio del valor en el campo de visibilidad puede sufrir retrasos debido al estado de internet o a la conexión con Firebase. Eso puede resultar que el marcador del Pacer user no ha cambiado después de que el Pacer user ha hecho clic en el icono.

Solicitar información de otros usuarios en el mapa	
Descripción	En la sección de buscador de *runner*, además de localizarse, el usuario también puede actualizar las ubicaciones y solicitar información de otros usuarios, de manera que puede establecer conexiones con la gente dentro de la comunidad de Pacer.
Actores	Pacer user
Precondición	El usuario tiene que identificarse
Postcondición	-
Flujo normal de eventos	
1. Al entrar en el mapa, el Pacer user puede visualizar las ubicaciones de otros Pacer users en el mapa, actualizando por el API de geolocalización de Capacitor. 2. La aplicación actualiza los datos de geolocalización (latitud y longuitud) en la base de datos de Cloud Firestore con un rendimiento alto, para todos los Pacer users.	

3. Un Pacer user puede hacer clic en el marcador de otro Pacer user para solicitar más información de él. La información contiene el nombre, la edad, el avatar y la distancia entre dos Pacer users. 4. Cuando el Pacer user solicita la información de otros usuarios, el sistema va a revisar su colección de localiación en BD donde contiene su id, la latitud, la longuitud y su estado de visibilidad. 5. El sistema coge la id en la colección de localización, con la que puede identificar ese usuario en otra colección de información personal. 6. El sistema encuentra las informaciones personales en dicha colección y las proporciona a la vista web 7. El API de Google Maps dispone de un *infowindow* para mostrar las informaciones adicionales encima del marcador. All íel Pacer user puede ver la información solicitada.
Excepciones
1. En el paso 3 del Flujo Normal, la función de calcular la distancia entre dos Pacer users puede equivocarse de vez en cuando debido al retraso de actuzalizar las ubicaciones.

Conversar con otros usuarios	
Descripción	Tanto en la sección de comunidad como en la de buscador de *runners*, el usuario puede encontrar a otros usuarios y empezar una conversación.
Actores	Pacer user
Precondición	El usuario tiene que identificarse
Postcondición	-
Flujo normal de eventos	
1. Al solicitar más información de otros usuarios en el mapa, el Pacer user puede encontrar un botón de *chat* en el *infowindow*. Al hacer clic en el botón, entra en una habitación privada donde se puede conversar. 2. El sistema revisa el Session Storage a la vista de web para identificar el destinatario de la conversación y Local Storage para identificar el remitente. 3. En la habitación privada, el Pacer user puede enviar mensajes. Los mensajes se muestran con la forma de una tarjeta en la que contiene el nombre, el contenido y el tiempo de envio. 4. Una vez establecido la conexión, el sistema almacena los mensajes en la base de datos, usando un formato formado de *ids* de ambas partes y el *timestamp*.	

5. Al entrar la sección comunidad, el Pacer user puede ver todas conversaciones que ha tenido con cada usuario y borrar mensajes en cada habitación. 6. El sistema coge todas las informaciones de mensajes desde la base de datos y las muestra en la lista de contacto. Cuando un mensaje está borrado por un usuario, el mensaje sigue existiendo en la base de datos y en la conversación de otro usuario.
Excepciones
-

Recibir notificación	
Descripción	La aplicación va a enviar notificacines a sus usuarios sobre el tema de sostenibilidad, consejos de entrenamiento, novedades de la empresa y etc. con ciertas frecuencias.
Actores	Pacer user
Precondición	El usuario tiene que identificarse
Postcondición	El dispositivo del usuario tiene que estar encendido
Flujo normal de eventos	
1. Una vez que el Pacer user instalado la aplicación, por defecto puede recibir notificación ya que la aplicación consigue el *token* (identificador) del dispositivo que usa. 2. El personal del departamento *marketing* puede editar los contenidos de la notificación en la consola de *Cloud Messaging Firebase*, configurando las frecuencias de envío, el grupo de usuario oriendado y el tipo de la notificación. 3. Al publicar la notificación en la plataforma de *Cloud Messaging*, el sistema revisa la configuración y empieza la planificación de envío. 4. Cuando la notificación está presionada, el sistema activa la acción de tratamiento "pushNotificationReceived" para abrir la aplicación 5. El Pacer user entra la aplicación.	
Excepciones	
En el paso 1 del Flujo Normal, para algunas versiones del sistema operativo Android, por defecto está activado la optimización del uso de batería, de manera que rechaza todos los tipos de notificación. Para la solución se puede consultar la sección "8.8 Capacitor Cloud Messaging Push Notification".	

7.2 Estructura del proyecto

Como es una aplicación híbrida se realiza a través de una base de código como se observa en la Ilustración 14 se pude visualizar la estructura del proyecto.

La aplicación de Ionic&Capacitor se implementa bajo el entorno de Node.js, por lo que el archivo *node_modules* proporciona muchos modelos y servicios que se requieren para las funcionalidades después de instalar los paquetes correspondientes en la consola de Ionic. El fichero *capacitor.config.json* contiene toda la información esencial del proyecto incluido el nombre y la identificación de la aplicación. El fichero *google-services.json* contiene todos los datos de configuración para la conexión de Android. El fichero de *package.json* contiene una lista de paquetes instalados en el proyecto con sus informaciones resumidas.

El nivel de interfaz de usuario, es el sistema con el que el usuario interactúan con el programa y les permite reaccionar ante la información que le ofrece. Excepto el archivo maptracking, que es una funcionalidad desarrollando separada del resto. Por ejemplo, en el archivo *main*, se implementa el diseño del menú principal, y además, la funcionalidad de buscador de runners y perfil de salud. En la sección "7.1 Árbol de interfaces" se puede consultar el esquema de las interfaces y sus relaciones entre sí

La base de datos, se compone de los servidores de base de datos donde se almacena y se manipula la información. En el fichero *firestore.indexes.json* y *firestore.rules* se contienen las configuraciones de Firebase y sus normas. Y además, el servidor de Firebase y de AngularFire se arrancan al iniciar la aplicación para asegurar la identificación de los usuarios y la presentación de los datos obtenidos en la base de datos. En el siguiente apartado se explicará concretamente la estructura de la base de datos.

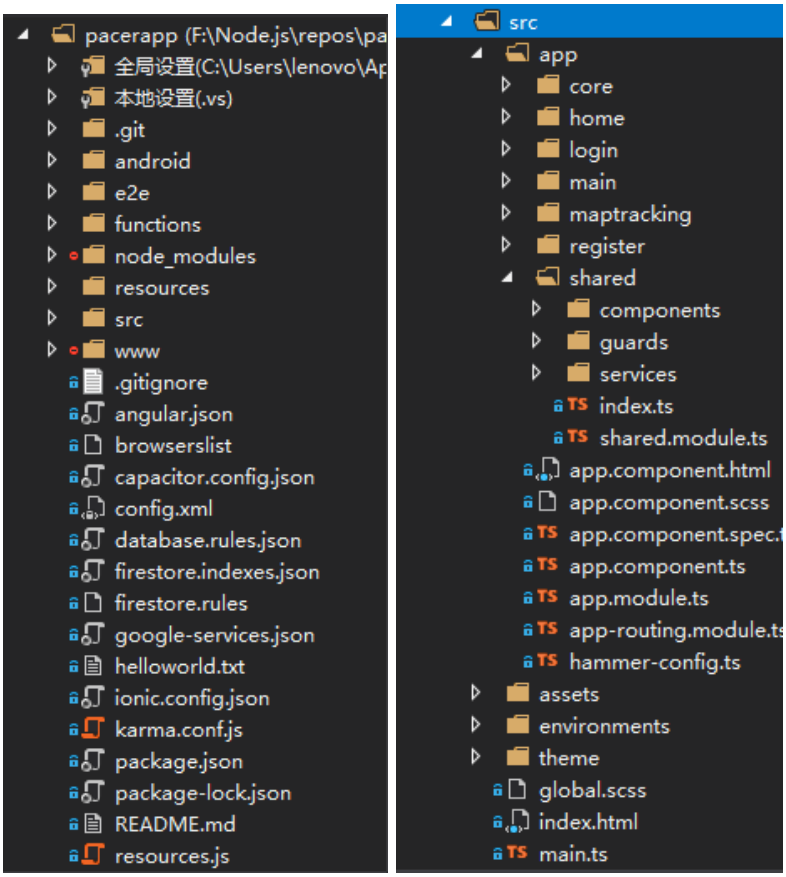


Ilustración 14. Estructura del codebase

7.3 Diseño de la interfaz del menú principal

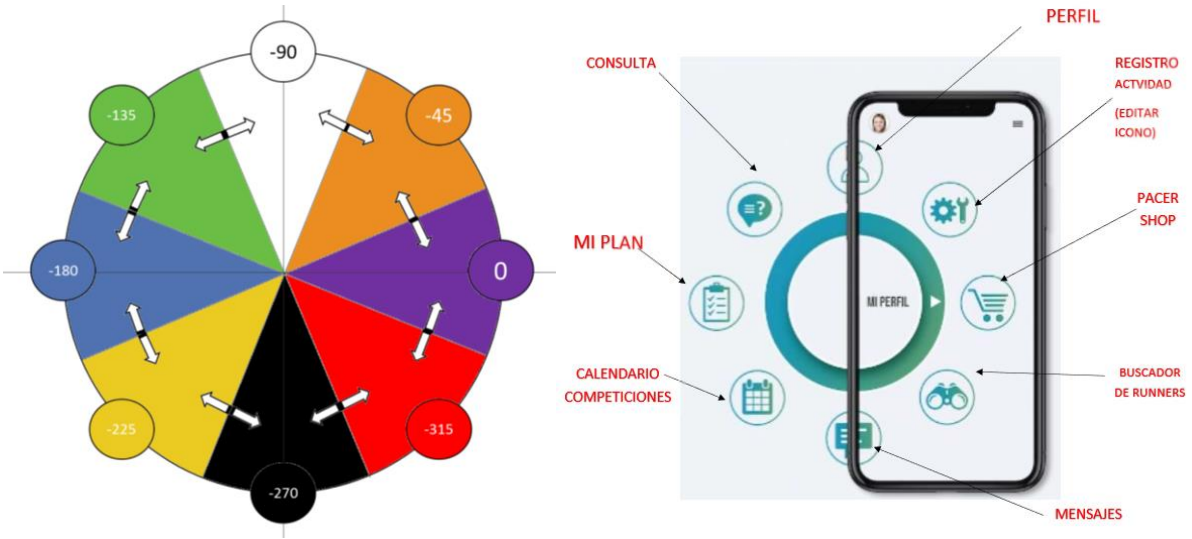


Ilustración 15. Diseño del menú principal

Como se muestra en la *Ilustración 15*, que son dos esbozos estructurales, el menú principal del usuario está diseñado con una interfaz parecida a Smartwatch que consiste en un semi-círculo y ocho esferas, las cuales representan diferentes funcionalidades y distintos

contenidos. La mitad de ellas se observan de un solo golpe de vista, sin tener que girar el menú.

Cuando el usuario toca el separador o el espacio blanco de la pantalla y se mueve hacia la dirección correspondiente al gesto del dedo, tanto hacía arriba como hacía abajo, el círculo rueda cambiando las esferas y sus títulos. Cada esfera es un menú en el cual, cuando el usuario cliquea en ella accede a un submenú.

Además, en esta página, hay otro menú *slide* dentro del icono comúnmente llamado "bocadillo" donde el usuario puede cerrar la sesión o hacer más configuraciones.

Para que las esferas se ruedan bidireccionalmente y se paran en ciertas posiciones cuando soltamos los dedos, hemos definido las acciones de gestos, las indicaciones de dirección, ocho posiciones fijas y sus alcances de afectar.

Gracias las inspiraciones que me han dado (Ionic Team, 2019) y (Soueidan, 2013), hemos realizado este diseño con el propio API "gesture" de Ionic con los siguientes pasos:

a. Formar un círculo hueco con el elemento *div* manipulando su fichero de css. Mover el círculo hacia el lateral izquierdo para que se muestra solo la mitad en la pantalla.
b. Crear ocho esferas con el elemento *div* también y colocarlos en las ocho posiciones fijas preconfiguradas, ajustando con el atributo "style" para que rodeen el círculo equivalentemente. Posteriormente las esferas pueden aparcarse en cualquier de las ocho posiciones fijas.
c. Importar los paquetes de gesto de Ionic. Importar "Renderer2" de Angular para manipular los estilos de elemento.
d. Definir las acciones de gestos que consiste en tres estados: empiezo, moviendo, acabado.

- Al empezar, definir la transformación a "none" de todos los elementos para que se vuelvan a posición original cada vez que entramos al menú principal.
- Al estar moviendo, distinguimos el gesto de usuario con una condición: si la posición en eje Y donde soltamos el dedo es mayor que la posición donde empezamos a presionar el dedo, indica que el usuario está girando hacia abajo, mejor dicho, según el sentido del reloj. En el caso contrario, indica que el usuario está

girando según el sentido contra del reloj. Como saben todos, un círculo está formado con 360 grados. Aquí en la interfaz del móvil, el punto inicial se ubica en la mitad del borde derecho. Si giran según el sentido del reloj, se aumenta el grado desde 0 hasta 360 volviendo al mismo punto y vice versa. Basándose en esa teoría, después de detectarse la dirección hacía donde quiere girar el usuario, se aumenta o se disminuye el grado según cada caso a cada uno de los ocho elementos.

- Al acabarse, hemos definido un alcance de radiación para cada una de las posiciones fijas como se muestra en la *Ilustración 15*, cuando una esfera se intenta parar, se acaba colocando en la posición más cercana(que la esfera está dentro de cuya radiación).

e. Adicionalmente, al tocar una de las esferas, se cambia a un color degradado con estilo de la aplicación. Y cuando una esfera se destaca, el título al lado se cambia a uno correspondiente.

Finalmente, la inspiración de la mayoría de los iconos que se presentan en el proyecto viene de (Ionicons:The premium icon pack for Ionic Framework, 2020) y (Noun Project:Free Icons & Stock Photos For Everything, 2020)

7.4 Sistema de autoidentificación

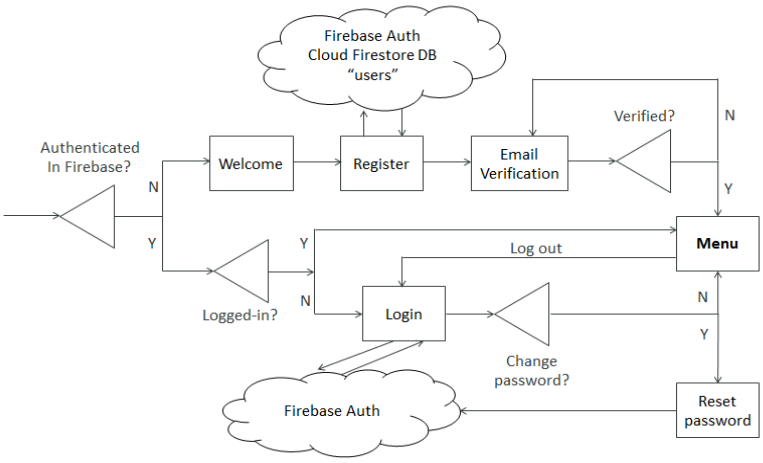


Ilustración 16. Estructura del sistema de autenticación

Visualmente en la *Ilustración 16*, esta aplicación tiene el sistema de autenticación funcionando de la siguiente manera:

- Si un usuario ya tiene la cuenta y se mantiene el estado *logged-in* la última vez que la usa, al entrar la aplicación entrará directamente en el menú principal.

