UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf

Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Diseño e implementación de una aplicación móvil para ayuda al entrenamiento deportivo

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática

Autor: Zarzycki Szymanski, Kevin Patryk

Tutor: Sáez Barona, Sergio

Curso 2017-2018

Resumen

En los últimos años el número de personas que han comenzado a aficionarse a correr ha aumentado de forma significativa, se puede observar como Valencia se ha convertido en una ciudad que alberga una gran cantidad de eventos y algunos de ellos tienen un gran reconocimiento mundial. La tecnología ha tenido un gran impacto en este aumento ya que gracias a dispositivos electrónicos, como los relojes con GPS que te muestran información detallada de tus entrenamientos, los corredores pueden tener un historial muy detallado y pueden observar su progresión.

En este trabajo se pretende ofrecer a todos esos corredores una aplicación para móviles Android donde puedan introducir todos los entrenamientos que vayan realizando, establecerse objetivos que quieran cumplir y poder comparar sus entrenamientos con otros corredores que utilicen la aplicación. Para proporcionar la conectividad entre todos los usuarios se realizará un servidor alojado en internet donde están almacenados todos los datos.

Durante la realización de este proyecto se enseñarán y usarán distintos métodos los cuales pueden ser usados por cualquier desarrollador para la realización de un proyecto software de calidad.

Palabras clave: Android, Backend, Rails, Kotlin, MVC, MVP

Abstract

In the last years the number of people who have started to take to running has increased in a significant way, it can be observed how Valencia has become a city that hosts a lot of events and some of them have a great worldwide recognition. The technology has had a great impact on this increase because thanks to electronic devices, such as GPS clocks that show you detailed information of your trainings, runners can have a very detailed history and they can observe its progression.

This work is intended to offer all those runners an Android mobile application where they can introduce all the training they are doing, establish goals they want to achieve and be able to compare their trainings with other runners who use the application. To provide connectivity among all users, a server hosted on the internet will be made where all the data is stored.

During the realization of this project will be taught and will use different methods which can be used by any developer to carry out a quality software project.

Key words: Android, Backend, Rails, Kotlin, MVC, MVP

Índice general

Índice de figuras

CAPÍTULO 1
Introducción

Actualmente el uso de la tecnología ofrece un grandes beneficios para las personas como puede ser la capacidad de tener un registro de todas aquellas cosas que realizamos, pero gracias a ella y al uso de internet se puede ampliar esta capacidad y ofrecer muchas más mejoras a las personas.

En el mundo del deporte la forma en la que se entrena es la pieza clave a la hora de conseguir objetivos, una mala planificación en la mayoría de casos resultará en el fracaso en estos objetivos. El uso de la tecnología puede jugar un gran papel para ayudar a cualquier persona a conseguir los objetivos que quiera realizar.

Dadas las razones que se han mencionado durante todo el documento se va a describir el desarrollo de una plataforma en la cual cualquier corredor pueda tener un historial de los entrenamientos realizados, tener una lista de objetivos a realizar (tanto objetivos cumplidos como objetivos futuros a realizar) y poder buscar a otros usuarios con sus mismas características para poder ver como entrenan y compararse con ellos.

Con esta herramienta el usuario será capaz de poder hacer una análisis de sus entrenamientos a la vez que observa como entrenan otras personas que tengan sus mismos objetivos (o entrenen de forma similar) y así puedan mejorar en sus entrenamientos de cara a esos objetivos que tengan planteados.

Esta herramienta estará principalmente centrada a los corredores pero podría ser ampliarse con relativa facilidad para poder ser utilizada en otros deportes.

1.1 Motivación

La temática de este TFG está rodeada del tema del deporte, un tema el cual lleva presente en mi vida desde que era pequeño. He practicado varios deportes durante mi vida (fútbol, natación, correr, triatlón), algunos de ellos por diversión y otros de forma más seria compitiendo.

Existen personas las cuales inician un deporte por mera diversión pero también existen muchas otras que aparte se establecen unos objetivos, los cuales muchas veces pueden llegar a ser duros y para los cuales se necesita una buena preparación. En muchos de los casos la figura de un entrenador es muy importante para conseguir esos objetivos, sin el muchas de las personas al carecer de conocimientos sobre el deporte realizan una mala preparación para sus objetivos o no llegan a conseguir todo el rendimiento que podrían ser capaces de obtener. Muchas personas no son capaces de afrontar el coste que tiene tener a un entrenador por lo tanto herramientas como la que se presenta en este trabajo

son una gran ayuda para todas aquellas personas que quieran mejorar y conseguir sus objetivos.

1.2 Objetivos

El desarrollo de una aplicación móvil completa tiene un alto coste de tiempo, por ello durante este trabajo se va a presentar una aplicación la cual ya tenga funcionalidades básicas para conseguir el objetivo deseado. Todas las funcionalidades que estarán presentes en la aplicación se pueden ver descritas de una forma más amplia dentro del apartado 4.

A parte de realizar todo el desarrollo otro de los objetivos es realizar todo el proyecto siguiendo buenas prácticas y arquitecturas software las cuales se irán comentando a lo largo del documento, todo esto se realiza para poder construir un proyecto software de gran calidad.

1.3 Impacto esperado

Con esta aplicación se desea crear una herramienta la cual implique una ayuda para los corredores y la creación de una gran red de corredores la cual pueda proporcionar gran información a aquel que la necesite.

En esta aplicación se podría definir dos tipos de usuarios. Primeramente tendríamos los usuarios que podrían definirse como «creadores», son esos usuarios que estarían más interesados en la aplicación desde el punto de vista de tener un sitio donde poder tener un historial de todos los entrenamientos que han realizado y realizar un análisis de su progreso en base a ellos. Por otro lado estarían los usuarios llamados «consumidores» los cuales están bastante interesados en aquello realizado por otras personas, ya sea para mejorar sus entrenamientos o por la simple curiosidad de que hacen. Aunque se hayan definido como un tipo de usuarios bastante diferente habitualmente los usuarios suelen ser una mezcla de estos dos tipos.

1.4 Estudio de mercado

Hoy en día el avance de las tecnologías han permitido que un gran porcentaje de la población tenga a su alcance un dispositivo móvil con el cual estar conectado a internet. Este gran avance ha hecho que las personas prefieran utilizar su móvil como herramienta principal dejando de lado otros dispositivos como los ordenadores en segundo lugar.

Como podemos observar en la gráfica de la figura 1.1 el uso del internet en los dispositivos móviles superó al ordenador hace unos pocos años en Estados Unidos, tendencia que como se puede observar va en aumento. Dado que el objetivo es llegar al máximo número de usuarios se ha decidido desarrollar la herramienta para móvil. La mejor solución seria proporcionar un desarrollo tanto para plataformas móviles y como para web pero dado al alto coste temporal que tiene el desarrollo de ambas plataformas se ha tenido que elegir una de ellas.

Dentro del ámbito móvil la solución se desarrollará para dispositivos Android, ya que desde hace unos años implican el mayor porcentaje de dispositivos en el mercado (como se puede observar en la gráfica de la figura 1.2). En esta decisión también ha entrado en juego el factor temporal, ya que un desarrollo para dos plataformas distintas implica un

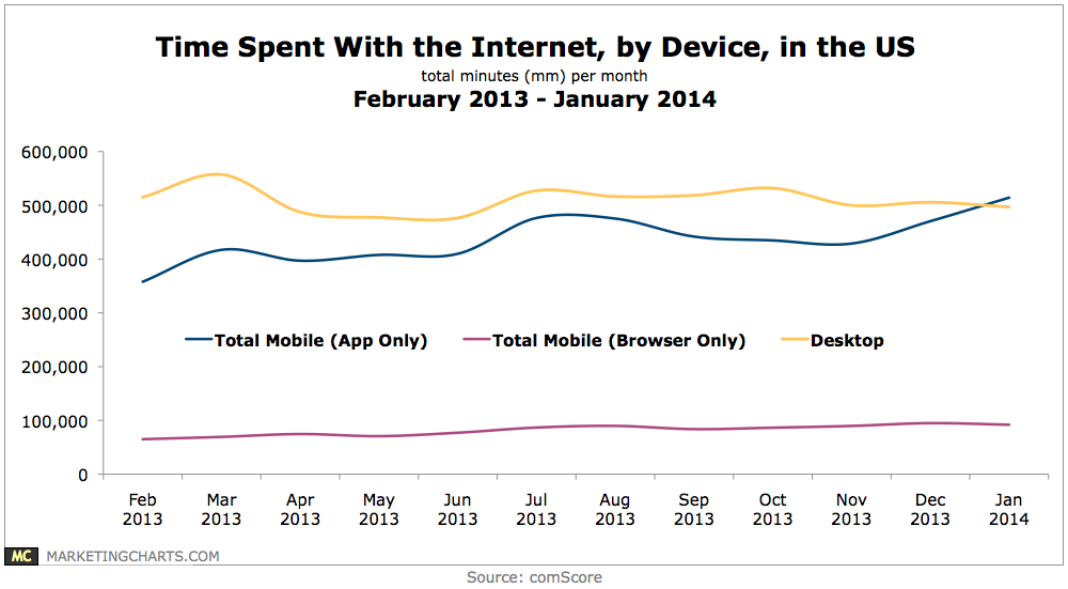

Figura 1.1: Tiempo gastado en internet por dispositivo en USA

Fuente: https://www.marketingcharts.com/industries/travel-and-hospitality-41153

alto coste temporal y también un alto coste de aprendizaje ya que para cada plataforma el desarrollo se realiza de forma distinta.

1.5 Estructura de la memória

Esta memoria está estructurada en cuatro bloques. El primer bloque hace referencia a la recopilación de todos los diseños y especificaciones de las funcionalidades a desarrollar. El segundo y tercer bloque hacen referencia al desarrollo del servidor y la aplicación móvil respectivamente, en cada uno de estos bloque se comentarán cosas como las tecnologías utilizadas, estructura del proyecto, tecnologías utilizadas, descripción del desarrollo de una funcionalidad, etc. Por último el cuarto bloque se trata de una muestra del resultado de la aplicación una vez terminado el desarrollo.

1.6 Metodología

Para el desarrollo del proyecto se seguirá una estructura de metodología similar al **desarrollo ágil**.