- Si un usuario ya tiene la cuenta pero hace el *log-out* la última vez que la usa, al entrar la aplicación entrará en la página de login. Si en este caso el usuario quiere crear una cuenta nueva, puede redireccionarse a la página de registro con un enlace.
- Si es un nuevo usuario para la aplicación, puede entrar la página de registro desde la página de bienvenida eligiendo el botón "únete con nosotros".

8. Implementación

La aplicación de Pacer Smart Beat ha sido implementada utilizando el *framework* Ionic, el idioma de programación Angular y el editor Visual Studio. Todos los datos han sido almacenados en *Cloud Firestore*, una base de datos no relacional. Algunas funcionalidades han utilizado *APIs* colaborados con otras plataformas, como *Google Cloud Platform*. A continuación, se explicarán los puntos claves durante la implementación de la aplicación.

8.1 Creación del proyecto con Ionic CLI

Una vez instalado el *CLI* de ionic y el paquete para Angular @ionic/angular, ya se puede crear un proyecto con herramientas preconfiguradas de ionic y abrirlo en Visual Studio.

En la interfaz de *shell*, al entrar en el directorio del proyecto, hay varias opciones de generar elementos para el proyecto. Introduce el comando *$ionic g*, saldrá una lista de opciones como se muestra en la *Ilustración 17*.

Entre los que hemos utilizado para el proyecto, "page" y "component" se usan para construir páginas individuales de aplicaciones con los contenidos principales, las cuales consisten en tres partes:
fichero de html, donde construyen los contenidos que se ven en la pantalla,
fichero de ts, donde implementan las funcionalidades correspondientes a los contenidos,
fichero de scss, donde diseñan los estilos de los contenidos visualmente.
Algunas páginas pueden contener su modelo para identificarse cuando en otros sitios piden su referencia. Para las que no contengan modelo al crearse, por ejemplo, los componentes, podemos crear modelos personalizados por nuestra propia cuenta según (Steffen, 18).

"service" se usa para implementar separadamente los servicios de base de datos, de conexiones de APIs, de funcionalidades globales para la aplicación, etc.

"module" ofrece un modelo único para cada página o componente. Se usa para identificar una página o un componente globalmente y realizar transmisiones o intercambios de información entre ellos.

"guard" juega un papel de portero. Se usa para establecer restricciones de entrar una página donde puede contener informaciones limitadas para algunos usuarios identificados. Se aplica normalmente en el sistema de autenticación, por ejemplo, un usuario no puede entrar al menú principal salvo que esté identificado.

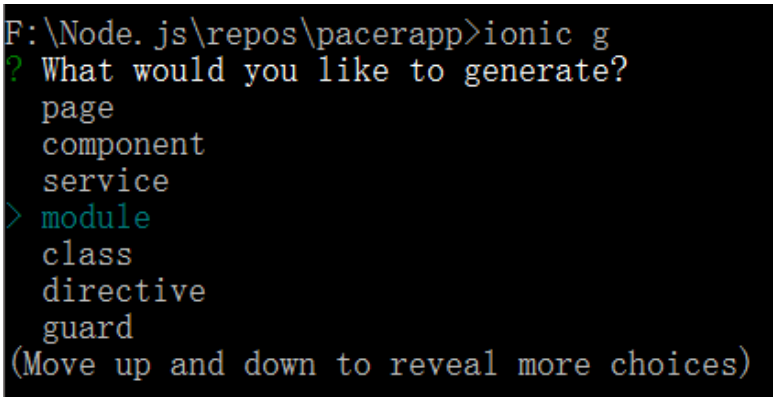


Ilustración 17. Interfaz de Ionic CLI para generar fichero

8.2 Establecimiento de la conexión con Firebase

El proyecto ha utilizado Firebase como el backend de la aplicación donde se realizan las *cloud* funcionalidades, se identifican los usuarios, se guardan todos los datos personales y las informaciones de las localizaciones, etc. Para establecer la conexión entre mi proyecto de ionic y firebase se requerieren unas configuraciones distintas según las plataformas donde se implementa la aplicación.

Antes de todo, en la dirección https://console.firebase.google.com aparece la interfaz de crear un proyecto. Como hemos creado uno antes, me aparece el enlace del proyecto al lado con el que puedo entrar en el *dashboard*.

Ilustración 18. Interfaz de crear un proyecto

Lo primero que es necesario hacer después de crearse un proyecto, es agregar firebase a una plataforma donde quieren montar el proyecto. En mi caso, lo añadía la plataforma *Web* (la base de código, donde se implementan las funcionalidades y se visualizan vía página web) y Android (plataforma nativa, donde la base de código se compila a un proyecto nativo adaptado a Android y se ejecuta en un dispositivo físico).

8.2.1 Web App

Ilustración 19. Registro de aplicación web

a. Primero, registrar la aplicación *web*. Después, hay que agregar el *SDK* de Firebase a nuestro proyecto para utilizar el servicio de ellos. Según indicación, ha de copiar los códigos de configuración al fichero src/environments/environments.ts del proyecto.

```
<script>
  // Your web app's Firebase configuration
  // For Firebase JS SDK v7.20.0 and later, measurementId is optional
  var firebaseConfig = {
    apiKey: "AIzaSyB9mJRPpZu33SpSiZ5d-Ec8KXHCyzJA5kU",
    authDomain: "pacer-ionic-db.firebaseapp.com",
    databaseURL: "https://pacer-ionic-db.firebaseio.com",
    projectId: "pacer-ionic-db",
    storageBucket: "pacer-ionic-db.appspot.com",
    messagingSenderId: "39891208292",
    appId: "1:39891208292:web:0f0e80374829bab1fbf771",
    measurementId: "G-RH4GFY72JJ"
  };
  // Initialize Firebase
  firebase.initializeApp(firebaseConfig);
  firebase.analytics();
</script>
```

Ilustración 20. Interfaz del servicio de Firebase

b. En el fichero src/app/app.module.ts, ha de importar la interfaz de Firebase que se han copiado al fichero de environments.ts y el servicio se iniciará tanto con el paquete de firebase como con los modelos de Angular Fire.

```
//firebase services + environment module
import { AngularFireModule } from '@angular/fire';
import { AngularFireAuthModule } from '@angular/fire/auth';
import { AngularFirestoreModule } from '@angular/fire/firestore';
import { AngularFireDatabaseModule } from '@angular/fire/database';
import { AngularFireMessagingModule } from '@angular/fire/messaging';
import { environment } from '../environments/environment';
import * as firebase from 'firebase';
firebase.initializeApp(environment.firebase);
```

Ilustración 21. Arranque de servicios de Firebase

c. Ya se puede utilizar los servicios específicos de Firebase o Angular Fire. Para usarlos solamente hay que importar los paquetes correspondientes en cada fichero de ts.

8.2.2 Android App

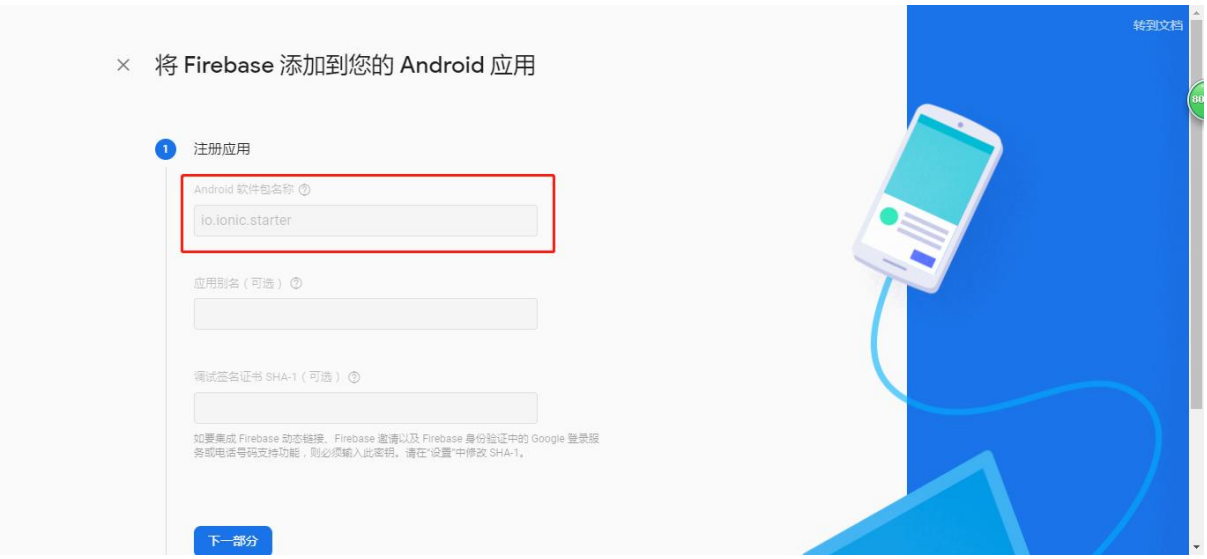


Ilustración 22. Primer paso de la implementación a Android

a. Como antes, tras registrar la aplicación Android en Firebase, el paso más importante es descargar y colocar correctamente el fichero *google-services.json*, el cual contiene toda la información necesaria para la conexión con Firebase, al directorio raíz de la base de código y al directorio *app* del archivo android. En la *Ilustración 24* y *Ilustración 24* se muestra el *path* correcto de localizar el fichero dentro del proyecto.

Ilustración 23. Segundo paso de la implementación a Android

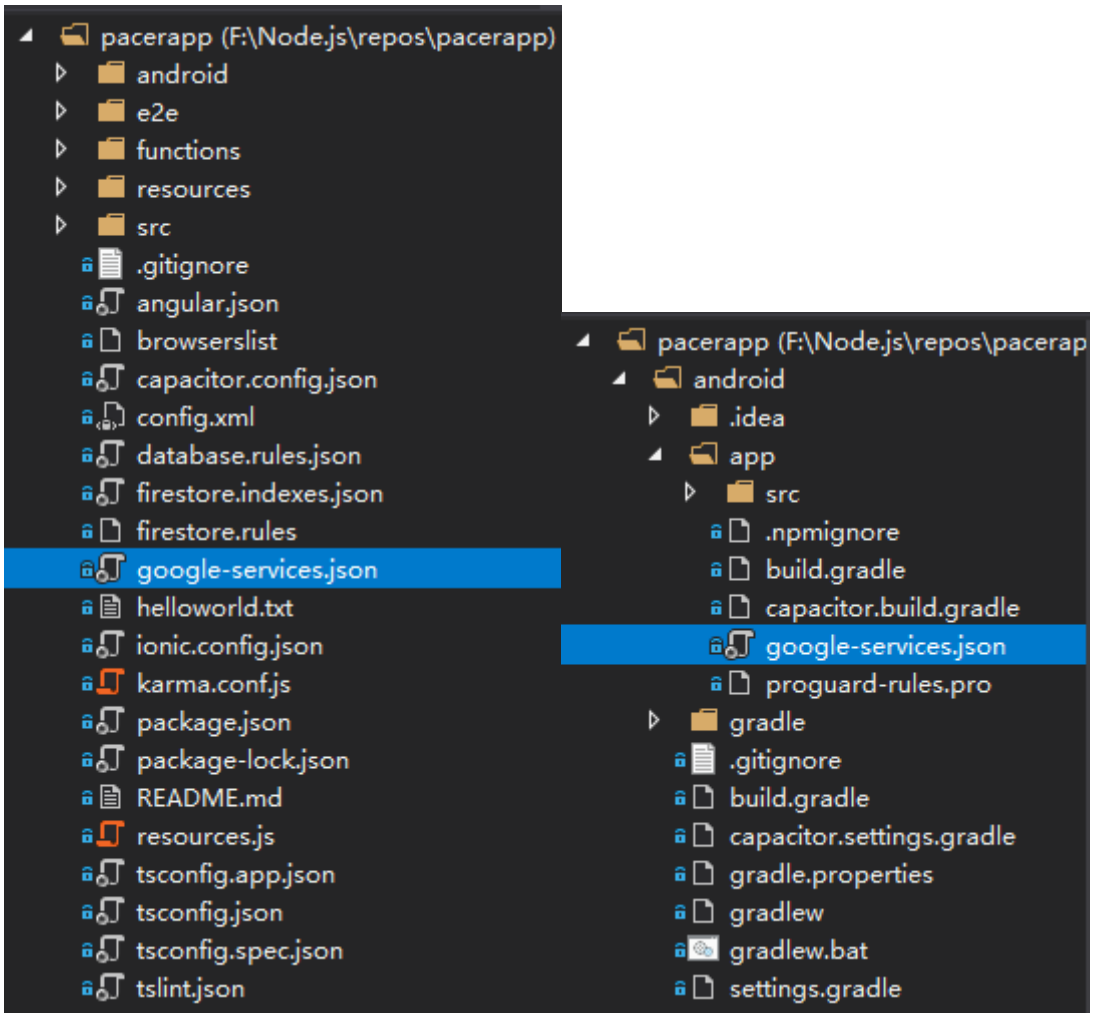


Ilustración 24, Ilustración 25. Colocación correcta de google-services.json

b. Luego, en Android Studio, ha de examinar y añadir los servicios de google manualmente si no están allí. Primero, como se muestra en *Ilustración 26*, en el

fichero *proyecto/build.gradle*, ha de añadir las tres líneas de código para activar el servicio de Google. Segundo, como se muestra en la *Ilustración 27*, en el fichero *proyecto/app/build.gradle*, se ha de agregar un plugin de Google. Finalmente, cuando el proyecto se compila, ya se puede realizar la conexión entre Firebase y el proyecto.

```
// Top-level build file where you can add configuration options common to all sub-projects/modules.

buildscript {

    repositories {
        google()
        jcenter()
    }
    dependencies {
        classpath 'com.android.tools.build:gradle:3.3.2'
        classpath 'com.google.gms:google-services:4.2.0'

        // NOTE: Do not place your application dependencies here; they belong
        // in the individual module build.gradle files
    }
}

allprojects {
    repositories {
        google()
        jcenter()
    }
}
```

Ilustración 26. file proyecto/build.gradle

```
apply plugin: 'com.android.application'
apply plugin: 'com.google.gms.google-services'

android {
    compileSdkVersion 28
    defaultConfig {
        applicationId "io.ionic.starter"
        minSdkVersion 21
        targetSdkVersion 28
        versionCode 1
        versionName "1.0"
        testInstrumentationRunner "android.support.test.runner.AndroidJUnitRunner"
    }
    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro'
        }
    }
}
```

Ilustración 27. file proyecto/app/build.gradle

8.3 Conexión con Google API

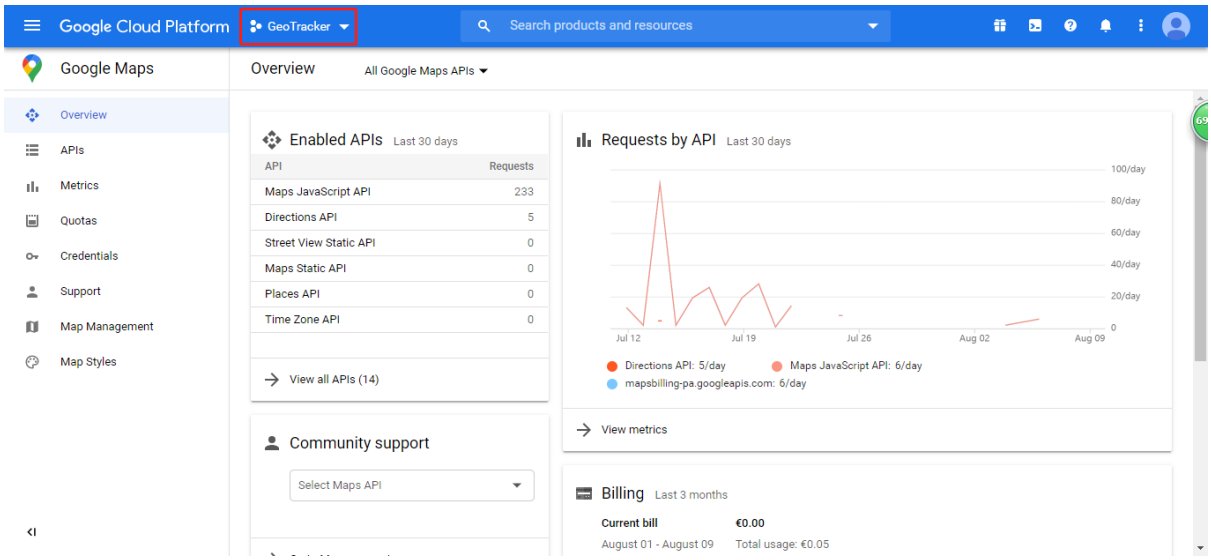


Ilustración 28. Dashboard de Google Cloud Platform

Primero, crear un proyecto en Google Cloud Platform según instrucciones. Hemos creado este proyecto como se ve en la *Ilustración 28*.

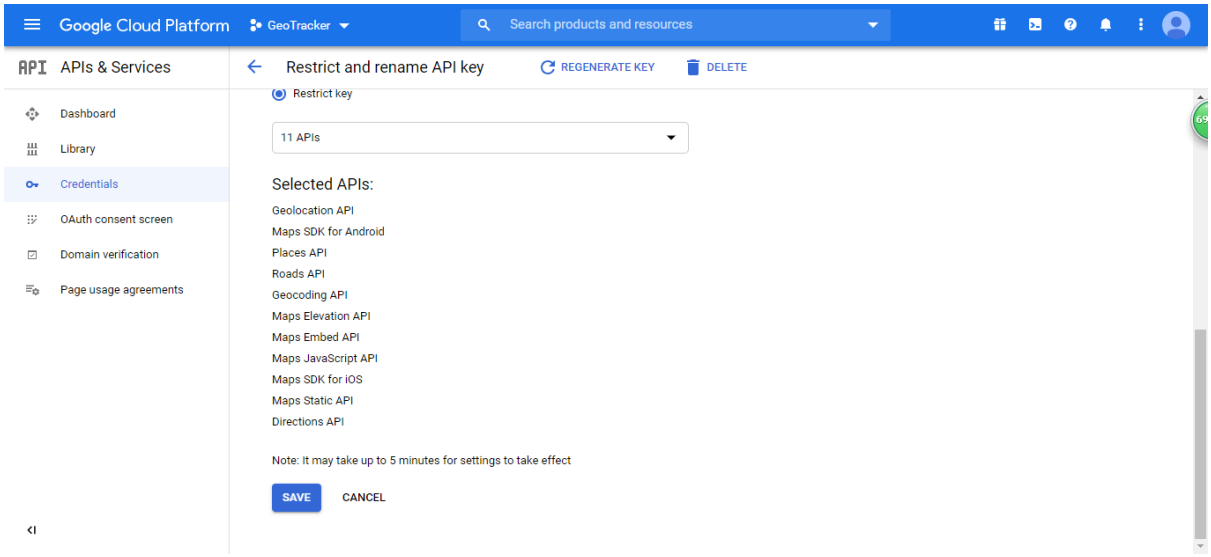


Ilustración 29. Selección de APIs

Segundo paso, elegir los *APIs* que se utilizará en el proyecto en el apartado "Credentials". Al crear una credencial, se genera una clave de *APIs* que luego ha de utilizar en la base de código. Voluntariamente se pueden aplicar restricciones que controlan cuáles sitios *web*, direcciones *IP* o aplicaciones tienen acceso a los APIs que haya elegido (*Ilustración 29*).
Último, en el fichero *index.html* del directorio *root* del proyecto, introduce:
<script async defer src="https://maps.googleapis.com/maps/api/js?key=YOUR_API_KEY"></script>
Ahora el proyecto ya está conectado a los servicios de Google Map.

8.4 Estructura del base de datos

Este proyecto ha utilizado Cloud Firestore Database para guardar todos los tipos de datos necesarios y se han realizado muchas interacciones entre ellos. Los explicaré por separado.

Con la ayuda de (Firebase Channel, 2020), hay que explicar cómo funciona la estructura de Cloud Firestore. Cloud Firestore se conoce como una base de datos tipo *NoSQL*, en diferencia con las bases de datos relacionales y tradicionales como *Realtime Database o Mango DB*, que usan tablas con filas y columnas y cada fila/columna tiene su propia restricción estricta para almacenar los datos, Coud Firetsore guarda los datos en una estructura colección/documento. No tiene las restricciones y normas sobre qué tipo de datos deberá poner en cada sitio de la base de datos, por ejemplo, todos los objetos tienen que pertenecer al mismo tipo o campo, en cambio, cada colección representa un tema o un título sobre el significado de los datos que contiene y cada documento dentro de la colección representa un conjunto de datos que puede contener cualquier tipo. De esta manera los

desarrolladores podemos iterar en el diseño de la base de datos añadiendo o modificando los campos específicos durante la implementación sin romper los restos datos.

Más concretamente, cada documento tiene su único id, que puede ser configurado manualmente por programadores o ser generado automáticamente por firebase. En el segundo caso, el id generado no tiene sentido, no afectará las funciones si solamente queremos conseguir el conjunto de datos dentro del documento. Sin embargo, en algunos casos del proyecto, como las colecciones se interactúan entre sí, necesito también conseguir el único id del documento para hacer unas referencias intermedias. En este caso, el id automático no me servirá de ninguna manera. Basándose en esta teoría, hemos intentado poner el id con sentido para todos los documentos. Esa teoría también sirve para la id de subcolección en el caso de "chats".

- Collection users - Document user id

  Como se muestra en la *Ilustración 30* y *Ilustración 31*, hemos guardado todos los conjuntos de datos de usuario en la colección llamada "users". Cuando un usuario hace el registro en el *frontend*, todos sus datos estarán registrados en un documento de la colección "users", con el id de usuario. Como la autenticación de Firebase está activado, hay cinco campos en la base de datos que son generado automáticamente, es decir, campos como "displayName", "email", "emailVerified", "photoUrl" y "uid" son campos obligatorios para el sistema de autenticación. Además, para coleccionar más datos en los futuros usos, hemos añadido campos como "birthday" y "deviceID".

Ilustración 30. Ejemplo del documento de usuario (1)

Ilustración 31. Ejemplo del documento de usuario (2)

- Collection maptracking - Document user id

Como se muestra en la *Ilustración 32*, hemos guardado todos los conjuntos de informaciones de localizaciones del usuario en una colección llamada “maptracking”. Cada documento también tiene su id único con la id de usuario. Obteniendo esa id, puedo ir a la colección “users” a buscar los datos de un usuario específico con la id correspondiente y mostrarlos en el mapa. En este caso, como se puede observar, es la información del usuario “Leo(gmail)”.

Aparte del campo “uid”, el campo “isTracking” sirve para controlar la visibilidad del usuario mientras los campos “lat” y “lng” sirven para actualizar la localización del usuario. La longitud y la latitud están bajo vigilancia de una función suscrita en el código con la ayuda del servicio de *google map API*. Por lo tanto, cuando un usuario está corriendo, los datos dentro de estos campos van cambiando bajo ciertas frecuencias.

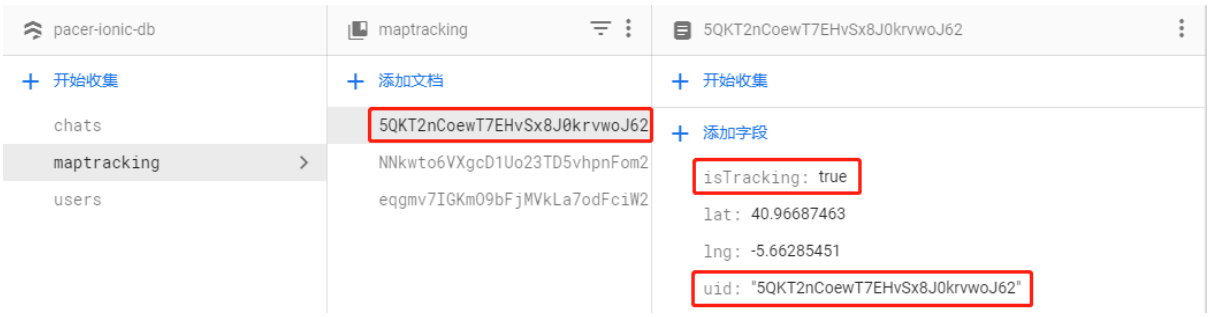


Ilustración 32. Ejemplo de la colección “maptracking”

- Collection chats - Document send user id - Collection receive user id - Document message sent time id

Como se muestra en la *Ilustración 33*, hemos guardado todos los conjuntos de los mensajes que envían los usuarios en una colección llamada “chats”. Como lo que quiere implementar es un *real-time chat* privado, la estructura de la base de datos es más complicada que otros.

Primero, la clave es distinguir claramente el remitente y el destinatario. Por lo tanto, cuando un usuario entra a una habitación privada disponiendo a chatear, ha de coger la id del usuario como id de remitente y la id de otro usuario con el que quiere hablar como la id de destinatario.

Segundo, ¿cómo se consigue la id de destinatario? Usando el *Session Storage*. Al entrar a una habitación, el usuario realmente ha creado una única “session storage” donde contiene la id del otro usuario.

Tercero, ha de poner la id de remitente como la id del documento, la id de destinatario como la id del subcolección (como se muestra en la *Ilustración 34*, según el documento oficial de Firebase, ahora es imposible conseguir la id de subcolección con alguna función de Firebase). Luego, para que la id del documento no sea generado automáticamente, hemos puesto el tiempo específico de un mensaje enviado como la id del documento, así que se puede controlar y encontrar los documentos fácilmente.

Últimamente, como todos las ids ya están bajo control, voy a la colección “users” a buscar los datos de usuarios específicos con ids correspondientes, por ejemplo, el nombre del usuario. Hasta ahora, ya tenemos una habitación completa de chat con información de mensajes y usuarios.

Ilustración 33. Ejemplo de la colección "chat"

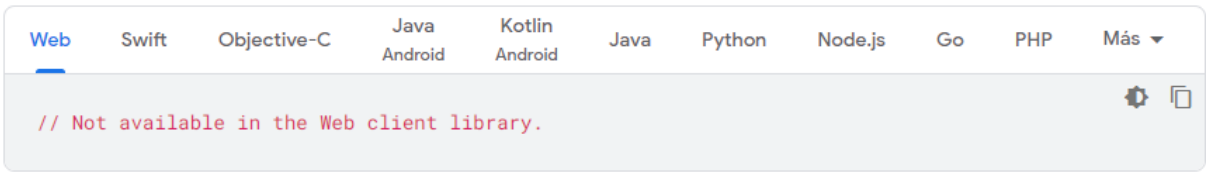


Ilustración 34. No se puede conseguir id de subcolección oficialmente

8.5 Sistema de autenticación de Firebase

Gracias a la ayuda de (Firebase, Inc., 2016) y (Firebase email authentication in Ionic 5), para realizar este sistema de autenticación conectándose con Firebase y almacenar todos los datos de usuario en Cloud Firestore, se implementa en los siguientes pasos:

a. En la página de registro, crear un formulario(*<form></form>*) con los campos necesarios para que el usuario lo rellene. Cada campo tiene su "formControlName" para identificarse. Coger los datos que se hayan rellenado usando la funcionalidad de "form", por ejemplo, "this.registerForm.value.email" es para coger el valor de email.

b. En el *shell*, usar el comando *$ionic g service* para crear un fichero de servicio donde se conecta a Firebase. Importar los paquetes necesarios de Angular/Firebase. Luego implementar la funcionalidad de Firebase "createUserWithEmailAndPassword" con la que se genera una cuenta usando el correo electrónico y la contraseña. Después de este paso, en la consola de Firebase ya aparece la cuenta creada. Y también en el apartado "console-aplication" de la página web aparecen los datos del usuario identificados en la sección "Local Storage" (A, 2016)

c. Para manipular los datos, entre las funciones locales y el firebase se necesita una interfaz exportada. Como se usarán la fecha de cumpleaños y el nombre real más adelante, aquí añade y pasa los valores de ellos también.

d. Volviendo al fichero de servicio, usar el paquete *AngularFirestore* y *AngularFirestoreDoucument* para crear una colección "users" donde almacenamos

todos los datos. Aquí el conjunto de datos tiene el formato de la interfaz "User" que hemos creado el paso anterior.

e. En la página de iniciar sesión, después de obtener los datos que haya introducido el usuario en la tabla, usar la funcionalidad de Firebase "signInWithEmailAndPassword" con la que se hace una comparación con los datos almacenado en el sistema de autenticación de Firebase. Obviamente, si se coinciden, le redirecciona al usuario al menú principal, en contrario, le da un aviso y se queda en la página de iniciar sesión.
f. En el *shell*, usar de nuevo el comando *$ionic g* guard para crear un fichero de guardia donde se vigila el estado del usuario globalmente. En el fichero "guard" se ven muchas líneas de códigos preconfigurados, lo que necesitamos modificar es añadir una condición dentro de la función "canActivate", la cual hemos definido en el servicio de autenticación. En ese fichero de servicio, también vigilamos el estado del usuario con el atributo "authState" del paquete *AngularFirebaseAuth* y si está identificado, pasamos el valor del usuario al *LocalStorage*. Según la condición mencionada antes, si el valor del usuario no es "null", lo que significa que existe, la guardia le deja pasar.
g. Adicionalmente, las funcionalidades como recuperación de contraseña y validación de correo también se realizan con el sistema de autenticación de Firebase en el fichero de servicio.