El desarrollo ágil se trata de un proceso iterativo e incremental, este tipo de desarrollo se caracteriza en una gran adaptación a los cambios de requisitos (ya que no se especifica todo el producto desde un principio), en rápidas entregas de versiones y en una gran participación del cliente. Este tipo de desarrollo surgió al comienzo del siglo XXI donde un conjunto de desarrolladores quería proporcionar una solución mejor a todas aquellas metodologías donde el desarrollo se realiza de principio a fin sin aceptar ningún cambio en la especificación [3].

Cada entrega de producto está compuesto por todas las etapas de desarrollo: análisis, diseño, desarrollo y pruebas [2]. Para la descripción de cada una de las funcionalidades se utilizarán las historias de usuario tal y como se utilizan en este tipo de metodología, estas historias de usuario se introducirán en la sección 4.

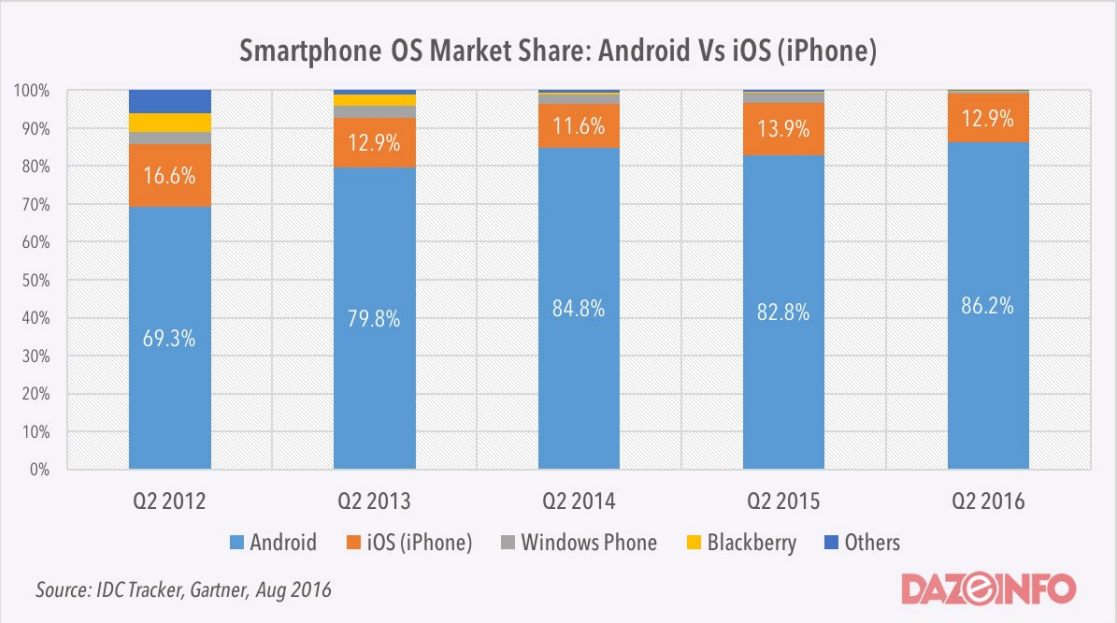

Figura 1.2: Cuota de mercado de los sistemas operativos móviles

Fuente: https://android.jlelse.eu/apple-vs-android-a-comparative-study-2017-c5799a0a1683

Cada una de las entregas se realiza en los llamados **sprints** que se componen de las etapas mencionadas anteriormente, normalmente la duración es de 2 semanas pero puede cambiar según el requerimiento de cada proyecto.

Para la realización de este proyecto se desarrollará cada una de las funcionalidades en un sprint distinto. Cada sprint estará compuesto por todo el desarrollo necesario para que la funcione, es decir, todo lo necesario tanto en el servidor como en la aplicación móvil. Es cierto que este tipo de metodologías están pensadas más a la hora de trabajar en equipo pero se ha querido adoptar muchos de los conceptos y buenas prácticas que ofrecen.

1.7 Clean Architecture

Cuando se está realizando un trabajo de software siempre es necesario intentar desarrollar un proyecto de calidad. A la hora de hablar sobre la arquitectura de un proyecto el término más conocido es el de **Clean Architecture**, el cual se basa en un grupo de prácticas las cuales hacen que nuestros proyectos sean:

- Independiente de cualquier framework.
- Testeable
- Independiente de la interfaz.
- Independiente de la base de datos.
- Independiente de cualquier agente externo.

En la figura 1.4 podemos ver un esquema de la forma que tiene una «arquitectura limpia», esta arquitectura no tiene que ser completamente igual a la que se ve en ella, la cosa más importante al optar por esta forma de arquitectura es la regla de dependencia

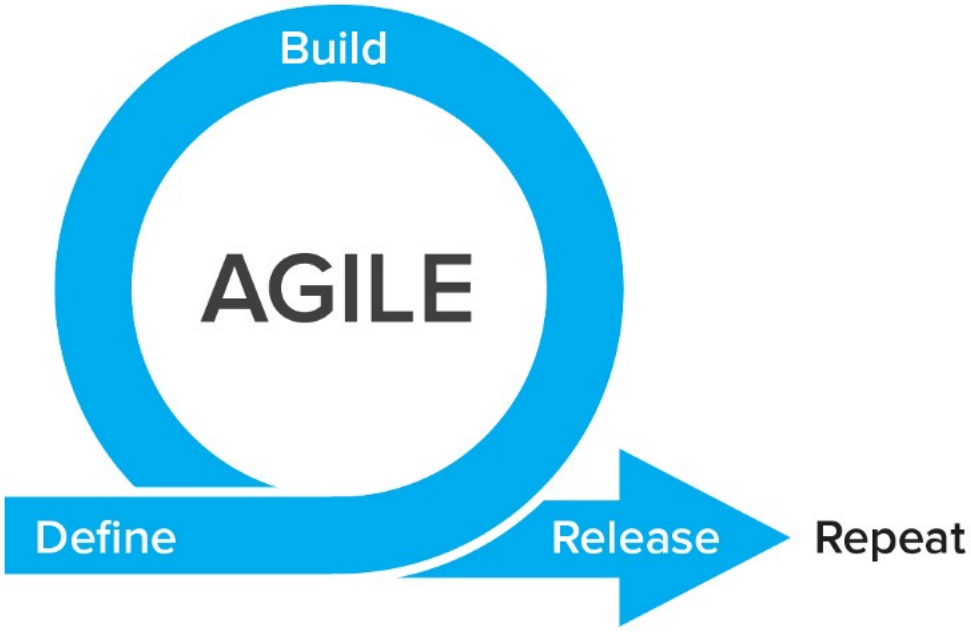

Figura 1.3: Estructura del desarrollo ágil

Fuente: https://medium.com/@LazaroIbanez/a-quick-overview-to-agile-5c87ffc9e0f2

(**Dependency Rule**) la cual dice que todas las dependencias solo pueden producirse hacia círculos interiores y nunca hacia fuera, es decir, un circulo de dentro tiene completo desconocimiento de lo que pasa en los círculos exteriores [1].

Durante este proyecto a la hora de realizar tanto el servidor como la aplicación móvil se podrá ver la aplicación de esta idea con dos tipos de arquitectura distintas.

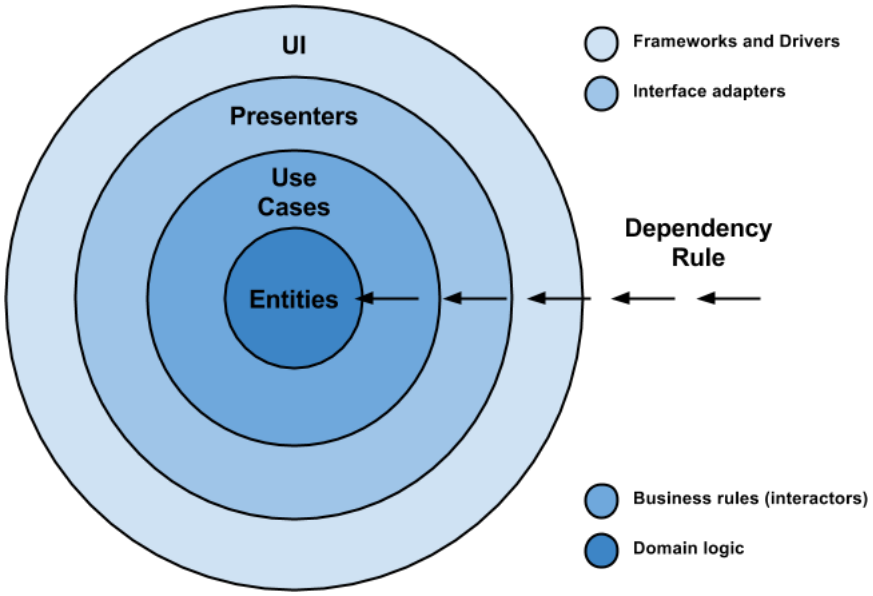

Figura 1.4: Esquema Clean Architecture

Fuente: https://fernandocejas.com/2014/09/03/architecting-android-the-clean-way/

CAPÍTULO 2
Estado del arte

Actualmente dado al gran uso que se le da a internet por parte de todo tipos de usuarios es muy difícil no encontrar alguna aplicación o herramienta de cualquier cosa que busquemos. En el caso de aplicaciones deportivas podemos encontrar también una gran cantidad de aplicaciones de todo tipo.

Existen muchas aplicaciones que permiten a los usuarios llevar un seguimiento de todos sus entrenamientos, a continuación se van a mencionar algunas de las aplicaciones más utilizadas.

2.1 Strava

Strava [1] es utilizada por millones de usuarios por todo el mundo, usuarios de todos los tipos, desde usuarios amateurs hasta atletas profesionales. Se trata de una red social la cual permite a usuarios de muchas disciplinas distintas compartir sus actividades con todo el mundo.

Permite un gran análisis de nuestras actividades con estadísticas de la velocidad, ritmo cardiaco, altura... También ofrece otros análisis como un historial de los tiempos que vamos realizando en una misma ruta.

Como hemos comentado Strava se centra mucho en el enfoque de la plataforma como red social, haciendo que haya una gran interacción de los usuarios. Una de las funcionalidades más destacadas para ello son los **segmentos** donde los usuarios compiten por ver quien hace un segmento de ruta en el menor tiempo posible.

Posiblemente Strava sea la aplicación más actualizada en la actualidad, aunque su uso es mucho más frecuente en gente la cual práctica deporte a menudo o incluso por muchos atletas profesionales como hemos comentado anteriormente. Uno de los motivos por los cuales sea la más utilizada es que gracias a ella se cubren muchos tipos de deportes a diferencia de otras aplicaciones como Runtastic las cuales tienen una aplicación para cada tipo de deporte.