8.6 Localización con Google Maps

Como lo que se ha explicado en el apartado anterior Conexión con Google APIs, ha de utilizar unos APIs que ofrecen Google Maps para implementar el buscador de corredores cercanos.

Gracias a las inspiraciones que me han traído (J., 2019), (Ramalingam, 2019) y (Grimm, Building an Ionic 4 Firebase Location Tracker with Capacitor & Google Maps, 2019), hemos implementado esta funcionalidad en los siguientes pasos:

a. Importar los paquetes necesarios: todo lo de firebase, *Geolocation* de *ionic-native*, google.
b. Para mostrar el mapa en la página, usar identificador "#map" en html(vista DOM) + *ViewChild* en ts. *ViewChild* es un decorador de propiedades que configura una consulta de vista. El detector de cambios busca el primer elemento o la directiva que coincida con el selector en la vista DOM. Si la vista DOM cambia y un nuevo hijo

coincide con el selector, la propiedad se actualiza. Luego usar la función "getCurrentPosition()" del paquete geolocation para localizar al usuario mismo y centrar el mapa en esta región según el nivel de *zoom*.

c. Crear marcador usando "new google.maps.Marker()". Crear ventanita de información usando "new google.maps.InfoWindow()". El marcador le indica al usuario su posición en el mapa y la ventanita que está encima del marcador le muestra las informaciones adicionales de él.
d. Para exponer más de un marcador en el mapa en paralelo, ha de crear un contenedor que tiene disponible todas las informaciones geológicas. En el contenedor, distinguimos el propio usuario de los restos usuarios con distintos iconos de marcador e informaciones en la ventanita, los cuales predefinimos públicamente.
e. Como en el apartado anterior de autenticación, se necesita otra interfaz exportada con la que conectamos con Firebase y se actualizan los datos de localización.
f. En el *shell*, crear un fichero de servicio donde se realiza lo fundamental de Firebase: *CRUD(Create, Read, Update, Delete)* con la ayuda de (Dechalert, 2019) y (Jabif, 2019) para datos de localización. Con *AngularFirestore* se crea una colección "maptracking" donde se actualizan las posiciones de usuario al correr. Cuando se necesita leer los datos dentro de la colección, ha de usar la función "snapshotChanges()" para demostrar un resumen de datos.
g. Para que la localización de un usuario se actualiza en tiempo real, ha de utilizar la función "watchPosition" de *Geolocation* con la que se suscribe a los movimientos y se vigila todos los momentos.
h. Aparte de la latitud y la longitud, en la colección también se ha puesto la variable "uid" y "isTracking".

   El "uid" se refiere al identificador del usuario actual que ya se ha almacenado en el *LocalStorage* al iniciar la sesión(después de iniciar la sesión y al mantenerse el estado, los datos de usuario siempre están en el *LocalStorage*). Se registra este identificador al entrar en el mapa, se guarda en la colección "maptracking", se puede conseguir desde la colección "maptracking" cuando el usuario solicita más información, luego se puede buscar en otra colección "users", coger las informaciones correspondientes a este identificador y mostrarlas en la página, con la ayuda de la documentación oficial de (Firebase Inc., 2019). Entonces, desde el punto de vista del usuario, cuando

cliquea un marcador en el mapa, tanto uno de su mismo como uno de los otros, le saldrá la información del nombre y de la edad.

El "isTracking" se refiere a una variable booleana con la que un usuario controla la visibilidad de su marcador en el mapa. Cuando el usuario cambia su visibilidad tocando el botón, en la colección la variable "isTracking" cambia entre "true" y "false".

Cuando se cambia a "false", ha de la función "unsubscribe()" para dejar de suscribirse a los movimientos.

8.7 *Chat* en tiempo real

Como se puede encontrar por Internet, existen muchos tutoriales para establecer un *chat* grupal ya que es relativamente fácil en comparación con un *chat* privado entre dos personas. Técnicamente se requiere más restricciones e identificaciones para entrar a una habitación de *chat*.

Con el objetivo de realizar la funcionalidad de chatear en tiempo real y la ayuda de tutoriales como (RobotSolar, 2019), (Fire Focus, 2020) y (Grimm, How to Create Ionic 4 Chat Bubbles with Elastic Textarea, 2019), las claves principales son:

- Establecer conexión privada entre dos usuarios usando *Local Storage* y *Session Storage*
- Distinguir el remitente y el destinatario usando **ngIf* de Angular
- Almacenar mensajes identificados por tiempo usando *Cloud Firestore*
- Mostrar mensajes leídos de la base de datos ordenando según el tiempo

a. Al iniciar la sesión, los datos del usuario identificado ya están almacenados en el *LocalStorage*, por lo que es fácil definir quién es el remitente. Para definir también quién es el destinatario, usamos el *Session Storage* para guardar los datos del otro usuario. En el apartado "console-aplication" de la página web se puede observar que en la sección "Session Storage" se agregará la información correspondiente cuando un destinatario en el mapa está seleccionado.
b. Puesto que ya tenemos las ids de los hablantes, cuando se envian los mensajes, se puede identificar que están enviado de quién a quién intercambiando las ids. En la colección "chats" de Firebase que hemos creado previamente, establecemos un *path*

de documentos para identificar la dirección de un mensaje, como una flecha con indicaciones.

Por ejemplo, cuando el usuario A envía un mensaje al usuario B, el path correcto desde el punto de vista de A en la colección "chats" será "chats - id de A - id de B". Y desde el punto de vista de B será "chats - id de B - id de A".

c. Puesto que ya tenemos mensajes identificados con los campos mencionados en Firebase, podemos distinguirlos visualmente en la vista DOM con la forma **ngIf* de Angular. **ngIf* se usa para establecer condiciones en fichero html, en caso de que la persona que envia un mensaje está identificado como el propio usuario, se coloca el mensaje por el lado derecho como mensaje de remitente, en caso contrario, se coloca por el lado izquierdo.

a. Adicionalmente, para que todos las ids de documento tengan sentidos y luego las podré manipular, hemos puesto el tiempo de envío como la id de cada mensaje. Será una serie de números que representa el instante cuando el usuario envía el mensaje. Entonces el *path* de un mensaje en la colección "chats" será:
"chats - id de remitente - id de destinatario - id de tiempo - contenidos"

b. Localmente, todos los mensajes se almacenan en un contenedor y se muestran cada vez que un usuario entra a una habitación. Se ordenan según el tiempo que se han registrado al enviarse.

8.8 Capacitor Cloud Messaging Push Notification

Enfrentándose con las competencias crueles en el mercado actual, la mayoría de empresas implementan la funcionalidad de notificación a sus aplicaciones, con la cual atrae a sus usuarios constantemente. Existen varias plataformas donde se puede realizar la interacción de notificación entre el servicio y el cliente con sus herramientas, en nuestro caso lo hemos implementado utilizando la herramienta *Cloud Messaging* de la plataforma Firebase y los plugins como "PushNotification" de Capacitor. Aprendiendo de la documentación oficial (Ionic Team, 2020), (Ionic Team, 2020) y más tutoriales como (Delaney, 2017), (Jerga, 2018), (Lorca, 2019), (Grimm, The Push Notifications Guide for Ionic & Capacitor, 2020), (android - Push Notifications when app is closed, 2014), La implementación actual es lo básica pero funciona bien no importa si está en pantalla activa, o en segundo plano, o está apagada completamente, siguiendo los siguientes pasos:

a. Importar los paquetes necesarios de Capacitor sobre notificación.

b. Realizar el registro de *push-notification* con Google vía *FCM (Firebase Cloud Messaging)*
c. Conseguir el *token* de autenticación del dispositivo, que se reconoce como el único identificador interno de un dispositivo móvil. Lo guardo en una colección de Cloud Firestore para usar luego.
d. Gestionar la acción tras recibir la notificación cuando la aplicación está en la pantalla activa y la acción cuando la aplicación está en estado desactivo o muerto.
   Atención: el uso de estos dos escuchadores simultáneamente puede resultar un caso de duplicación de acciones. Esto se debe a que teniendo la aplicación abierta y presionando sobre la notificación se lanzarán tanto *pushNotificationReceived* como *pushNotificationActionPerform* dado que se cumplen ambas casuísticas.
e. Realizar el envío de notificación a todos los usuarios desde la base de datos. Realizar la configuración de la notificación en el apartado *Cloud Messaging* en la consola de Firebase.
f. La notificación deberá funcionar después de haber implementado acciones sobre tanto el estado activo como el estado desactivo de la aplicación. Sin embargo, cuando desconectamos el móvil del ordenador y apagamos la aplicación completamente (no en segundo plano sino apagarlo forzado), en los dispositivos Android ya no suelen recibir notificaciones, aunque la opción "open notification" está activa. Eso ocurre debido al *DOZE mode* y la optimización de batería por defecto en sistema *Android 8.0 Oreo*. Si el usuario quiere recibir notificación de nuestra aplicación como whatsapp, simplemente desactiva la optimización de batería según los siguientes pasos:
   Settings>> Battery >> battery Optimization >> find your app >> select >> if optimized click on don't optimize >> try pushing notification

9. Conclusiones

Gracias al trabajo de fin de máster, que podido desarrollar en una empresa real, he podido poner en práctica y profundizar todo lo que he aprendido en el máster, con las asignaturas como Programació de Telèfons i Dispositius Mòbils Autònoms, Viabilitat de Projectes Empresarials Innovadors y aprender otras nuevas, no solo en el ámbito del desarrollo de una aplicación sino también, de otros departamentos como *marketing*, negocios, financiación….

Como he comentado anteriormente, mi posición en la empresa me ha permitido la creación de una aplicación híbrida para dispositivos móviles que consisten en funcionalidades con mucha variedad como buscador de corredores, comunicación en tiempo real, registro de actividad física y etc.

Este trabajo cumple todos los objetivos que he planificado con la empresa inicialmente y la mayoría de las funcionalidades se han desarrollado hasta la fecha actual. La aplicación integra diversas plataformas como Firebase y Google Cloud.

Personalmente, la realización de este trabajo me ha enriquecido mucho tanto en el aprendizaje de los nuevos conocimientos como en la experiencia de trabajar en equipo y trabajar en empresa real. Y no solo eso, sino en una empresa como una start-up, que posee unas características muy especiales. Siempre considero que lo que aprendemos en el aula y lo que afrontamos en el trabajo se refuerzan mutuamente y me alegro de que la implementación de este trabajo me haya confirmado esta teoría. Técnicamente esta aplicación híbrida que he realizado con Ionic y Angular es totalmente diferente con las experiencias que he tenido con plataforma Android, por eso me ha dado mucha satisfacción construirla desde la primera pantalla en blanco hasta donde es hoy en día. Además, me ilusiona especialmente, tener la responsabilidad de ofrecer a los corredores una herramienta diferente a lo que existe en el mercado.

Finalmente, quería agradecer a todas aquellas personas que me han ayudado a realizar este trabajo y a quienes me han dado la oportunidad de desarrollarme de manera personal y profesional en el mercado laboral

9.1 Extenciones futuras

Según la planificación del proyecto de la empresa, esta aplicación posee un gran potencial de expansión y renovación de sus funcionalidades. Revisando opciones del menú principal, faltarán implementar más opciones como; *calendario de competiciones, entrenamiento personalizado y consultas*, las cuales me gustaría seguir creando después de haber realizado el TFM

Por la parte del calendario de competición, realizaremos una interfaz como se muestra en la *Ilustración 35*, que es un esbozo que hicimos anteriormente, donde el usuario podrá visualizar los eventos deportivos celebrados en España.

También hemos pensado establecer la colaboración con las organizaciones que celebran los eventos con una forma de llevar a nuestros usuarios a la página *web* donde apuntan los eventos si les interesan y al volver los nombres de participantes aparecen en la lista de cada evento.

Respecto al entrenamiento personalizado, realizaremos una parte de la interfaz como se muestra en la *Ilustración 36* donde el usuario podrá editar el calendario interactivo marcando fechas de hacer actividades y planificando su entrenamiento personalizado. Para los usuarios novatos, les recomendaremos un tutorial básico de entrenamiento durante una semana para que puedan iniciar su carrera siguiendo el camino correcto sin sufrir lesiones. Para los usuarios expertos, podrán saltar el tutorial y planificar su entrenamiento con una forma totalmente libre. Con la funcionalidad de *registrar actividad física*, vamos recopilando los datos físicos como ritmo cardíaco del usuario cada vez que corre y con los datos que conseguimos les daremos recomendaciones profesionales integrando la tecnología de inteligencia artificial. Además, para todos los corredores, en cualquier momento si se les ocurre cualquier duda relacionada con el entrenamiento de correr, por ejemplo, de su planificación, o lesión, les facilitaremos un acceso con el que podrán contactar con unos expertos deportivos.

Adicionalmente, como la mayoría de las aplicaciones, implementaremos una opción de *configuración*, donde el usuario podrá cambiar el idioma preferente, inactivar *push* notificación, contactar con nosotros para realizar sugerencias sobre la aplicación.

Hoy en día, tanto en el mundo tecnológico como en el deportivo, las cosas van cambiando rápidamente. Por tanto, las actualizaciones de la aplicación serán dinámicas, siempre valorando las opiniones de los usuarios.

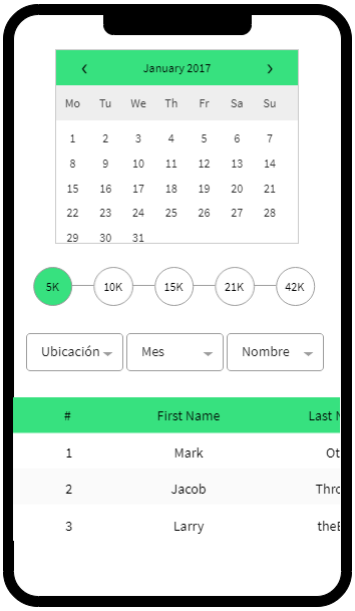


Ilustración 35. Esbozo para calendario de competición

Ilustración 36. Esbozo para calendario de entrenamiento

10. Bibliografías

1. AKumar BJaya. (14 noviembre 2016). Javascript - Local storage in Angular 2. Recuperado: 26 april 2020, Disponible a: Stack Overflow: https://stackoverflow.com/questions/40589730/local-storage-in-angular-2

2. Angular Team. (07 junio 2019). Angular Overview. Recuperado: 01 marzo 2020, disponible a: Angular.io: https://angular.io/

3. Cordova Team. (2015). Resumen. Recuperado: 25 marzo 2020, disponible a: Apache Cordova: https://cordova.apache.org/docs/es/latest/guide/overview/

4. danielstham and jamesdiego. (25 octubre 2017). AngularFire The official library for Firebase and Angular. Recuperado: 20 april 2020, disponible a : GitHub: https://github.com/angular/angularfire/tree/7eb3e51022c7381dfc94ffb9e12555065f060639#angularfire

5. DechalertAphinya. (09 mayo 2019). How to CRUD in Angular + Firebase Firestore. Recuperado: 20 april 2020, disponible a: Medium: https://medium.com/madhash/how-to-crud-in-angular-firebase-firestore-456353d7c62

6. DelaneyJeff. (01 agosto 2017). Push Notifications on the Web. Recuperado: 25 junio 2020, disponible a: Fireship.io: https://fireship.io/lessons/push-messages-with-firestore/

7. Drifty, Inc. (2016). Ionic Cross-Platform Mobile App Development. Recuperado: 01 marzo 2020, disponible a: Ionic Framework: https://ionicframework.com/#

8. ESTUDIO DE GRUPO NN. (01 julio 2019). *DEPORTE Y NEGOCIO*. Recuperado: 15 septiempre 2020, de EXPANSIÓN: disponible a : https://www.expansion.com/directivos/deporte-negocio/2019/07/01/5d19c509468aebbe0e8b45a9.html

9. Fire Focus. (04 april 2020). Ionic Firebase chat - User to User chat (private / one to one chat). Recuperado: 10 julio 2020, disponible a: Youtube: https://www.youtube.com/watch?v=NiR_niTNHMo

10. Firebase Channel. (27 julio 2020). Get to know Cloud Firestore Playlist. Recuperado: 17 april 2020, disponible a: Youtube: https://www.youtube.com/playlist?list=PLl-K7zZEsYLluG5MCVEzXAQ7ACZBCuZgZ

11. Firebase Inc. (19 mayo 2016). Elige una base de datos: Cloud Firestore o Realtime Database. Recuperado: 25 april 2020, disponible a: Firebase Docs: https://firebase.google.com/docs/database/rtdb-vs-firestore

12. Firebase Inc. (03 diciembre 2019). Obtén datos con Cloud Firestore. Recuperado: 17 april 2020, disponible a: Firebase Docs: https://firebase.google.com/docs/firestore/query-data/get-data

13. Firebase, Inc. (19 mayo 2016). Firebase Auth. Recuperado: 18 april 2020, disponible a: Firebase Docs: https://firebase.google.com/docs/auth/

14. Firebase, Inc. (2016). Firebase Overview. Recuperado: 16 april 2020, disponible a: Firebase: https://firebase.google.com/

15. Google. (noviembre 2011). APIs de geolocalización. Recuperado: 14 mayo 2020, disponible a: Google Maps Platform | Google Cloud: https://cloud.google.com/maps-platform/

16. GrimmSimon. (28 mayo 2019). Building an Ionic 4 Firebase Location Tracker with Capacitor & Google Maps. Recuperado: 15 mayo 2020, disponible a: Youtube: https://www.youtube.com/watch?v=Sq0NbvQihrk

17. GrimmSimon. (07 mayo 2019). How to Create Ionic 4 Chat Bubbles with Elastic Textarea. Recuperado: 15 julio 2020, disponible a: Youtube: https://www.youtube.com/watch?v=XRkaLmfTQcY

18. GrimmSimon. (14 julio 2020). The Push Notifications Guide for Ionic & Capacitor. Recuperado: 26 junio 2020, disponible a: Devdactic Code to Success: https://devdactic.com/push-notifications-ionic-capacitor/

19. Ionic Team. (26 noviembre 2019). Android Development. Recuperado: 01 marzo 2020, disponible a: Ionic Docs: https://ionicframework.com/docs/developing/android

20. Ionic Team. (26 noviembre 2019). Gestures - Ionic Documentations. Recuperado: 01 junio 2020, disponible a: Ionic Docs: https://ionicframework.com/docs/utilities/gestures

21. Ionic Team. (26 noviembre 2019). Live Reload. Recuperado: 05 junio 2020, disponible a: Ionic Docs: https://ionicframework.com/docs/cli/livereload

22. Ionic Team. (april 2020). Capacior:Cross-Platform native runtime for web apps. Recuperado: 15 marzo 2020, disponible a: Capacitorjs: https://capacitorjs.com/

23. Ionic Team. (2020). Ionicons:The premium icon pack for Ionic Framework. Recuperado: 01 april 2020, disponible a: Ionicons 5.1.2: https://ionicons.com/

24. Ionic Team. (2020). Known Incompatible Cordova Plugins - Capacitor. Recuperado: 16 mayo 2020, disponible a: Capacitor Docs: https://capacitorjs.com/docs/cordova/known-incompatible-plugins

25. Ionic Team. (2020). Push Notifications - Capacitor. Recuperado: 25 junio 2020, disponible a: Capacitor Docs v2.3.0: https://capacitorjs.com/docs/apis/push-notifications

26. Ionic Team. (2020). Using Push Notifications with Firebase in an Ionic + Angular App. Recuperado: 25 junio 2020, disponible a: Cpacitor Docs: https://capacitorjs.com/docs/guides/push-notifications-firebase

27. J.Didin. (30 agosto 2019). Ionic 3, Angular 5, Firebase and Google Maps Location Tracking. Recuperado: 01 junio 2020, disponible a: djamware: https://www.djamware.com/post/5a48517280aca7059c142972/ionic-3-angular-5-firebase-and-google-maps-location-tracking

28. JabifDayana. (16 april 2019). Angular CRUD with Firebase. Recuperado: 20 april 2020, disponible a: Angular Templates: https://angular-templates.io/tutorials/about/angular-crud-with-firebase

29. JergaFilip. (25 octubre 2018). How to get push notifications working with Ionic 4 and Firebase. Recuperado: 26 junio 2020, disponible a: freeCodeCamp: https://www.freecodecamp.org/news/how-to-get-push-notifications-working-with-ionic-4-and-firebase-ad87cc92394e/

30. JIMÉNEZCARLOS. (16 octubre 2019). 5.250 millones de euros en 2022: as íse cimenta el verdadero negocio del running y del triatlón. Recuperado: 28 septiembre 2020, disponible a: Runner's world: https://www.runnersworld.com/es/noticias-running/a29438016/negocio-running-triatlon/

31. LorcaFuenzalidaBranko. (02 noviembre 2019). Entendiendo el Plugin Push Notifications de Capacitor. Recuperado: 27 junio 2020, disponible a: Medium: https://medium.com/@brankofuenzalida/entendiendo-el-plugin-push-notifications-de-capacitor-8ca84cdd8d38

32. Max Lynch (CEO y co-fundador de Ionic). (26 noviembre 2019). Cordova vs Capacitor. Recuperado: 05 april 2020, disponible a: Ionic Docs: https://ionicframework.com/resources/articles/capacitor-vs-cordova-modern-hybrid-app-development

33. NetkowMatt. (08 april 2020). Announcing Capacitor 2.0. Recuperado: 10 april 2020, disponible a: The Ionic Blog: https://ionicframework.com/blog/announcing-capacitor-2-0/

34. Noun Project Inc. (2020). Noun Project:Free Icons & Stock Photos For Everything. Recuperado: 12 junio 2020, disponible a: Noun Project: https://thenounproject.com

35. RamalingamKumarSathish. (02 julio 2019). Ionic 4 Google Map Playlist. Recuperado: 28 mayo 2020，disponible a: Youtube: https://www.youtube.com/playlist?list=PLV1euzd9ziJIjb3yNGXR_w0HK6xhqqNlV

36. RobotSolar. (01 april 2019). Ionic + Firebase app chat playlist. Recuperado: 01 julio 2020，disponilble a: Youtube: https://www.youtube.com/playlist?list=PL0az06iNsMl4OSbuxkstjR8e2lQDj1DVk

37. SharmaAditya. (none). Firebase email authentication in Ionic 5. Recuperado: 25 april 2020，disponible a: Enappd: https://enappd.com/blog/firebase-email-authentication-in-ionic-apps/153/

38. SoueidanSara. (09 agosto 2013). Building a Circular Navigation with CSS Transforms. Recuperado: 29 april 2020，disponible a: Codrops: https://tympanus.net/codrops/2013/08/09/building-a-circular-navigation-with-css-transforms/

39. Steffen. (07 noviembre 2018). angular - Ionic 4 costum components. Recuperado: 16 marzo 2020，disponible a: Stack Overflow: https://stackoverflow.com/questions/53185901/ionic-4-custom-components

40. user3757628. (19 junio 2014). android - Push Notifications when app is closed. Recuperado: 29 junio 2020，disponible a: Stack Overflow: https://stackoverflow.com/questions/24313539/push-notifications-when-app-is-closed

41. ZakDale. (04 febrero 2020). Generate App Icon and Splash Screen Images for Ionic Framework using Capacitor. Recuperado: 06 junio 2020，disponible a: Medium: https://medium.com/@dalezak/generate-app-icon-and-splash-screen-images-for-ionic-framework-using-capacitor-e1f8c6ef0fd4

42. 小叶子 leaveescy. (08 agosto 2018). Android color 颜色-色号总结. Recuperado: 15 marzo 2020，disponible a: 博客园: https://www.cnblogs.com/leavescy/p/9441827.html

11. Anexos

Anexo 1. Cambio de icono y splash screen para proyecto de Capacitor

Para un proyecto de capacitor, con la ayuda de (Zak, 2020), hay que aplicar las siguientes instrucciones a fin de sustituir el icono por defecto por el icono diseñado:

1) Utilizar capacitor-resources: crear un js fichero en el directorio root llamado resources.js
2) En el fichero package.json, bajo el apartado “scripts”, añadir la instrucción "resources": "capacitor-resources -p android,ios && node resources.js"
3) Escribir los siguientes códigos simulando el proceso de cambiar icono en capacitor

const fs = require('fs'); function copyImages(sourcePath, targetPath, images) { for (const icon of images) { let source = sourcePath + icon.source; let target = targetPath + icon.target; fs.copyFile(source, target, err => { if (err) throw err; console.log(`${source} >> ${target}`); }); } } const SOURCE_ANDROID_ICON = 'resources/android/icon/'; const SOURCE_ANDROID_SPLASH = 'resources/android/splash/'; const TARGET_ANDROID_ICON = 'android/app/src/main/res/'; const TARGET_ANDROID_SPLASH = 'android/app/src/main/res/'; const ANDROID_ICONS = [{ source: 'drawable-ldpi-icon.png', target: 'drawable-hdpi-icon.png' }, { source: 'drawable-mdpi-icon.png', target: 'mipmap-mdpi/ic_launcher.png' }, { source: 'drawable-mdpiicon.png', target: 'mipmap-mdpi/ic_launcher_round.png' }, { source: 'drawable-mdpi-icon.png', target: 'mipmapmdpi/ic_launcher_foreground.png' }, { source: 'drawable-hdpiicon.png', target: 'mipmap-hdpi/ic_launcher.png' }, { source: 'drawable-hdpi-icon.png', target: 'mipmaphdpi/ic_launcher_round.png' }, { source: 'drawable-hdpi-icon.png', target: 'mipmap-hdpi/ic_launcher_foreground.png' }, { source: 'drawable-xhdpi-icon.png', target: 'mipmap-xhdpi/ic_launcher.png' }, { source: 'drawable-xhdpi-icon.png', target: 'mipmapxhdpi/ic_launcher_round.png' }, { source: 'drawable-xhdpi-icon.png', target: 'mipmap-xhdpi/ic_launcher_foreground.png' }, { source: 'drawable-xxhdpi-icon.png', target: 'mipmap-xxhdpi/ic_launcher.png' }, { source: 'drawable-xxhdpi-icon.png', target: 'mipmapxxhdpi/ic_launcher_round.png' }, { source: 'drawable-xxhdpi-icon.png', target: 'mipmap-xxhdpi/ic_launcher_foreground.png' }, { source: 'drawable-xxxhdpi-icon.png', target: 'mipmapxxxhdpi/ic_launcher.png' }, { source: 'drawable-xxxhdpi-icon.png', target: 'mipmap-xxxhdpi/ic_launcher_round.png' }, { source: 'drawable-xxxhdpi-icon.png',

target: 'mipmapxxxhdpi/ic_launcher_foreground.png' }]; const ANDROID_SPLASHES = [{ source: 'drawable-land-mdpi-screen.png', target:

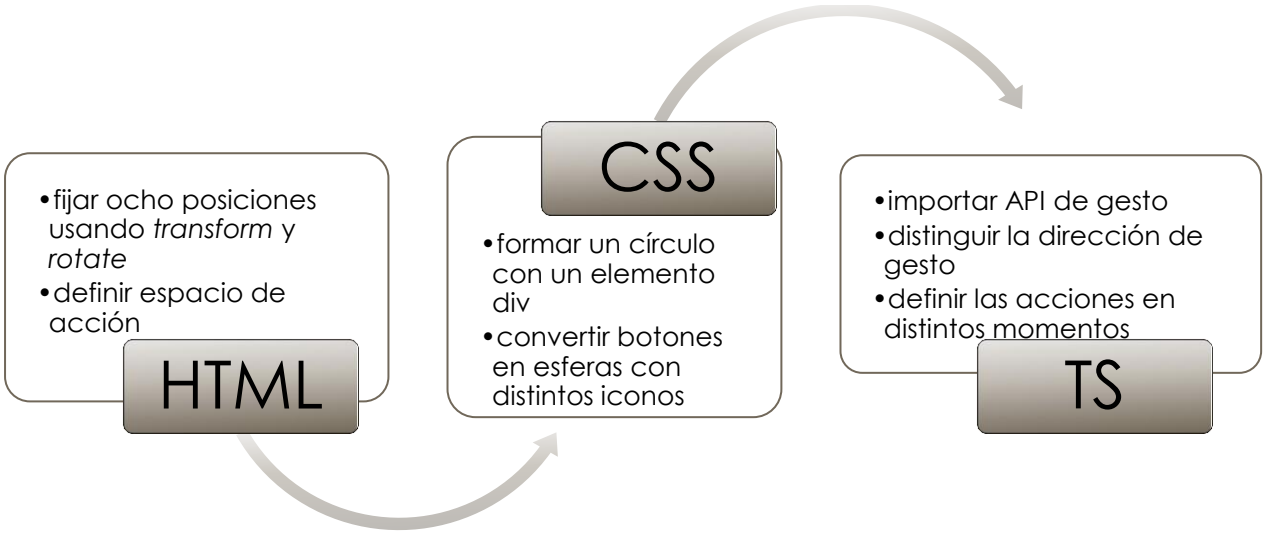
Facultad de Informática de Barcelona, Universidad Politécnica de Cataluña | 60

'drawable/splash.png' }, { source: 'drawable-land-mdpi-screen.png', target: 'drawable-land-mdpi/splash.png' }, { source: 'drawable-landhdpi-screen.png', target: 'drawable-land-hdpi/splash.png' }, { source: 'drawable-land-xhdpi-screen.png', target: 'drawable-landxhdpi/splash.png' }, { source: 'drawable-land-xxhdpi-screen.png', target: 'drawable-land-xxhdpi/splash.png' }, { source: 'drawableland-xxxhdpi-screen.png', target: 'drawable-landxxxhdpi/splash.png' }, { source: 'drawable-port-mdpi-screen.png', target: 'drawable-port-mdpi/splash.png' }, { source: 'drawable-porthdpi-screen.png', target: 'drawable-port-hdpi/splash.png' }, { source: 'drawable-port-xhdpi-screen.png', target: 'drawable-portxhdpi/splash.png' }, { source: 'drawable-port-xxhdpi-screen.png', target: 'drawable-port-xxhdpi/splash.png' }, { source: 'drawableport-xxxhdpi-screen.png', target: 'drawable-portxxxhdpi/splash.png' }]; copyImages(SOURCE_ANDROID_ICON, TARGET_ANDROID_ICON, ANDROID_ICONS); copyImages(SOURCE_ANDROID_SPLASH, TARGET_ANDROID_SPLASH, ANDROID_SPLASHES);

4) En la consola, indicar `npm run resources`

Después de resolver todo lo anterior, ya se puede establecer o copiar el proyecto a android studio y animarlo en un dispositivo o un simulador con el icono y la pantalla de entrada personalizada.