También ofrece herramientas para aquellos que quieran explorar, permite crear y compartir rutas con toda la comunidad de usuarios para que así a los usuarios puedan tener una gran cantidad de lugares a explorar allá donde estén.

[1] Strava URL: https://www.strava.com

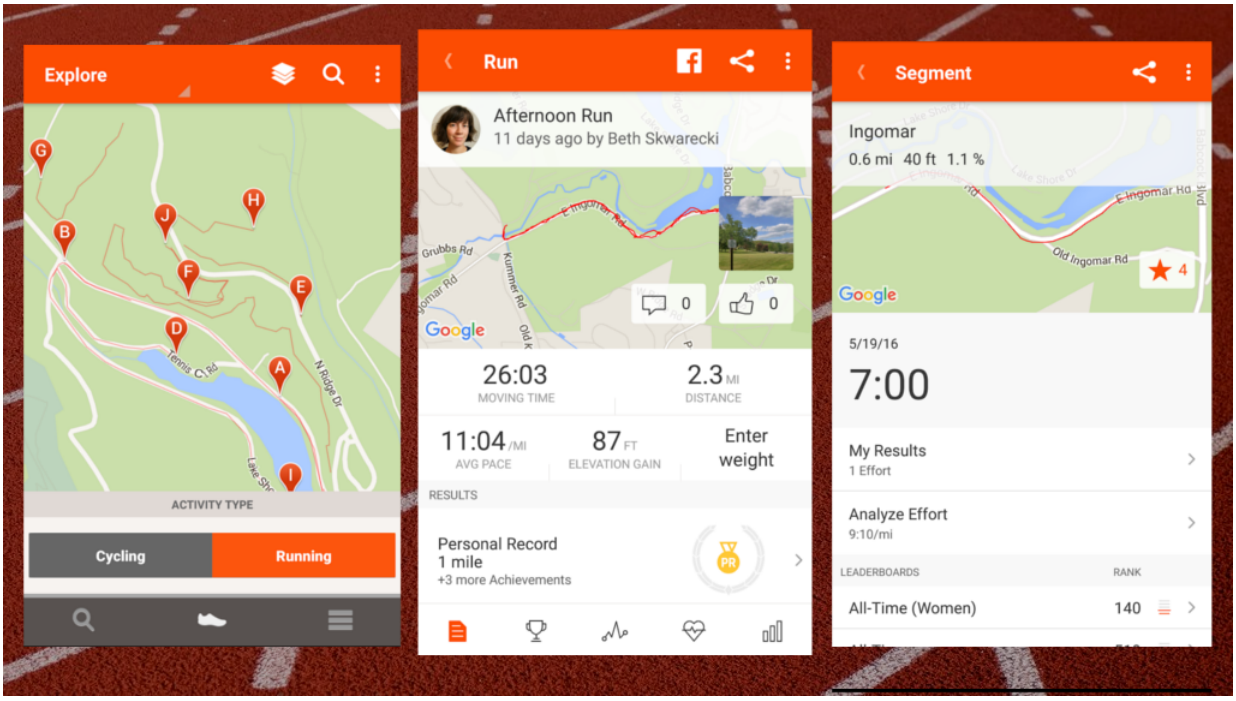

Figura 2.1: Ejemplo segmentos de Strava

Fuente: https://www.lifehacker.com.au/2016/06/whats-the-difference-between-all-these-running-apps/

2.2 Runtastic

Runtastic [2], comprada por Adidas recientemente, se trata de una aplicación móvil la cual permite a los usuarios registrar sus actividades físicas y llevar un análisis de ellas. Lo que diferencia a esta plataforma respecto a otras es que en ella se pueden encontrar una gran cantidad de planes de entrenamiento para todo tipo de usuarios (se puede ver un ejemplo en la figura 2.2) y vídeos con diversos consejos deportivos. También nos ofrece la posibilidad de explorar nuevas rutas para nuestros entrenamientos.

Actualmente Runtastic proporciona más de 15 aplicaciones móviles de salud, fitness y seguimiento según las necesidades que tenga cada usuario. En esta lista de aplicaciones podemos encontrar aplicaciones para el seguimiento de pasos, seguimiento de nuestra frecuencia cardiaca, contador de distintos tipos de ejercicios físicos como las dominadas, etc.

A diferencia de Strava donde la mayoría de usuarios tienen un nivel medio o alto a la hora de realizar actividades físicas Runtastic se centra más en usuarios de nivel medio y bajo que quieren entrenarse de una forma más sencilla ofreciendo para muchos de ellos planes de entrenamiento para guiarles en sus objetivos.

2.3 Endomondo

Endomondo [3] es quizá una de las aplicaciones más populares junto a las dos anteriores, esta nos ofrece características muy similares a las anteriores como son el seguimiento de actividades y los planes de entrenamiento personales. Esta aplicación, junto a Runtas-

[2] Runtastic URL: https://www.runtastic.com/es/
[3] Endomondo URL: https://www.endomondo.com/

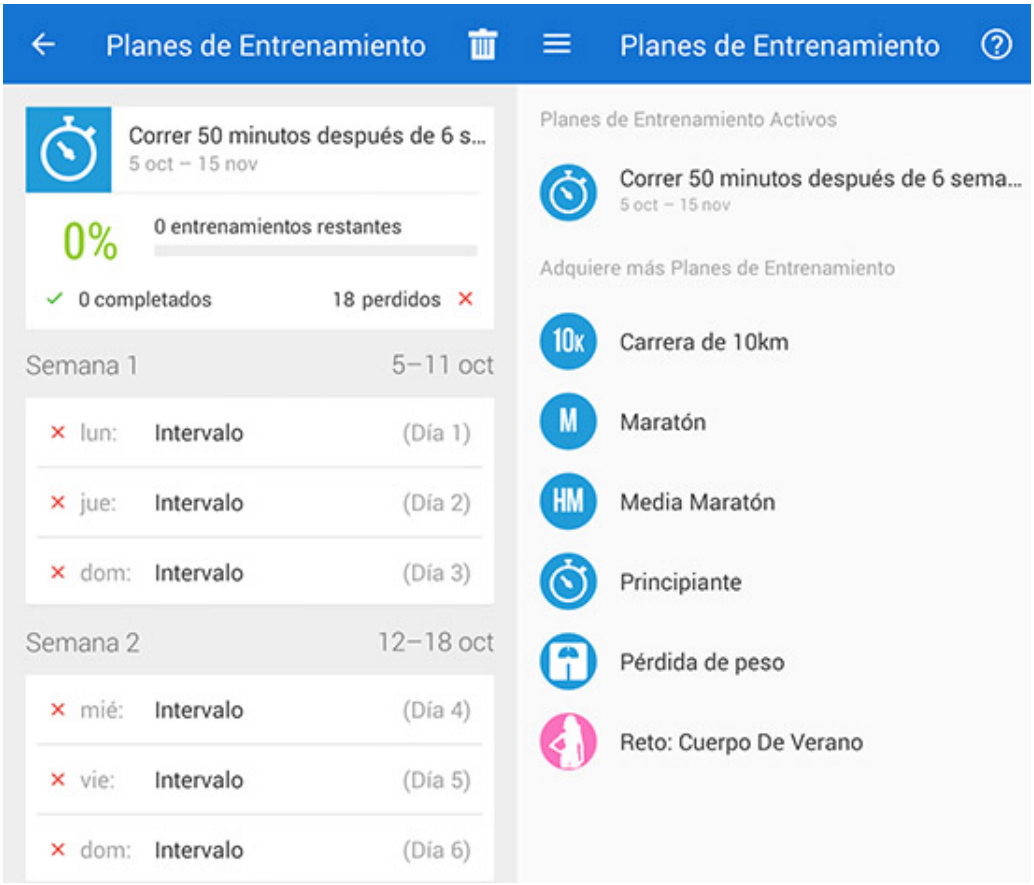

Figura 2.2: Ejemplo plan de entrenamiento Runtastic

Fuente: http://www.werunbarcelona.com/planes-entrenamiento-runtastic-pro/

tic, posiblemente sea la más utilizada por todas aquellas personas que realizan deporte de forma ocasional o de una forma menos profesional.

Pertenece a la gran empresa Under Armour la cual es muy conocida por la venta de ropa y accesorios deportivos, a parte de tener más aplicaciones móviles como MyFitnessPal que se trata de la aplicación de nutrición más utilizada y la cual es recomendada también desde Endomondo.

2.4 Crítica al estado del arte

Las tres aplicaciones presentadas son posiblemente las mejores que se pueden encontrar de este tipo actualmente en el mercado, ofrecen un buen análisis de los entrenamientos, planes de entrenamiento y algunas funcionalidades de gamificación las cuales hacen que los usuarios interaccionen entre ellos (como hemos podido ver con los segmentos de Strava).

Todas son aplicaciones las cuales están dirigidas por grandes empresas, estas empresas siempre buscan obtener beneficio económico y aquí lo podemos comprobar con anuncios o teniendo que pagar para poder acceder a los planes de entrenamiento o herramientas que no se ofrecen de forma gratuita.

Es comprensible que cualquier empresa busque beneficio de alguna forma, pero por otro lado no está bien restringir todo el uso a los usuarios por tener que pagar. Duran-

te este trabajo se intenta iniciar un proyecto el cual estaría encaminado a proporcionar funcionalidades parecidas a estas pero de una forma gratuita a todos los usuarios.

También comentar que en este proyecto se desarrolla una funcionalidad la cual no esta presente en ninguna de las tres aplicaciones. Esta funcionalidad es la de poder buscar a atletas mediante una búsqueda y para poder ver como están entrenando. Las tres aplicaciones podemos observar con facilidad como tienen un componente de red social donde podemos ver como están entrenando nuestros amigos más cercanos, pero no te permite realizar una búsqueda más específica de otra gente a la cual nos puede parecer de gran interés ver como entrenan.

CAPÍTULO 3
Análisis

Durante este capitulo se va a realizar un pequeño análisis a la aplicación realizando una presentación de las entidades necesarias y la relación entre ellas, la persistencia donde se hablará de forma más especifica la forma en la que se guardará la información y por último se mostrarán los casos de uso que podrá realizar el usuario.