Anexo 2. Implementación para el menú principal



```
.circle {
  display:block;
  height: 350px;
  width: 350px;
  border-radius: 50%;
  border: 1px solid gray;
  position:absolute;
  left:-175px;
  top:270px;
}
```

Ilustración 37. Css para formar un círculo

```
<div #control style="width:100%;min-height:100%">
    <div class="rotate" #shop style="transform:rotate(-90deg) translateX(300px) rotate(90deg)" tappable routerLink="./shopmenu" >
    | <ion-icon src="assets/icon-slide/shopping.svg" class="round-button"></ion-icon>
    </div>
    <div class="rotate" #message style="transform:rotate(-135deg) translateX(300px) rotate(135deg)" tappable routerLink="../chatroom">
    | <ion-icon src="assets/icon-slide/message.svg" class="round-button"></ion-icon>
    </div>
    <div class="rotate" #calendar style="transform:rotate(-180deg) translateX(300px) rotate(180deg)" tappable>
    | <ion-icon src="assets/icon-slide/calendar.svg" class="round-button"></ion-icon>
    </div>
    <div class="rotate" #settings style="transform:rotate(-225deg) translateX(300px) rotate(225deg)" tappable>
    | <ion-icon name="settings-outline" style="--ionicon-stroke-width: 16px" class="round-button"></ion-icon>
    </div>
    <div class="rotate" #clock style="transform:rotate(-270deg) translateX(300px) rotate(270deg)" tappable routerLink="../maptracking">
    | <ion-icon src="assets/icon-slide/clock.svg" class="round-button"></ion-icon>
    </div>
    <div class="rotate" #list style="transform:rotate(-315deg) translateX(300px) rotate(315deg)" tappable>
    | <ion-icon src="assets/icon-slide/list.svg" class="round-button"></ion-icon>
    </div>
    <div class="rotate" #perfil style="transform:rotate(0deg) translateX(300px) rotate(0deg)" tappable routerLink="./perfil">
    | <ion-icon src="assets/icon-slide/user.svg" class="round-button"></ion-icon>
    </div>
    <div class="rotate" #location style="transform:rotate(-45deg) translateX(300px) rotate(45deg)" tappable routerLink="./map">
    | <ion-icon src="assets/icon-slide/search_runner.svg" class="round-button"></ion-icon>
    </div>
```

Ilustración 38. Colocación de ocho esferas

```
import { Component, ViewChild, ElementRef, OnInit, AfterViewInit, Renderer2 } from '@angular/core';
import { Gesture, GestureController, AlertController } from '@ionic/angular';
```

Ilustración 39. Los paquetes de gesto

```
if((detail.currentY - detail.startY) < 0){      //up
  this.degree++;
}else{      //down
  this.degree--;
  if(this.degree < 0){
    this.degree = 360 + this.degree;
  }
}
```

Ilustración 40. Detección de gesto

```
if(((this.degree>=0)&&(this.degree<=22.5)) || ((this.degree>337.5)&&(this.degree<=360))){
  this.degree = 0;
  this.renderer.setStyle(this.perfil.nativeElement,"transform",`rotate(0deg) translateX(300px) rotate(0deg)`);
  this.renderer.setStyle(this.location.nativeElement,"transform",`rotate(-45deg) translateX(300px) rotate(45deg)`);
  this.renderer.setStyle(this.shop.nativeElement,"transform",`rotate(-90deg) translateX(300px) rotate(90deg)`);
  this.renderer.setStyle(this.message.nativeElement,"transform",`rotate(-135deg) translateX(300px) rotate(135deg)`);
  this.renderer.setStyle(this.calendar.nativeElement,"transform",`rotate(-180deg) translateX(300px) rotate(180deg)`);
  this.renderer.setStyle(this.settings.nativeElement,"transform",`rotate(-225deg) translateX(300px) rotate(225deg)`);
  this.renderer.setStyle(this.clock.nativeElement,"transform",`rotate(-270deg) translateX(300px) rotate(270deg)`);
  this.renderer.setStyle(this.list.nativeElement,"transform",`rotate(-315deg) translateX(300px) rotate(315deg)`);
  this.changeTitle("Mi Perfil");
```

Ilustración 41. Ejemplo de acción acabada

Anexo 3. Implementación de sistema de autoidentificación


```
<form [formGroup]="registerForm" (ngSubmit)="register()" novalidate>
  <ion-item>
    <ion-input clearInput placeholder="Dirección de correo electrónico*" type="email" formControlName="email"></ion-input>
  </ion-item>
```

Ilustración 42. formulario para registrar

```
await this.authSvc.onRegister(this.registerForm.value.email, this.registerForm.value.password,
                              this.registerForm.value.name, this.registerForm.value.birthday);
```

Ilustración 43. Coger valores desde el formulario

```
import { AngularFireAuth } from '@angular/fire/auth';
import { AngularFirestore, AngularFirestoreDocument } from '@angular/fire/firestore';
```

Ilustración 44. Importar paquete de AngularFire

```
//register
onRegister(email:string, password:string, firstName:string, birthday:Date){
  return new Promise((resolve, reject)=>{
    this.afAuth.createUserWithEmailAndPassword(email, password)
      .then((result) => {
      /* Call the SendVerificaitonMail() function when new user sign
      up and returns promise */
        this.SendVerificationMail();
        result.user.updateProfile({
          displayName:firstName,
        }).then(()=>{
          console.log(result.user);
          this.birthday = birthday;
          this.SetUserData(result.user);
        })
        resolve(result)
    }).catch(err => reject(err))
  })
}
```

Ilustración 45. Función para hacer registro

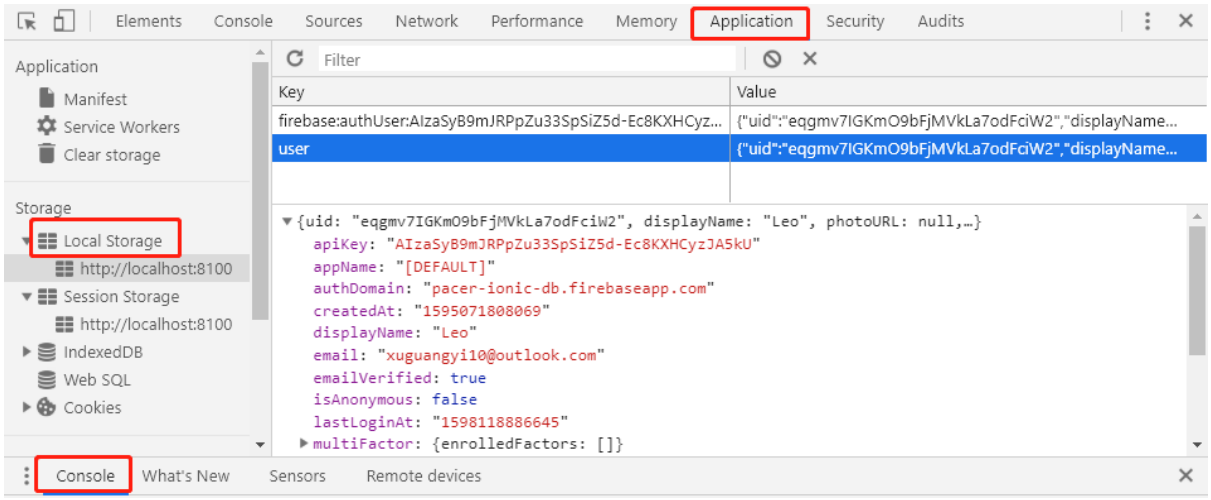


Ilustración 46. Interfaz de Local Storage

```typescript
export interface User {
  uid: string;
  email: string;
  displayName: string;
  emailVerified: boolean;
  photoURL?: string;
  deviceID?: string;
  birthday: Date;
}
```

Ilustración 47. Interfaz exportado "User"

```typescript
/* Setting up user data when sign in with username/password,
sign up with username/password and sign in with social auth
provider in Firestore database using AngularFirestore + AngularFirestoreDocument service */
SetUserData(user) {
  const userRef: AngularFirestoreDocument<any> = this.afs.doc(`users/${user.uid}`);
  const userData: User = {
    uid: user.uid,
    email: user.email,
    displayName: user.displayName,
    photoURL: user.photoURL,
    emailVerified: user.emailVerified,
    deviceID: this.device.uuid,
    birthday: this.birthday
  }
  return userRef.set(userData, {
    merge: true
  })
}
```

Ilustración 48. Función para almacenar datos en Cloud Firestore

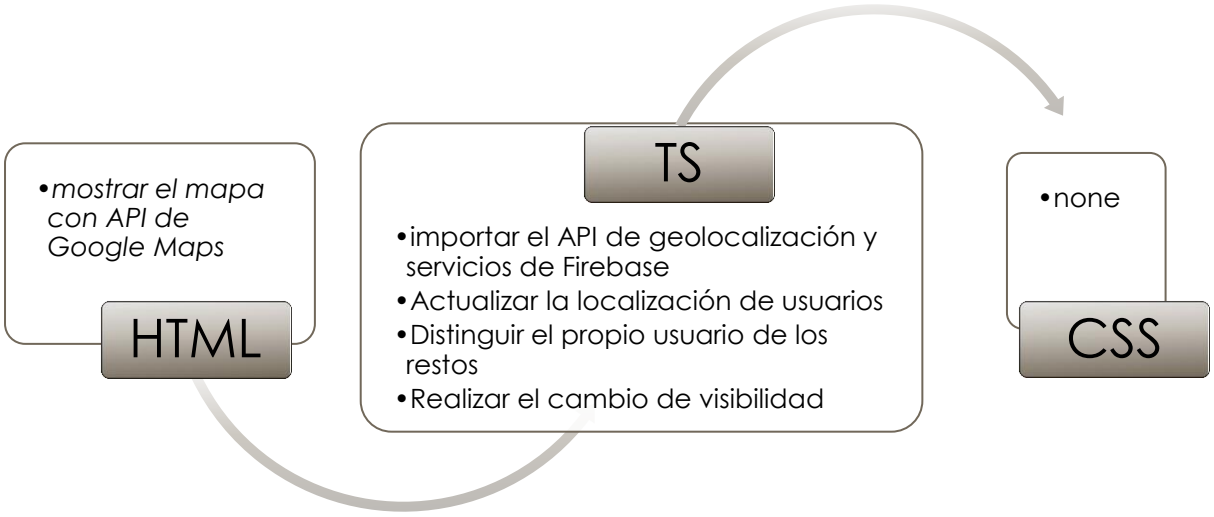
```typescript
//login
async onLogin(email, password){
  return new Promise((resolve, reject)=>{
    this.afAuth.signInWithEmailAndPassword(email, password)
      .then((result) => {
      console.log(result);
      this.router.navigate(['main']);
    }).catch(async err =>{
      reject(err);
      const alert = await this.alertCtrl.create({
        header: 'Algo va mal...',
        message: 'El usuario no existe o la contraseña no está correcta.',
        buttons: ['OK']
      });
      await alert.present();
    })
  })
}
```

Ilustración 49. Función para hacer login

```
import { Injectable } from '@angular/core';
import { CanActivate, ActivatedRouteSnapshot, RouterStateSnapshot, UrlTree } from '@angular/router';
import { Observable } from 'rxjs';
import { AuthService } from '../services/auth.service';
import { Router } from '@angular/router';
@Injectable({
  providedIn: 'root'
})
export class AuthGuard implements CanActivate {
  constructor(private authSvc:AuthService,private router:Router) {}
  canActivate(
    next: ActivatedRouteSnapshot,
    state: RouterStateSnapshot): Observable<boolean | UrlTree> | Promise<boolean | UrlTree> | boolean | UrlTree {
    if(this.authSvc.isLoggedIn){
      return true;
    }
    console.log("Access denied");
    this.router.navigateByUrl('/login');
    return false;
  }

}
```

Ilustración 50. Implementación en fichero "guard"

Anexo 4. Implementación de geolocalización en tiempo real




```
import * as firebase from 'firebase';
import { Geolocation } from '@ionic-native/geolocation/ngx';
declare var google: any;
```

Ilustración 51. Importar paquetes necesarios para geolocalización

```
this.geolocation.getCurrentPosition().then((resp) => {
  this.myLat = resp.coords.latitude;
  this.myLng = resp.coords.longitude;
  let mylocation = new google.maps.LatLng(resp.coords.latitude,resp.coords.longitude);
  this.map = new google.maps.Map(this.mapElement.nativeElement, {
    zoom: 14,
    center: mylocation
  });
});
```

Ilustración 52. Función para conseguir localización actual

```
addMarker(location, image, content, id, name) {
  const marker = new google.maps.Marker({
    position: location,
    map: this.map,
    icon: image,
    title: "Hola"
  });

  const infoWindow = new google.maps.InfoWindow({
    content:content
  });

  google.maps.event.addListener(marker,'click',function(){
    this.map.setZoom(20);
    this.map.setCenter(marker.getPosition());
    infoWindow.open(this.map,marker);
  });

  this.markers.push(marker);
```

Ilustración 53. Función para crear marcador y infowindow

```
myIcon = {
  url: 'assets/icon/Marker_Pacer_maps.png',
  scaledSize: new google.maps.Size(50,50)
};
userIcon = {
  url: 'assets/icon/Marker_runner_maps.png',
  scaledSize: new google.maps.Size(50,50)
};
gostIcon = {
  url: 'assets/icon/Marker_ghost_maps.png',
  scaledSize: new google.maps.Size(50,50)
};
```