3.1 Entidades

En la introducción ya se ha podido ver la idea general de la aplicación y cualquier persona puede hacerse una idea de las entidades las cuales van a existir. Como se ha comentado esta aplicación no va a tener una complejidad alta por lo tanto el número de entidades va a ser reducido. A continuación se van a presentar las entidades que habrá en el proyecto:

- **Usuario**: Guardará la información referente a la persona que utiliza la aplicación.
- **Entrenamientos**: Guardará la información referente a los entrenamientos hechos por cierto usuario.
- **Objetivos**: Guardará la información de los objetivos realizados o por realizar de los usuarios.
- **Estadísticas**: Guardará estadísticas relevantes sobre los entrenamientos realizados por el usuario.

En la figura 3.1 podemos ver un diagrama ER en el cual se muestra la relación entre las entidades, decir que se ha suprimido los atributos de cada entidad de este diagrama dado que se mostrarán a continuación en la siguiente sección. En este diagrama podemos observar lo siguiente:

- Un usuario puede tener de 0 a N entrenamientos.
- Un usuario puede tener de 0 a N objetivos.
- Un usuario siempre va a tener unas estadísticas aunque no haya realizado ningún entrenamiento, en ese caso saldrán con los valores por defecto.

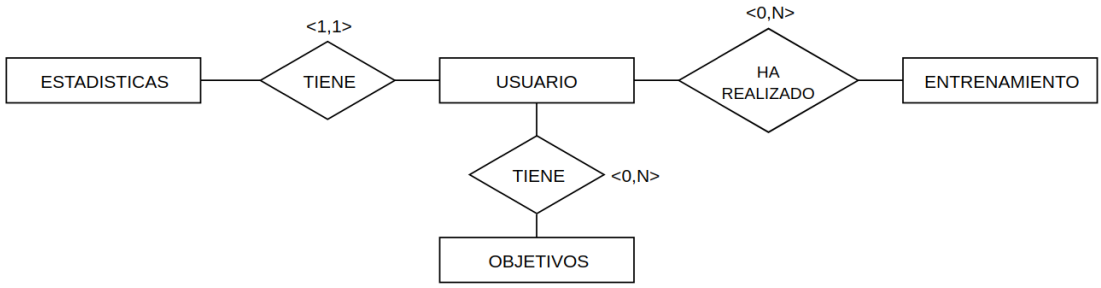

Figura 3.1: Diagrama ER

3.2 Persistencia

Para guardar la toda la información se utilizará una base de datos relacional, esto es debido ya que como hemos podido ver en la sección anterior existe una gran relación entre los dados de la aplicación. Esto no quiere decir que no se pueda utilizar por ejemplo una base de datos no relacional, pero dada la naturaleza de nuestros datos sería complicar el desarrollo innecesariamente.

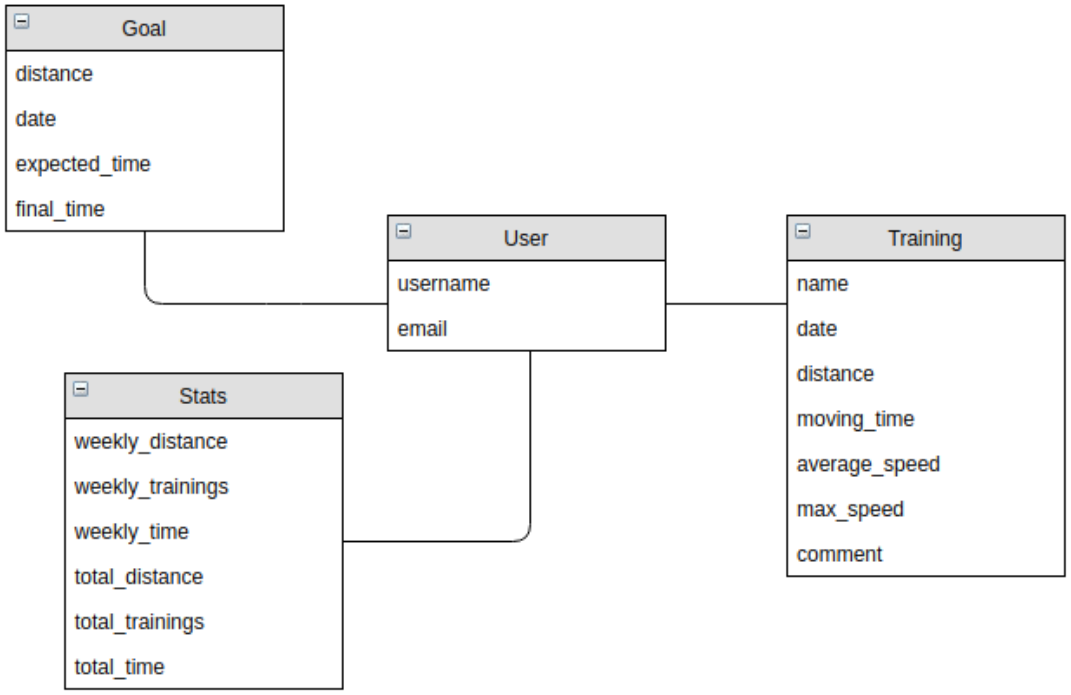

Figura 3.2: Estructura de la base de datos

Podemos observar en la figura 3.2 la estructura que tendrá la base de datos de nuestra aplicación, en ella nos podemos encontrar las siguientes tablas y atributos:

- Tabla de usuario (User):
  - **username**: Nombre de usuario.
  - **email**: Email del usuario.
- Tabla de entrenamiento (Training):

- **name**: Nombre del entrenamiento.
- **date**: Fecha de realización.
- **distance**: Distancia total recorrida.
- **moving_time**: Tiempo total en movimiento.
- **average_speed**: Velocidad media.
- **max_speed**: Velocidad máxima.
- **comment**: Comentario sobre el entrenamiento.

- Tabla de estadísticas (Stats):
  - **weekly_distance**: Distancia total recorrida durante los últimos 7 días.
  - **weekly_trainings**: Número total de entrenamientos durante los últimos 7 días.
  - **weekly_time**: Tiempo total de entrenamiento durante los últimos 7 días.
  - **total_distance**: Distancia total recorrida de todos los entrenamientos.
  - **total_trainings**: Número total de entrenamientos.
  - **total_time**: Tiempo total de entrenamiento.
- Tabla de objetivos (Goal):
  - **distance**: Distancia del objetivo (5km, 10km, 21km, 42km).
  - **date**: Fecha en la que se realizará el objetivo.
  - **expected_time**: Tiempo esperado a realizar.
  - **final_time**: Tiempo final realizado en el objetivo.

En cada una de las entidades existirá un campo `id` el cual identificará a todos los objetos de cada una de las entidades.

3.3 Casos de uso

En la figura 3.3 se pueden observar todos los casos de uso presentes en la aplicación, dado que se trata de una aplicación básica se puede observar como no existe gran complejidad en estos.

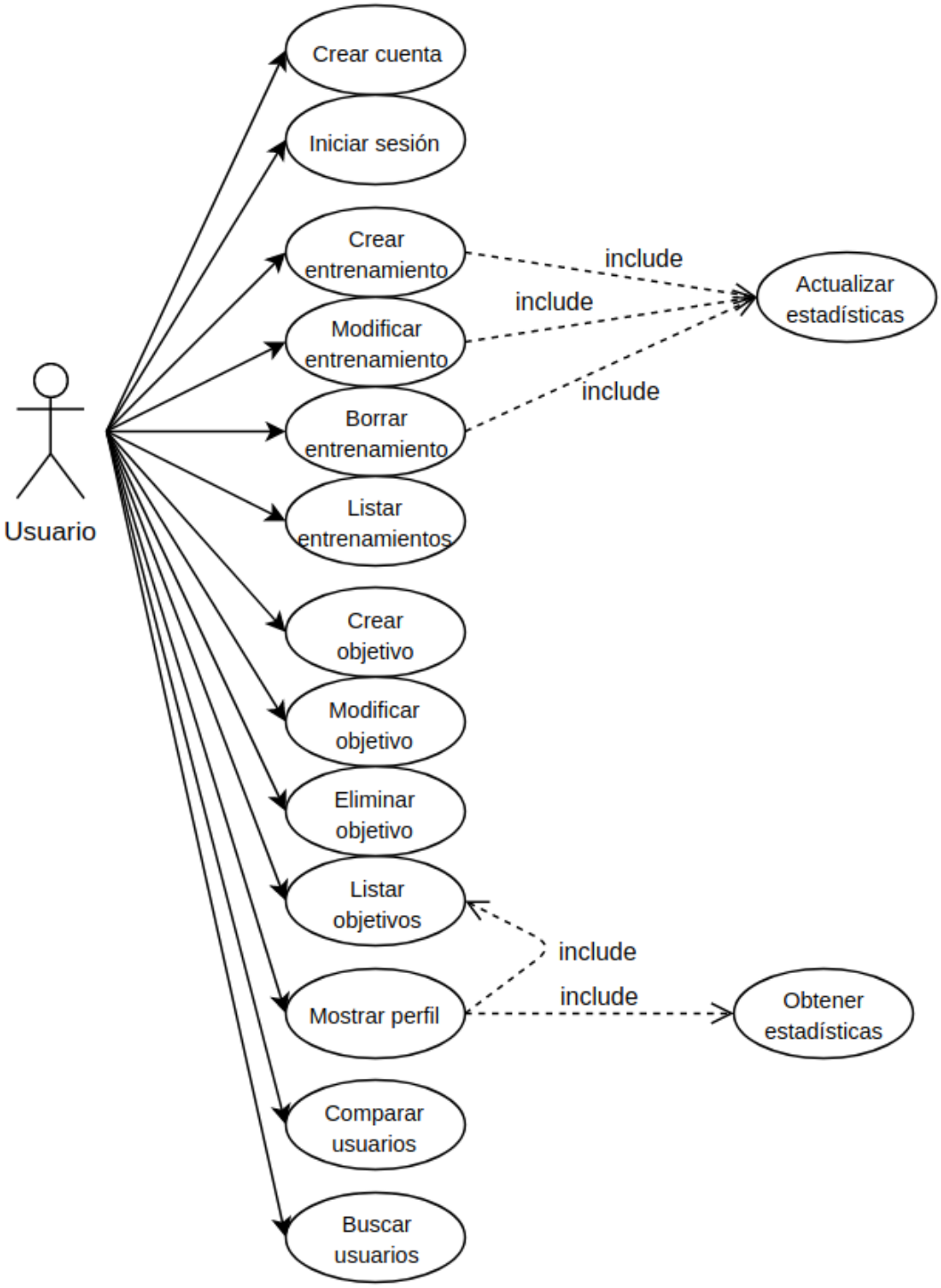

Figura 3.3: Diagrama casos de uso

CAPÍTULO 4
Diseño

En este capitulo se van a presentar las pantallas que van a representar la aplicación móvil, se presentarán mediante **wireframes**, que son diseños sencillos que presentan la estructura que va a tener la pantalla. Estos diseños no son definitivos, se tratan de una aproximación con lo que en la versión final el diseño de la aplicación puede sufrir algunos cambios.

También con cada una de las pantallas se proporcionará los **DoD (Definition of Done)** a nivel de cada funcionalidad. Los DoD son una lista de cosas que tiene que cumplir la funcionalidad desarrollada para considerarse terminada [4]. Con esta lista nos aseguramos que se desarrolla todo aquello que se necesita y que también no se desarrollan cosas innecesarias.

4.1 Inicio de sesión y registro

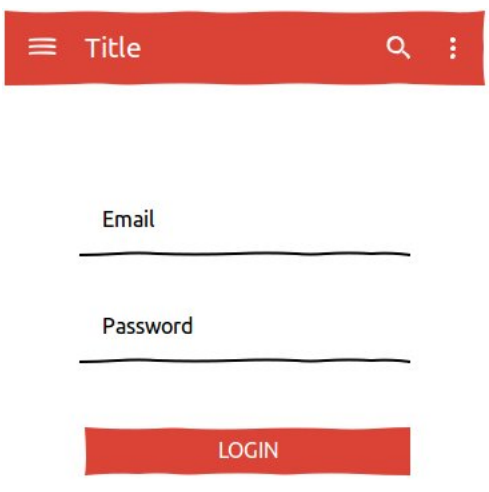

Figura 4.1: Pantalla inicio de sesión

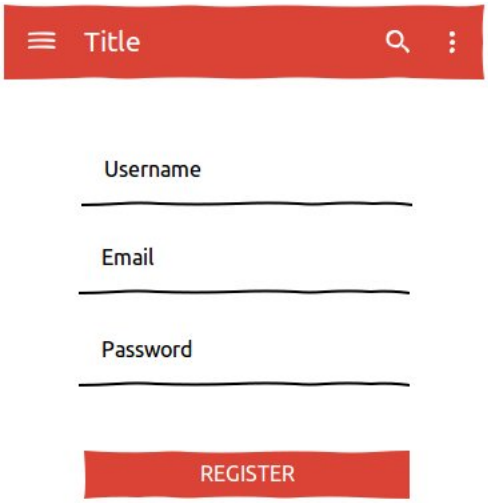

Figura 4.2: Pantalla registro

Definition of done:

- El usuario debe de ser capaz de poder iniciar sesión en su cuenta.
- El usuario debe de ser capaz de poder crearse una cuenta.

4.2 Entrenamientos

En esta funcionalidad al usuario se le proporcionará la habilidad de poder crear entrenamientos o importarlos de otras plataformas:

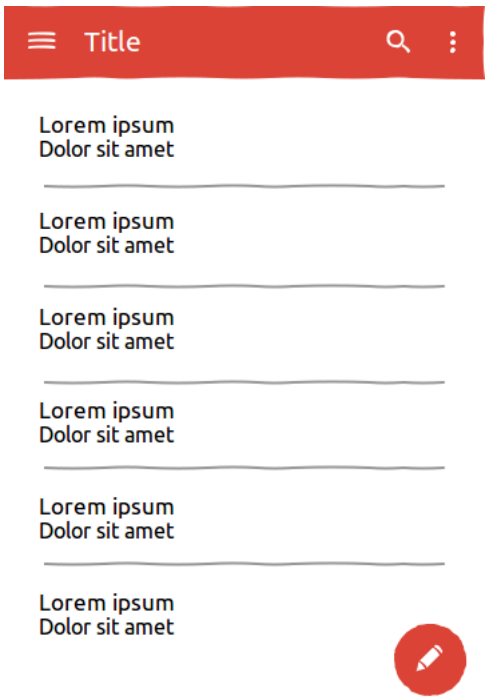

Figura 4.3: Pantalla lista de entrenamientos

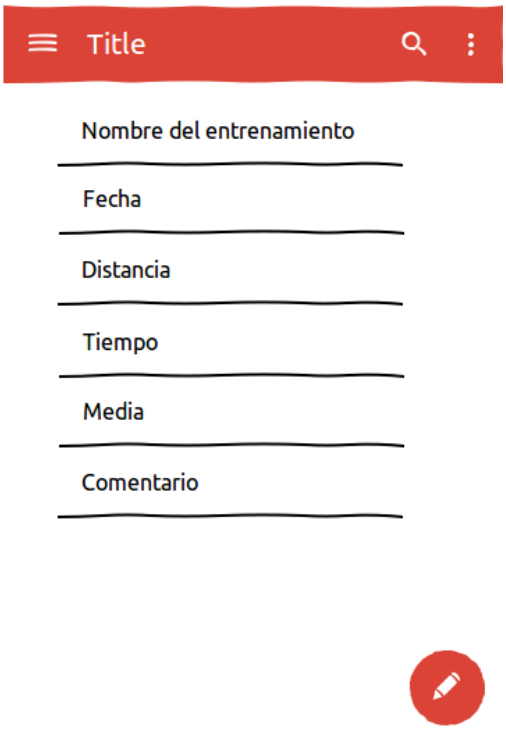

Figura 4.4: Pantalla crear entrenamiento

Definition of done:

- El usuario debe de ser capaz de poder crear un entrenamiento.
- El usuario debe de ser capaz de poder importar un entrenamiento desde otra plataforma.
- El usuario debe de ser capaz de poder borrar un entrenamiento.
- El usuario debe de ser capaz de poder ver todos los entrenamientos que ha realizado.

4.3 Objetivos

Los objetivos son metas que el usuario desea cumplir en un cierto tiempo, ya sea en una carrera o metas personales que el usuario quiere superar para mejorar. Con esta funcionalidad el usuario podrá crear nuevos objetivos, una vez los haya completado podrá

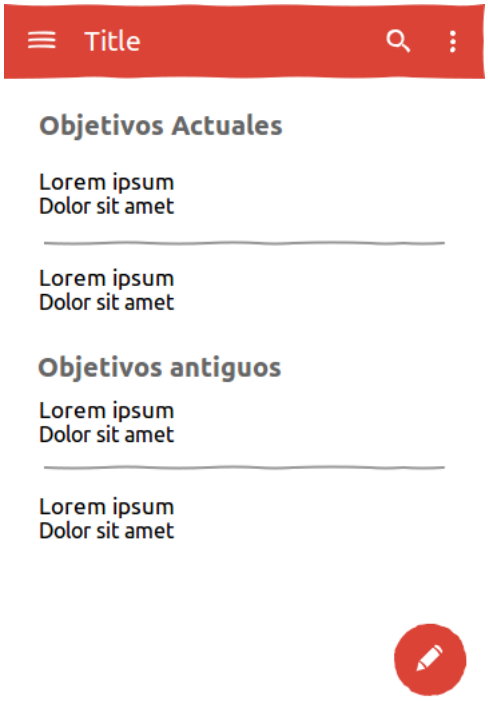

Figura 4.5: Pantalla lista de objetivos

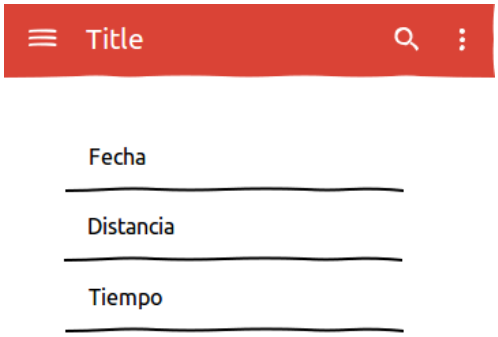

Figura 4.6: Pantalla crear objetivo

marcarlo como terminado y pasará al historial de objetivos donde podrá ver el progreso que ha tenido.

Definition of done:

- El usuario debe de ser capaz de poder crear un objetivo.
- El usuario debe de ser capaz de poder completar un objetivo.
- El usuario debe de ser capaz de poder ver todos los objetivos completados.

4.4 Búsqueda de usuarios

En esta pantalla se le proporcionará al usuario un formulario mediante el cual podrá ajustar los parámetros para encontrar a los usuarios con las características que desee.

Definition of done:

- El usuario debe de ser capaz de poder buscar usuarios.
- El usuario debe de ser capaz de poder ver sus perfiles.

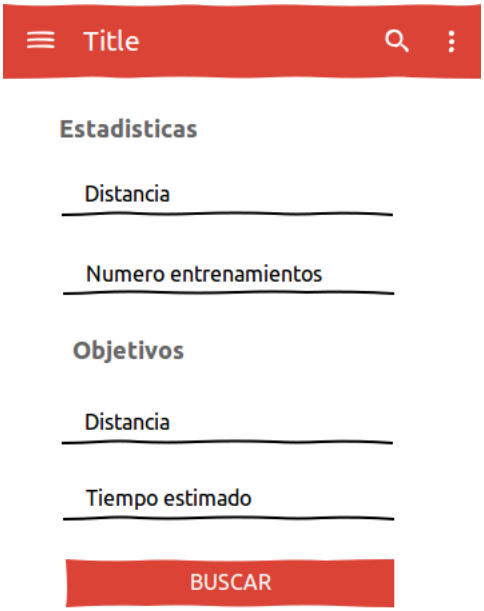

Figura 4.7: Pantalla búsqueda de usuarios

4.5 Perfil

El perfil del usuario muestra unas pequeñas estadísticas referentes a los entrenamientos realizados por el usuarios, tanto de forma global como estadísticas más dinámicas que se van actualizando cada día.

Figura 4.8: Pantalla perfil de usuario

Definition of done:

- El usuario debe de ser capaz de poder ver su perfil o el de otros usuarios.

4.6 Gráficas

Cuando un usuario utiliza la búsqueda para encontrar a otros usuarios puede observar su perfil, pero también puede entrar en una pantalla de gráficas donde se compararán algunos parámetros entre los dos usuarios.

Figura 4.9: Pantalla de gráficas

Definition of done:

- El usuario debe de ser capaz de ver una o varias gráficas donde se le compare con otro usuario.