Ilustración 54. Distinción entre iconos según diferentes estados

```
let userContent =
  '<ion-grid>'+
    '<ion-row>'+
      '<ion-col>'+
      //  '<img [src]="'+ userData.data()['photoURL'] +'" width="100px" height="120px" tappable/>'+
        '<img src="assets/icon/User.png" width="100px" height="120px" tappable/>'+
      '</ion-col>'+
      '<ion-col>'+
        '<h1 class="center_vertical">'+ userData.data()['displayName'] +'</h1>'+
        '<h3>'+ this.calculateAge(userData.data()['birthday']) +' años</h3>'+
        '<h3>Distancia:'+ distance +' km</h3>'+
      '</ion-col>'+
    '</ion-row>'+
    '<ion-row justify-content-center align-items-center>'+
      '<ion-button id="chatBtn" color="pacergreen" shape="round" style="width:230px">Chat</ion-button>'+
    '</ion-row>'+
  '</ion-grid>';
```

Ilustración 55. Contenido de infowindow para otros usuarios

```
this.myContent =
'<ion-grid>'+
  '<ion-row>'+
    '<ion-col>'+
    //  '<img [src]="'+ userData.data()['photoURL'] +'" width="100px" height="120px" tappable/>'+
      '<img src="assets/icon/User.png" width="100px" height="120px" tappable/>'+
    '</ion-col>'+
    '<ion-col>'+
      '<h1 class="center_vertical">'+ userData.data()['displayName'] +'</h1>'+
      '<h3>'+ this.calculateAge(userData.data()['birthday']) +' años</h3>'+
    '</ion-col>'+
  '</ion-row>'+
'</ion-grid>';
```

Ilustración 56. Contenidos de infowindow para propio usuario

```
export interface Location {
  uid: string;
  lat: number;
  lng: number;
  isTracking: boolean
}
```

Ilustración 57. Interfaz exportado "Location"

```
//READ
  public locations = this.locationsCollection.snapshotChanges().pipe(
    map(actions => {
      return actions.map(p => {
        const data = p.payload.doc.data() as Location;
        data.uid = p.payload.doc.id;
        return data;
      });
    })
  );
```

Ilustración 58. Función de lectura de datos

```
//UPDATE
  async updateLocation(uid: string, lat:number, lng:number, isTracking:boolean): Promise<void> {
    try {
      await this.locationsCollection
        .doc(uid)
        .set({ lat: lat, lng:lng, uid:uid, isTracking:isTracking }, { merge: true });
    } catch (err) {
      console.log(err);
    }
  }
```

Ilustración 59. Función para actualizar datos

```
  let watch = this.geolocation.watchPosition();
  this.subscription = watch.subscribe((data) => {
```

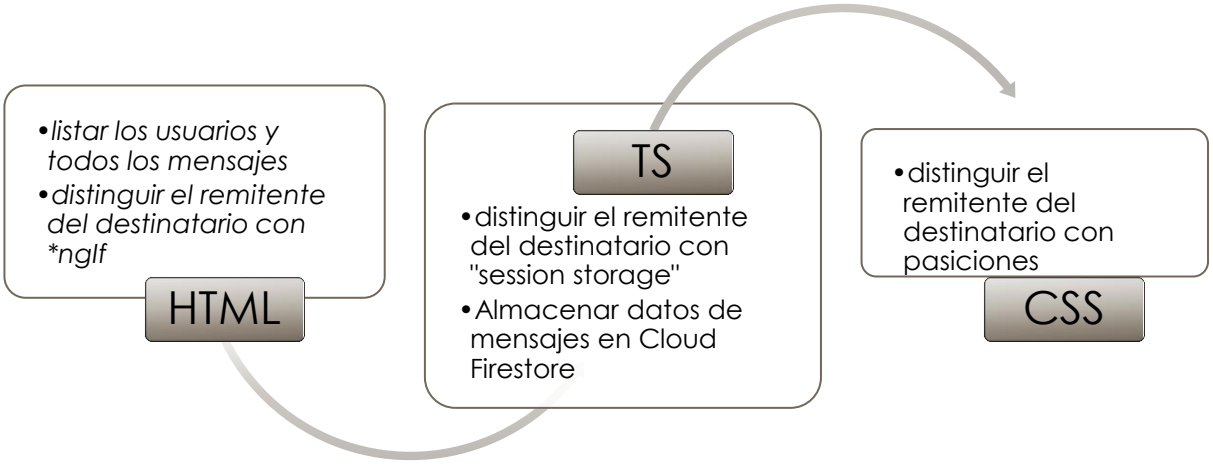
Ilustración 60. Función para vigilar movimientos

```
this.locationSvc.locations.subscribe(data =>{
  this.deleteMarkers();
  data.forEach(loc =>{
    if(loc.uid != this.currentUser.uid){
      console.log("This is other user",loc.uid);
      if(!loc.isTracking){
        console.log("Status is not tracking");
      }else{
        firebase.firestore().collection("users").doc(loc.uid).get().then(userData =>{
```

```
else{
  console.log("This is current user",loc.uid);
  if(!loc.isTracking){
    console.log("Status is not tracking");
    let updatelocation = new google.maps.LatLng(loc.lat,loc.lng);
    this.addMarker(updatelocation,this.gostIcon,"<b>Estás invisible ahora.</b>",null,null);
    this.setMapOnAll(this.map);
  }else{
    firebase.firestore().collection("users").doc(loc.uid).get().then(userData =>{
```

Ilustración 61, Ilustración 62. Función para distinguir identidad y estado de usuario

Anexo 4. Implementación de chat en tiempo real




```
this.name = sessionStorage.getItem("name");
this.o_uid = sessionStorage.getItem("uid");
this.uid = JSON.parse(localStorage.getItem("user")).uid;
```

Ilustración 63. Local storage and Session Storage

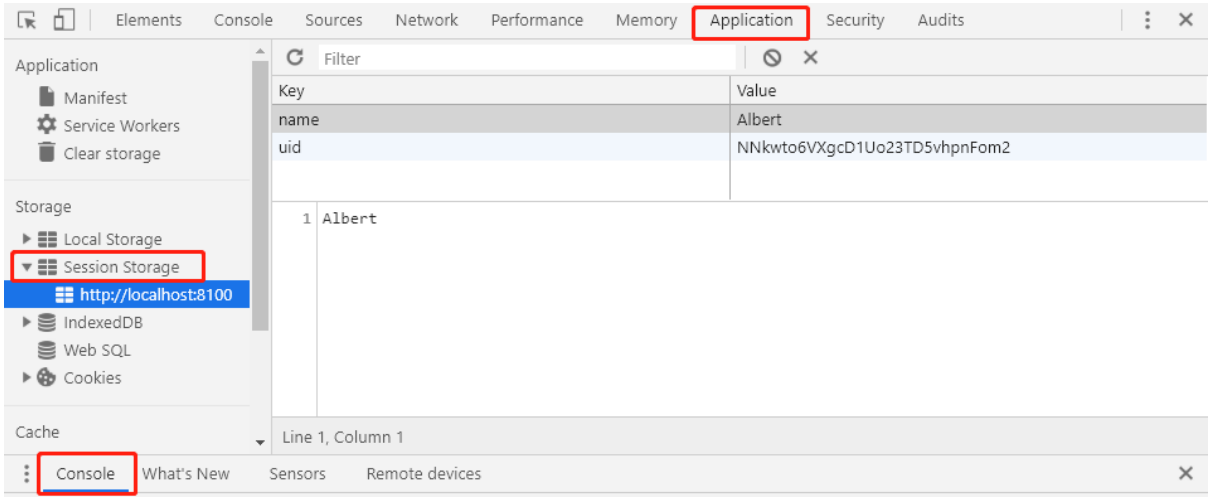


Ilustración 64. Interfaz de Session Storage

```
//my chats collection
firebase.firestore().collection("chats").doc(this.uid).collection(this.o_uid).doc(Date.now().toString()).set({
    time: Date.now(),
    uid:this.uid,
    msg:this.textMsg
});

//other's chats collection
firebase.firestore().collection("chats").doc(this.o_uid).collection(this.uid).doc(Date.now().toString()).set({
    time: Date.now(),
    uid:this.uid,
    msg:this.textMsg
}).then(() =>{
  this.textMsg = "";
});
```

Ilustración 65. Identificación de dirección de envío

```
<ion-row *ngFor="let chat of chats">
  <ion-col size="9" lines="none" *ngIf="chat.uid != uid" class="message other-message">
  <ion-col offset="3" size="9" lines="none" *ngIf="chat.uid == uid" class="message my-message">
```

Ilustración 66, Ilustración 67. Distinción visualmente entre remitente y destinatario

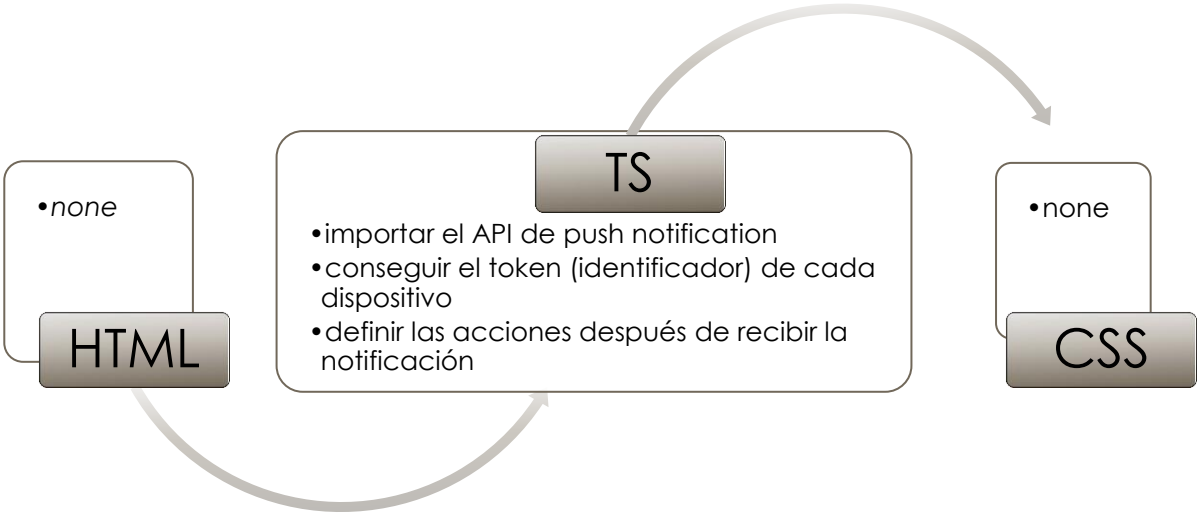
```
//my chats collection
firebase.firestore().collection("chats").doc(this.uid).collection(this.o_uid).doc(Date.now().toString()).set({
  time: Date.now(),
  uid:this.uid,
  msg:this.textMsg
});

//other's chats collection
firebase.firestore().collection("chats").doc(this.o_uid).collection(this.uid).doc(Date.now().toString()).set({
  time: Date.now(),
  uid:this.uid,
  msg:this.textMsg
}).then(() =>{
  this.textMsg = "";
});
```

Ilustración 68. Identificación de mensajes por tiempo

```
firebase.firestore().collection("chats").doc(this.uid).collection(this.o_uid).orderBy("time").onSnapshot(snap =>{
  this.chats = [];
  snap.forEach(child =>{
    this.chats.push(child.data());
  })
});
}
```

Ilustración 69. Mensajes ordenados según tiempo

Anexo 6. Implementación de Push Notification




```
import {
  Plugins,
  PushNotification,
  PushNotificationToken,
  PushNotificationActionPerformed,
  Capacitor
} from '@capacitor/core';
import { Router } from '@angular/router';

const { PushNotifications, Modals } = Plugins;
```

Ilustración 70. Paquetes de Notificación

```
PushNotifications.requestPermission().then((permission) => {
    if (permission.granted) {
      // Register with Apple / Google to receive push via APNS/FCM
      PushNotifications.register();
```

Ilustración 71. Registro de push-notification

```
PushNotifications.addListener('registration', (token: PushNotificationToken) => {
    console.log('My token: ' + JSON.stringify(token));
    firebase.firestore().collection("tokens").doc(JSON.stringify(token.value)).set({
      token: JSON.stringify(token.value)
    }, {merge:true});
  }
);
```

Ilustración 72. Get token

```
PushNotifications.addListener(
    'pushNotificationReceived',
    async (notification: PushNotification) => {
```

Ilustración 73. Acción tras recibir notificación

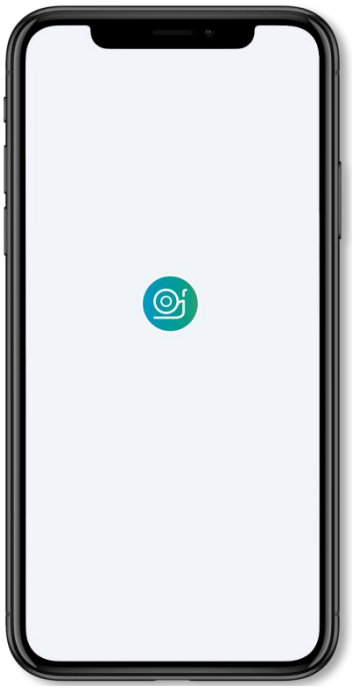
```
PushNotifications.addListener(
    'pushNotificationActionPerformed',
    async (notification: PushNotificationActionPerformed) => {
```

Ilustración 74. Acción tras presionar sobre notificación

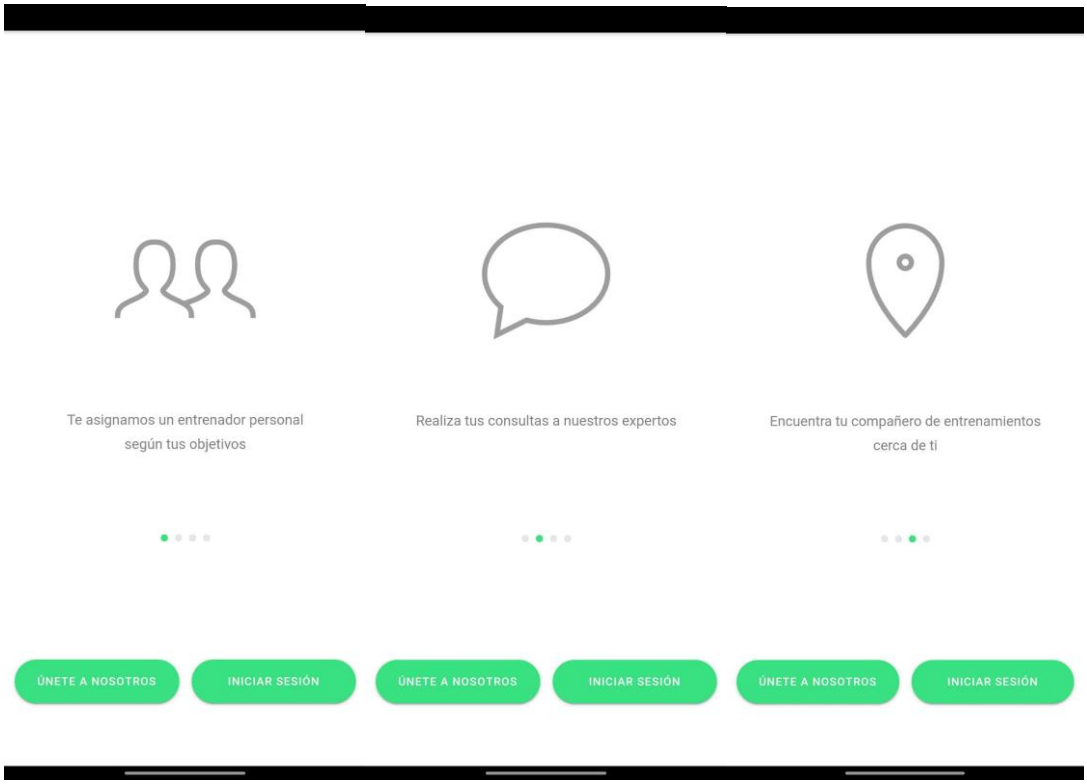
Ilustración 75. Configuración de notificación