CAPÍTULO 5
Desarrollo del servidor

5.1 Ruby on Rails

Ruby on Rails (RoR) es un **framework de aplicaciones web** de **código abierto** escrito en Ruby. RoR nace con la idea de simplificar al máximo la tarea del programador haciendo que se pueda desarrollar aplicaciones escribiendo el menor código posible y con poca configuración [5]. Este framework está escrito en **Ruby** que es un lenguaje que para muchos de los usuarios es muy legible en comparación con muchos otros lenguajes y esto es gracias a la **metaprogramación**, que consiste en escribir código que manipula otros programas o a si mismo y también a la realización de tareas que se hacen en tiempo de compilación y que normalmente se harían en tiempo de ejecución [6].

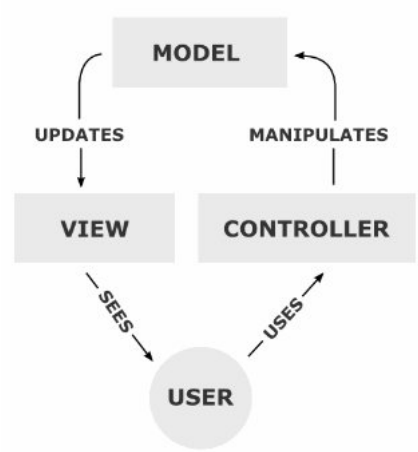

Figura 5.1: Esquema del patrón MVC

Fuente: https://en.wikipedia.org/wiki/Model-view-controller

Sigue el paradigma del patrón **Modelo Vista Controlador** el cual divide la aplicación en tres grandes aspectos. El modelo representa toda aquella información que se necesita para mostrarla en la vista (en RoR el encargado de esta capa es Active Record). La vista representa a todos los componentes gráficos, en ella se representa toda la información de los modelos. Por último tenemos al controlador el cual es responsable de procesar todas las peticiones, actúa como un mediador entre la vista y el modelo [7].

Una de las características de RoR es que uno de los principios en los que se basa es el principio **DRY** (Don't repeat yourself). Este principio se basa en la premisa de que no debería de existir código duplicado con el mismo comportamiento, en vez de tener varios trozos de código iguales dentro del proyecto [8].

5.2 Tecnologías utilizadas

En esta sección se va a comentar algunas de las tecnologías más destacadas en el servidor, tanto librerías para el código como herramientas que ayudan al despliegue del servidor.

5.2.1. RSpec

En cualquier proyecto software la parte de testing es muy importante, con ella podemos encontrar fácilmente errores que puedan aparecer al modificar nuestro código y comprobar que todo lo que hemos realizado funciona correctamente. Ruby on Rails ya tiene por defecto una herramienta que nos permite hacer testing pero gracias a **RSpec** [1] se pueden hacer tests mucho más legibles incluso para personas que no sepan programar. RSpec está pensado para potenciar el **Test Driven Development** (TDD), que se trata de realizar primero los tests de la funcionalidad que queramos desarrollar y posteriormente escribir el código que hagan pasar estos test, esta idea se basa en que hay que escribir el mínimo código posible que completar la funcionalidad que deseamos.

5.2.2. Sidekiq

Por otro lado también cabe destacar **Sidekiq** [2], se trata de una herramienta para la realización de tareas en segundo plano de una manera mucho más eficiente. Sidekiq utiliza hilos para poder manejar varios trabajos en un mismo proceso. Esta librería está hecha para Ruby pero se integra muy bien con RoR.

5.2.3. Heroku

A la hora de desplegar un servidor hacen falta ciertos conocimientos en sistemas los cuales muchos de los programadores en un principio no conocen. Existen herramientas las cuales facilitan este proceso y **Heroku** [3] es una de ellas. Heroku es una plataforma como servicio (PaaS) que permite a los programadores desplegar aplicaciones en la nube de una forma muy sencilla y con un alto grado de configuración. Heroku se basa en un sistema de contenedores permitiéndonos escalar nuestras aplicaciones de forma muy sencilla en cuestión de minutos.

5.3 Arquitectura del proyecto

Generalmente un proyecto de Ruby on Rails tiene la estructura que se puede observar en la figura 5.2. A continuación se va a comentar algunos de los directorios más destacados de este tipo de proyectos:

- **app**: Este directorio contiene todos los componentes principales del servidor como los controladores, vistas, modelos...
- **config**: Contiene todo lo relacionado con la configuración del proyecto: base de datos, entornos, archivo de rutas...

[1] RSpec URL: https://rspec.info
[2] Sidekiq URL: https://github.com/mperham/sidekiq
[3] Heroku URL: https://www.heroku.com/platform

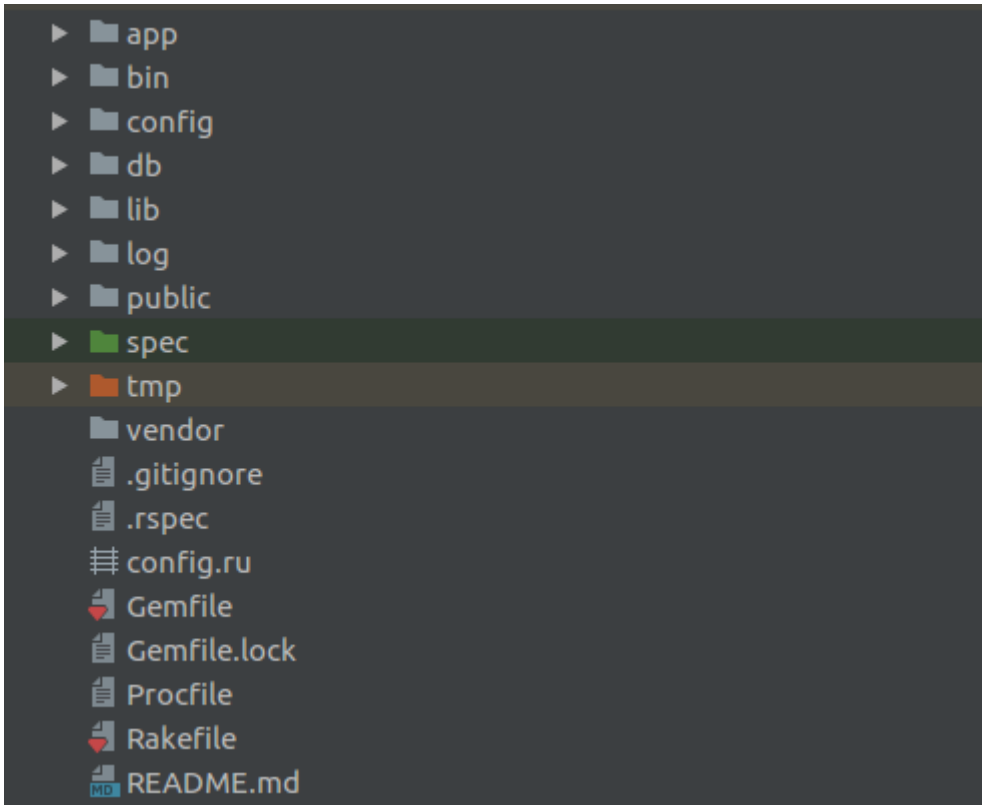

Figura 5.2: Estructura de un proyecto de Ruby on Rails

- **db**: Contiene archivos referentes a la base de datos. En el caso de utilizar una base de datos SQL podremos encontrar los ficheros con las migraciones que se han realizado en la base de datos.

- **spec**: En este directorio están todos los ficheros de test que se realicen, podemos encontrar carpetas con test para los controladores, modelos...

5.4 Desarrollo

Durante toda esta sección se describirán diversos aspectos del desarrollo de la parte del servidor. El desarrollo del servidor y de la aplicación móvil se ha realizado de forma paralela pero para facilitar la lectura de las dos cosas se ha decidido comentar cada uno de los desarrollos por separado.

Hablar de todo el desarrollo de un proyecto software puede ser muy extenso, por eso se realizará una descripción de los apartados más importantes para que cualquier persona pueda entender como se ha realizado el desarrollo.

5.4.1. Base de datos

En la sección de análisis (sección 3) se ha podido ver la estructura que tendrá la base de datos en el servidor. Durante el desarrollo se utilizará **SQLite3**, que es la herramienta que viene por defecto. Para el entorno de producción se va a utilizar **PostgreSQL** que es la herramienta proporcionada por **Heroku** para montar bases de datos relacionales.

5.4.2. Desarrollo de la API

El servidor desarrollado se trata de una **API** (Application Programming Interface) que expondrá al usuario una serie de rutas (llamadas endpoints) mediante las cuales podrá acceder (o modificar) a los diversos datos del servidor.

La gran mayoría de los endpoints comparten la misma estructura que podemos observar en la figura 5.3. Primeramente están los **controladores** que son archivos los cuales contienen las definiciones de distintos endpoints (leer, crear, borrar, actualizar...) los cuales afectan sobre cierta colección de datos (por ejemplo los entrenamientos). Los controladores son los encargados de recibir la llamada que ha realizado el usuario sobre cierto endpoint. Desde los controladores se llaman a las **acciones** que se tratan de los casos de uso desde donde se va a realizar toda la lógica. Por último las acciones llaman a los **servicios** que son los encargados de realizar cualquier interacción sobre la base de datos y devolverle la información a las acciones. Una vez toda la información ha sido obtenida y manipulada en las acciones se le devuelve al controlador y este le devuelve una respuesta al usuario, ya sea una respuesta correcta o un error.

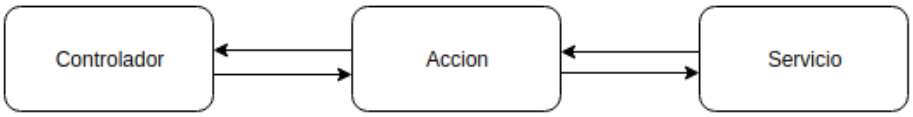

Figura 5.3: Estructura de un endpoint

Para mostrar todo el procedimiento de creación de un endpoint se va a mostrar todos los procesos seguidos en el desarrollo del endpoint para crear los entrenamientos. Este endpoint a parte nos enseña también como utilizar la librería **Sidekiq** para lanzar tareas en segundo plano.

Primeramente antes de empezar el desarrollo del endpoint hay que crear la ruta en el archivo de rutas que se puede encontrar en la carpeta *config/routes.rb* del proyecto. Para facilitar el desarrollo en API con distintas versiones a veces es usual separar cada versión en distintos archivos, por ello los endpoints se han declarado en *config/routes/api_v1.rb*. Al tratarse de una API con una sola versión no seria necesario esta separación pero para más claridad se ha seguido los estándares.

Como podemos observar en la figura 5.4 la línea `resources :trainings` es la encargada de crear todas las rutas necesarias para poder manipular los entrenamientos, que son las siguientes:

```
namespace :api, defaults: {format: 'json' } do
  namespace :v1 do
    resources :trainings
    resources :statistics, only: [:index]
    resources :users, only: [:show] do
      get 'search', on: :collection
    end
  end
end
```

Figura 5.4: Archivo de rutas de la API

- **index (GET api/v1/trainings)**: muestra una lista de entrenamientos

- **show (GET api/v1/trainings/{id})**: muestra un entrenamiento
- **create (POST api/v1/trainings)**: crea un entrenamiento
- **update (PUT api/v1/trainings/{id})**: modifica un entrenamiento
- **destroy (DELETE api/v1/trainings/{id})**: destruye un entrenamiento

```
def create
  user = current_resource_owner

  param! :name, String, required: true
  param! :date, Integer, required: true
  param! :distance, Float, required: true
  param! :moving_time, Integer, required: true
  param! :average_speed, Float, required: true
  param! :max_speed, Float, required: true
  param! :strava_id, Integer, required: true

  training = CreateTraining.run(user, training_params)
  render json: {message: "Training created successfully.",
                 data: training.id}, status: 201
end
```

Figura 5.5: Método `create` del controlador de entrenamientos

Una vez creada la ruta hay que crear el método del controlador el cual manejará la llamada al endpoint, en este caso se trata del método `create`. En cualquier método de los controladores el primer paso a realizar es una comprobación de los parámetros que se reciben por cualquier llamada a ese endpoint, esta comprobación se facilita gracias a la gema `rails_param`. Como se puede observar en la figura 5.5 se definen todos los parámetros que va a aceptar la llamada, el tipo que se espera y si son necesarios o opcionales.

```
class CreateTraining
  class << self
    def run(user, training_params)
      training = TrainingService.create(user, training_params)
      UpdateUserStatisticsWorker.perform_async(user.id,
        training.distance, training.moving_time)
      training
    end
  end
end
```

Figura 5.6: Ejemplo de acción

A continuación se procede a llamar a la acción a la cual le corresponda manejar la llamada, en este caso se trata de `CreateTraining`. Aunque este no sea el caso las acciones pueden contener más de un caso de uso dentro de ellas. Dentro de las acciones se realiza toda la lógica necesaria para manipular la respuesta de la llamada, en este caso se pide los datos al servicio, se lanza un worker y se devuelve la información al controlador, que en este caso se trata del entrenamiento que se ha creado en la base de datos.

```
def create(user, training_params)
  if training_params[:strava_id].present?
    if training_params[:strava_id] != 0
      check_strava_id(training_params[:strava_id])
  end
  training = user.trainings.create(training_params)
  Errors.couldnt_create_training unless training.save
  training
end
```

Figura 5.7: Ejemplo de método de un servicio

Los servicios son archivos en los cuales existen métodos que se comunican directamente con la base de datos, suelen ser acciones concretas como en este caso crear un entrenamiento. En los servicios se realizan las comprobaciones necesarias, si la acción no se ha podido completar se lanza un error para informar al usuario. Una vez que el servicio ha terminado su tarea devuelve a la acción que le ha invocado una respuesta.

Si todo el proceso se ha realizado sin ningún error el controlador recibe la respuesta y se la devuelve al usuario, como podemos ver en la última línea de la figura 5.5.

```
class UpdateUserStatisticsWorker
  include Sidekiq::Worker

  def perform(user_id, training_distance, training_time)
    user = User.find(user_id) rescue nil
    Errors.user_not_found unless user
    statistic = user.statistic rescue nil
    Errors.statistic_not_found unless statistic

    statistic.total_trainings += 1
    statistic.total_distance += training_distance
    statistic.total_time += training_time

    week_trainings = User.find(user_id).trainings.where("date >= ?",
                                (Date.today - 7.day).to_time.to_i)
    statistic.weekly_trainings = week_trainings.count
    statistic.weekly_time = week_trainings.sum(:moving_time)
    statistic.weekly_distance = week_trainings.sum(:distance)

    statistic.save
  end
end
```

Figura 5.8: Worker para ejecutar tareas en segundo plano

En figura 5.6 podemos ver como desde la acción se llama a `UpdateUserStatisticsWorker` (figura 5.8) que se trata de una clase que ejecutará una tarea en segundo plano. Esta tarea se encargará de actualizar las estadísticas del usuario, al no ser una información necesaria para la respuesta de la llamada podemos realizarla en segundo plano sin ningún problema, así reducimos el tiempo de calculo de la respuesta y esta se entregará en un menor tiempo al usuario.

Una vez desarrollados todo lo necesario para que el *endpoint* funcione correctamente se realizan los tests. Hacerlos es opcional, pero a la larga ofrecen grandes beneficios ya que podremos estar seguros simplemente ejecutando toda la batería de tests que tenemos si todo lo que realizamos (o si cambiamos alguna cosa ya hecha) funciona correctamente. Es verdad que los tests pueden ser un arma de doble filo ya que debemos saber como realizarlos correctamente porque un test mal hecho puede hacernos creer que todo funciona correctamente cuando no es así. Los tests son una tarea la cual hay que dedicar un tiempo en aprender y poco a poco ir mejorando.

Como hemos comentado anteriormente para la parte de servidor vamos a utilizar **RSpec** para realizar los test ya que dada su facilidad de uso realizar tests es una tarea muy sencilla a comparación con otros tipos de lenguajes. Para realizarlos tests vamos a crear uno por cada una de las clases creadas en el desarrollo del endpoint:

- **Controlador**: Comprobar todas las respuestas posibles que puedan haber: que la respuesta sea correcta, que devuelva error si el usuario no ha iniciado sesión, comprobar errores que hayamos lanzado nosotros por incumplirse alguna condición...
- **Acción**: Comprobar que se realiza bien la lógica dentro de la acción. En nuestro caso se tratan de acciones muy sencillas y este tipo de test suelen ser muy sencillos, pero en acciones más complicadas se podría testear por ejemplo que se realiza un correcto filtrado de la respuesta recibida desde el servicio.
- **Servicio**: Comprobar que la base de datos nos devuelve lo que realmente le hemos pedido: comprobar como está ordenado, comprobar si está limitado el número de objetos a devolver(si lo hemos especificado)...
- **Worker**: Comprobar que el *worker* realiza la tarea que deseamos correctamente, por ejemplo en nuestro caso comprobar que `UpdateUserStatisticsWorker` calcula bien las estadísticas del usuario.

Escribir tests es una tarea muy grande, y tan importante como el propio desarrollo, de ello que no exista una única forma de hacerlos y por lo tanto la que hemos presentado arriba es una forma básica de comprobar que el funcionamiento de lo que hemos desarrollado.

En la figura 5.9 podemos ver un ejemplo de un test del controlador de entrenamientos para el endpoint de creación de entrenamientos, en este test se puede observar como gracias a RSpec se pueden realizar tests muy legibles con facilidad.

```
describe "POST /api/v1/trainings" do
  it "creates a new training" do
    act_as_logged_in @user
    header "Content-Type", "application/json"
    post "api/v1/trainings", {
      name: "Training",
      date: 123456,
      distance: 17.6,
      moving_time: 123,
      average_speed: 20.6,
      max_speed: 30.5,
      strava_id: 0
    }.to_json

    expect(last_response.status).to eq(201)
  end

  it "gets an error if the user is not logged" do
    header "Content-Type", "application/json"
    post "api/v1/trainings", {
      name: "Training",
      date: 123456,
      distance: 17.6,
      moving_time: 123,
      average_speed: 20.6,
      max_speed: 30.5
    }.to_json

    expect(last_response.status).to eq(401)
  end
end
```

Figura 5.9: Ejemplo test controlador

CAPÍTULO 6
Desarrollo de la aplicación móvil

6.1 Android

Android es un sistema operativo desarrollado por la empresa Google (inicialmente el desarrollo fue comenzado por la empresa Android Inc. que compro Google en el año 2005) basado en una versión modificada del kernel de Linux. En los últimos años el sistema ha ido creciendo de gran forma y por ello podemos ver versiones adaptadas del sistema operativo para coches (Android Auto), televisiones (Android TV) o relojes (Wear OS).

Android fue desvelado en 2005 y la primera versión comercial se lanzó en 2007, desde entonces han ido saliendo muchas mejoras del sistema operativo (las cuales se pueden observar en la figura 6.1).

Es el sistema operativo móvil más vendido mundialmente consiguiendo una cifra de más de 2 mil millones de usuarios activos en el año 2017.

Versión	Nombre	Fecha lanzamiento	API
?	Android P	?	28
8.1	Oreo	6 Diciembre 2017	27
8.0	Oreo	21 Agosto 2017	26
7.1	Nougat	4 Octubre 2016	25
7.0	Nougat	22 Agosto 2016	24
6.0	Marshmallow	5 Octubre 2015	23
5.1	Lollipop	9 Marzo 2015	22
5.0	Lollipop	3 Noviembre 2014	21
4.4	KitKat	31 Octubre 2013	19
4.3	Jelly Bean	24 Julio 2013	18
4.2	Jelly Bean	13 Noviembre 2012	17
4.1	Jelly Bean	9 Julio 2012	16
4.0	Ice Scream Sandwich	19 Octubre 2011	15
2.3	Gingerbread	9 Febrero 2011	10

Tabla 6.1: Versiones de Android

Fuente: https://en.wikipedia.org/wiki/Android_(operating_system)

6.2 Kotlin

Desde el lanzamiento de Android el desarrollo de una aplicación móvil para Android se realizaba con el lenguaje Java, un lenguaje que para mucha gente se ha quedado desactualizado actualmente en comparación con otros lenguajes. La empresa Jetbrains conocida por crear programas como IntelliJ IDEA, en el cual se basa la herramienta oficial para el desarrollo Android (Adroid Studio), decidieron crear un nuevo lenguaje el cual permita un desarrollo más eficiente. En 2016 se presenta la primera versión de Kotlin y un año después durante la Google I/O de 2017 se anuncia Kotlin como lenguaje oficial de Android.