Anexo 7. Interfaces realizadas

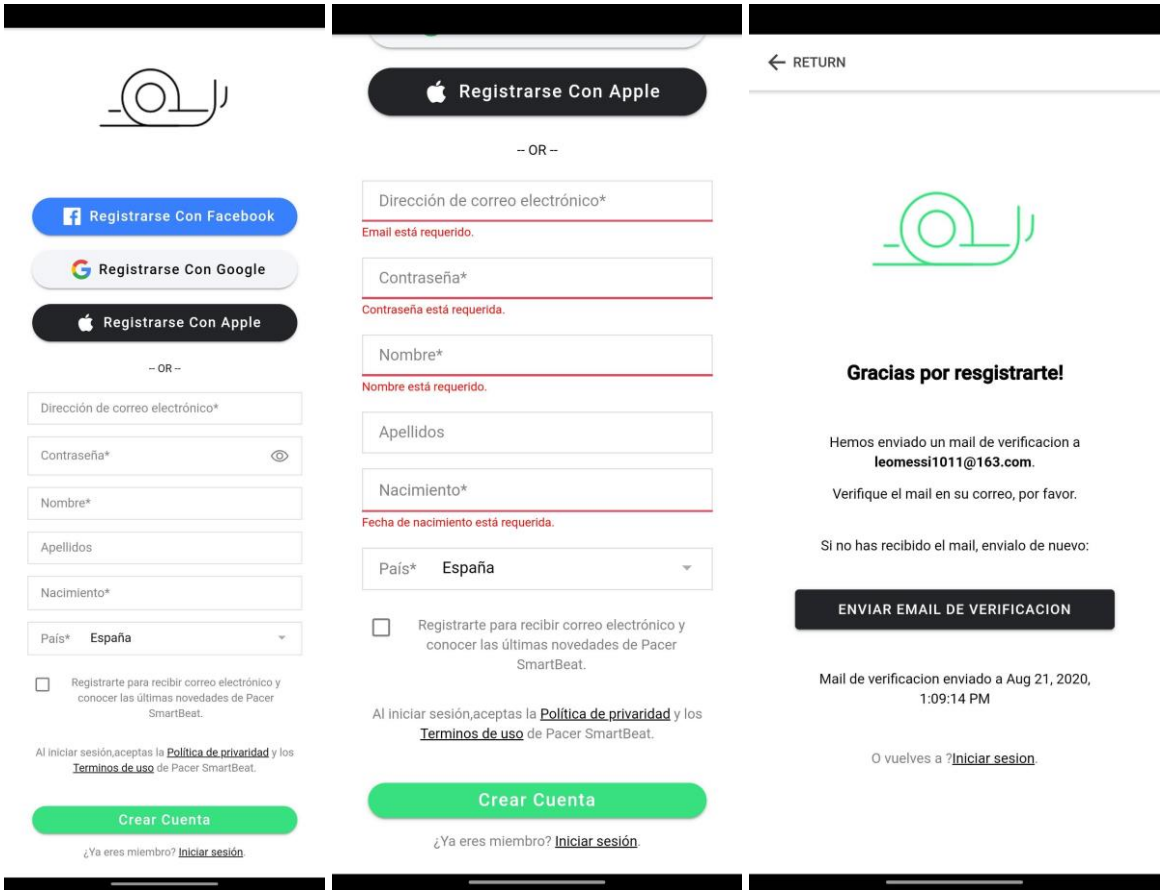
- Icono y portal de aplicación



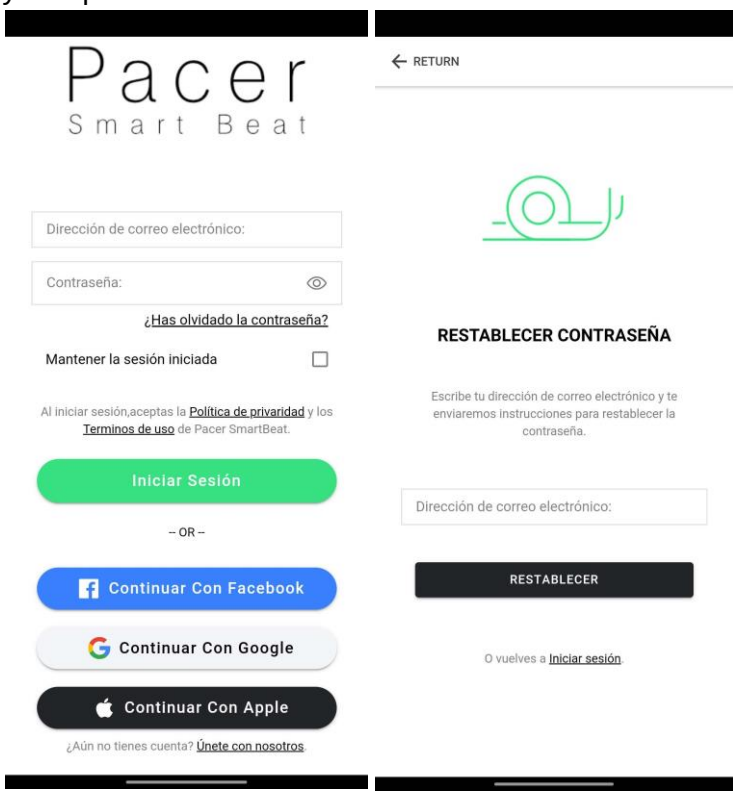
- Bienvenida para nuevos usuarios



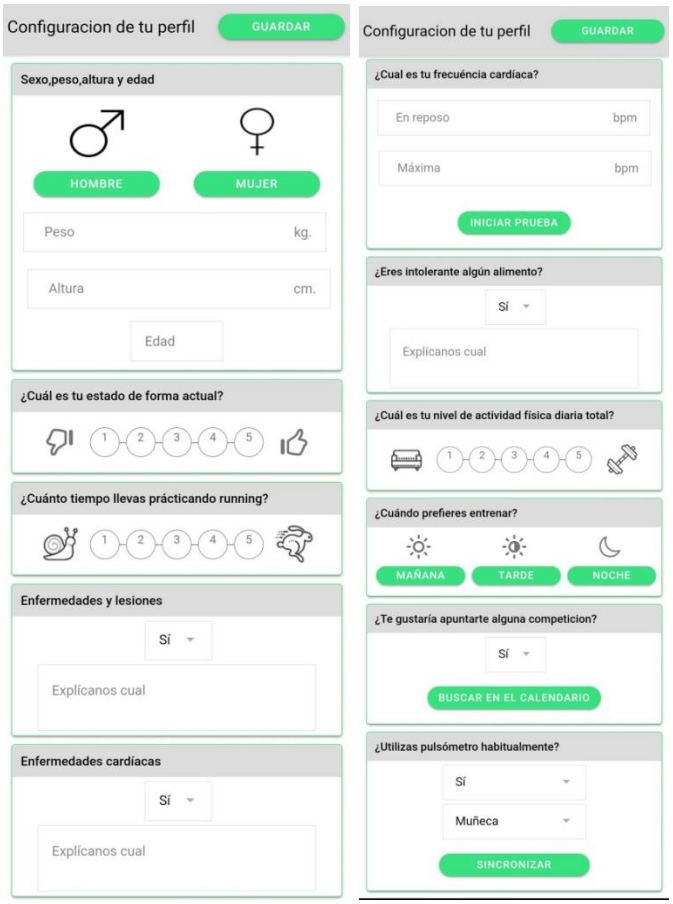
- Registración y verificación de correo



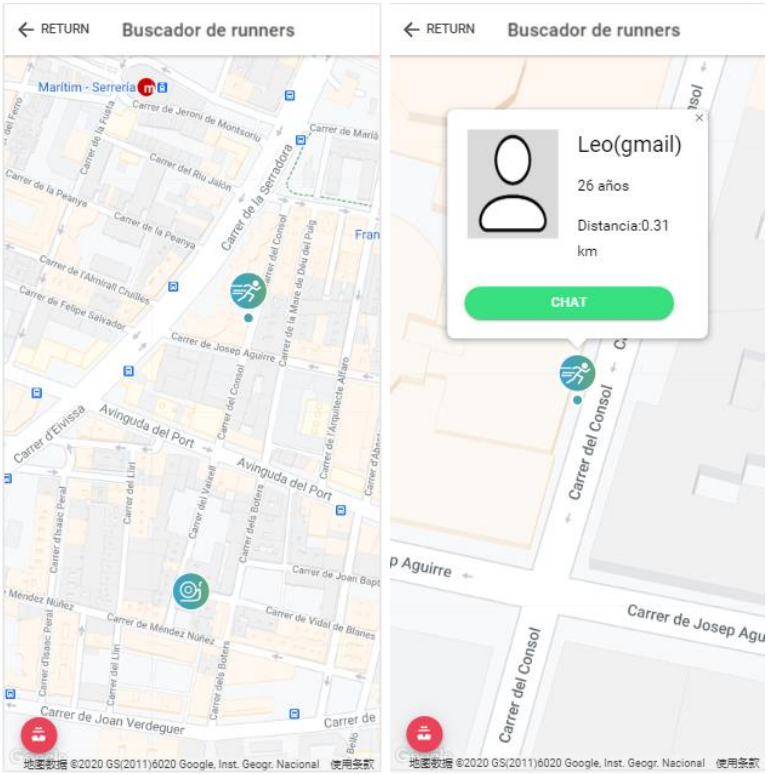
- Iniciar sesión y recuperación de contraseña



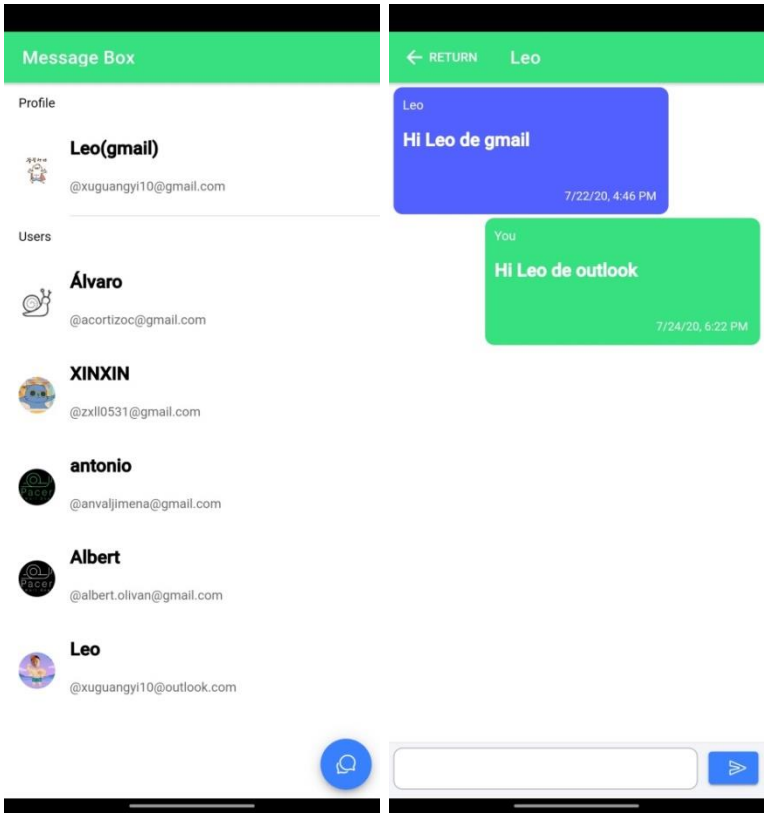
- Perfil de usuario



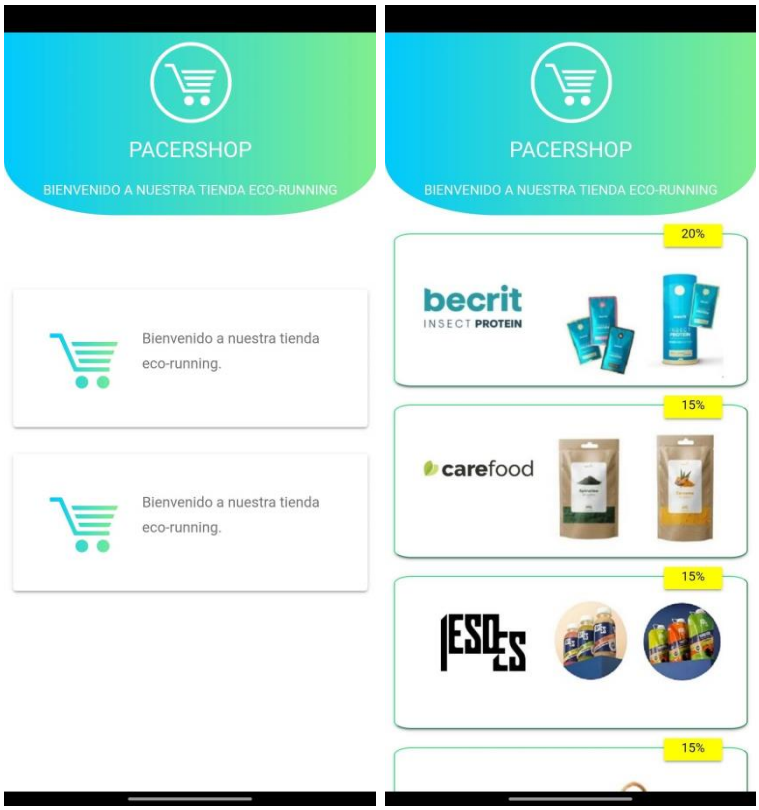
- Buscador de runners



- Habitación de *chat*



- Marketplace



- *Push notification*