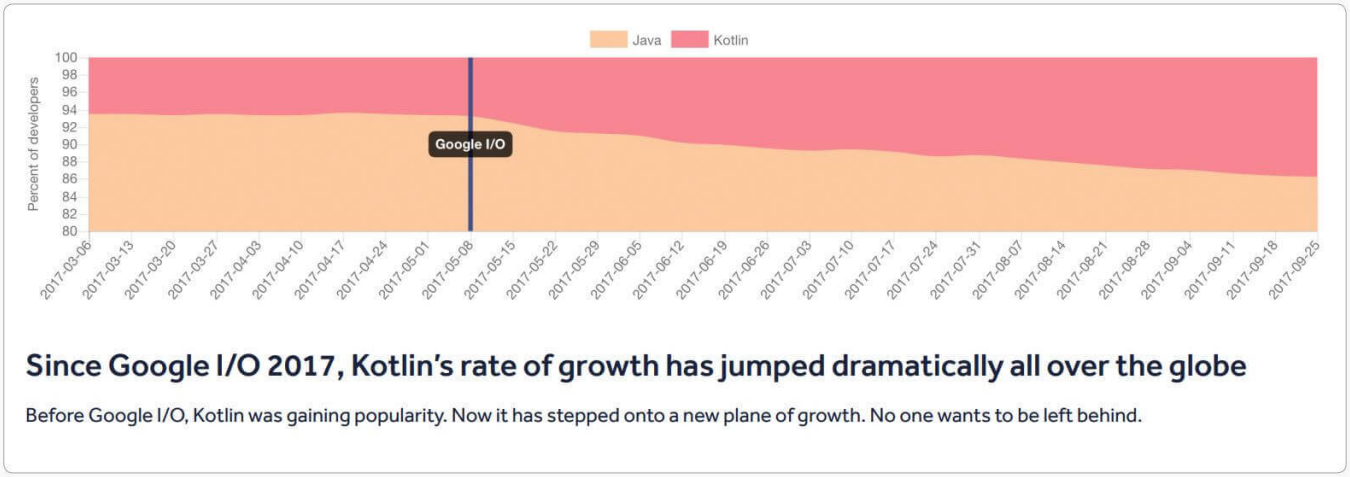

Figura 6.1: Crecimiento del número de desarrolladores Kotlin desde la Google I/O de 2017

Fuente: https://www.spaceotechnologies.com/kotlin-official-programming-language-android-app-development/

Desde sus comienzos Kotlin tuvo un gran apoyo por una gran cantidad de usuarios dados a sus muchas mejoras respecto a Java. Algunas de estas mejoras son:

- Compila código que puede ser ejecutado en la JVM (Java Virtual Machine).
- Se puede compilar también código Javascript (lo cual también permite desarrollar aplicaciones para navegadores).
- Código simplificado y elegante (Comparación en la figura 6.2).
- Totalmente interoperable con Java permitiéndonos así utilizar cualquier librería o clase escrita en Java.
- El código en Java puede convertirse a Kotlin con facilidad.
- El sistema de tipos está enfocado en eliminar el peligro de las referencias a objetos nulos.

Kotlin es un lenguaje muy joven pero que dado el gran apoyo de la comunidad (apoyando en el desarrollo del lenguaje y en el desarrollo de muchas librerías escritas completamente en Kotlin dado su poder) está creciendo de manera muy rápida haciendo que sea cuestión de poco tiempo para que todo el desarrollo nuevo en Android se realice en este lenguaje.

```java
// JAVA
public class User {
  private final String firstName;
  private final String lastName;
  private final int age;

  public User(String firstName, String lastName, int age) {
    this.firstName = firstName;
    this.lastName = lastName;
    this.age = age;
  }

  public String getFirstName() {
    return firstName;
  }

  public String getLastName() {
    return lastName;
  }

  public int getAge() {
    return age;
  }
}

class Main {
  public static void main(String[] args) {
    User user = new User("John", "Doe", 30));
  }
}

// KOTLIN
data class User(val firstName: String,
                val lastName: String,
                val age: Int)

fun main(args: Array<String>) {
  user = User("John", "Doe", 30))
}
```

Figura 6.2: Comparación de código Java y Kotlin

6.3 Tecnologías utilizadas

El desarrollo de una aplicación Android se puede hacer utilizando simplemente todo lo proporcionado por el propio sistema pero como es habitual el uso de librerías facilita ampliamente el desarrollo. Existen alguna librerías de uso tan extendido que casi se considera necesario conocer su utilización, dos librerías que cumplen esto son **Retrofit** y **RxJava**.

6.3.1. Retrofit

Realizar consultas a una API es una acción muy común actualmente ya que la gran mayoría de aplicaciones están conectadas a internet. Para facilitar esta tarea una de las librerías más utilizadas es Retrofit, permitiéndonos consumir datos JSON o XML de una forma muy sencilla.

Mediante esta librería se definen todas las llamadas a una API de una forma mucho más legible y sencilla de realizar para el desarrollador. Posteriormente al utilizar una de las llamadas previamente definidas Retrofit se encarga de realizar la llamada a la API, una vez terminada la llamada convierte el resultado al modelo de datos que le especifiquemos. La conversión que se realiza es de texto en formato JSON a POJOs (Plain Old Java Object). Retrofit acepta distintos tipos de conversores según nuestras necesidades, en nuestro caso se ha utilizado el conversor Moshi.

6.3.2. RxJava

Consumir datos de una API es un proceso en el cual algunos de los casos puede tardar cierto tiempo, durante este tiempo es poco deseable dejar al usuario con la aplicación congelada y por lo tanto es preferible hacer las llamadas en segundo plano. RxJava es una librería la cual nos facilita esta tarea mediante secuencias de observables, extiende el patrón observable para soportar las secuencias de datos o eventos y añade una gran cantidad de operadores para trabajar con estas secuencias.

Como hemos comentado anteriormente existen ciertas librerías en Android las cuales es muy común su uso, la combinación de las dos librerías presentadas suele ser bastante habitual a la hora de desarrollar una aplicación que consume datos de internet.

La fusión de estas librerías es muy sencilla y muy potente. En las definiciones de las llamadas de Retrofit simplemente tenemos que decirle que esperamos secuencias de observables del modelo del cual esperamos la respuesta. Una vez realizada la llamada con RxJava podemos manipular esta secuencia de la forma que deseemos. También es muy útil a la hora de definir que acción hay que realizar en el caso de que se produzca un error en la llamada.

En esta sección se ha presentado las dos librerías de una forma breve, durante la sección 6.5 se presentará el uso práctico dentro del código de la aplicación.

6.4 Arquitectura del proyecto

En la figura 6.3 se puede observar la estructura que tiene un proyecto de Android. Las carpetas que componen el proyecto y su función son las siguientes:

- **app**: Es la carpeta principal del proyecto donde se almacena todo el código y recursos necesarios para el desarrollo.

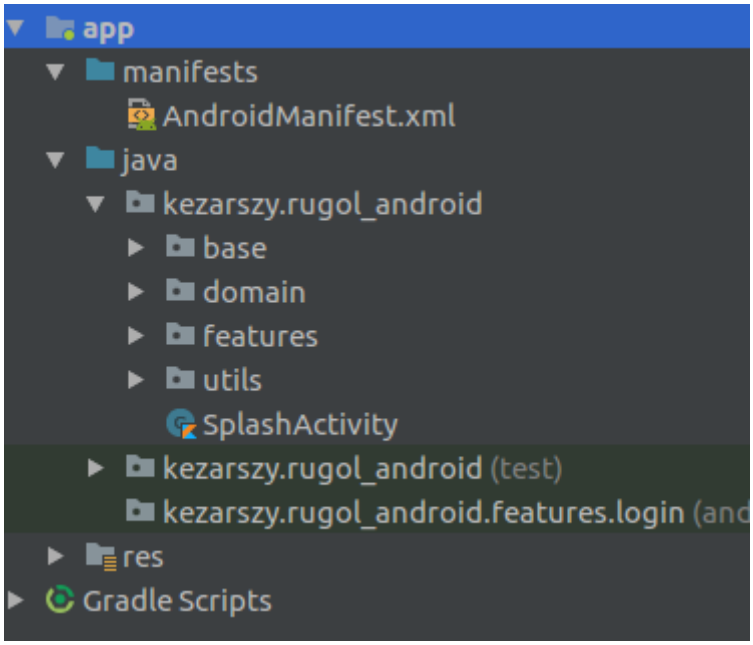

Figura 6.3: Estructura de un proyecto Android

- **app/manifests**: El manifiesto es un archivo necesario en cualquier aplicación, en este se proporciona toda la información esencial sobre la aplicación al sistema de Android para poder ejecutarla correctamente.
- **app/java**: En esta carpeta se encuentra todo el código de la aplicación. Por una parte tenemos la carpeta principal donde el usuario creará las actividades, modelos, etc... Por otra parte hay dos carpetas en las cuales si el usuario desea puede escribir test unitarios y de integración.
- **app/res**: Aquí se almacenan todos los recursos necesarios para la aplicación: layouts, imágenes, iconos, colores, fuentes, textos...
- **gradle scripts**: En esta carpeta está todo lo relacionado a la compilación del proyecto, para ello se utiliza Gradle, un paquete de herramientas de compilación avanzado para automatizar y administrar todo el proceso permitiendo una gran configuración y personalización.

6.5 Desarrollo

En esta sección se comentará el desarrollo de una de las funcionalidades de la aplicación, para seguir con la continuidad del desarrollo comentado por parte del servidor se hablará de la funcionalidad de crear un entrenamiento desde la aplicación móvil.

6.5.1. Arquitectura MVP

Al igual que en el servidor en el desarrollo de la aplicación hemos utilizado una arquitectura para estructurar todo el código, esta arquitectura se llama MVP (Model View Presenter). Se podría decir que esta arquitectura es una derivación del MVC (visto anteriormente en el desarrollo del servidor) [10].

Es una arquitectura la cual consigue desacoplar de la vista toda la lógica, con esto lo que se consigue son dos mejoras. La primera mejora es que gracias a esta separación el

desarrollador es capaz de testear la aplicación con bastante más facilidad que teniéndolo todo acoplado en la vista. Otra de las grandes mejoras es la modularidad que nos proporciona, es decir, a la hora de querer cambiar alguna implementación (por ejemplo la base de datos local) se realizará de una forma mucho más sencilla al no depender tanto unas cosas de otras.

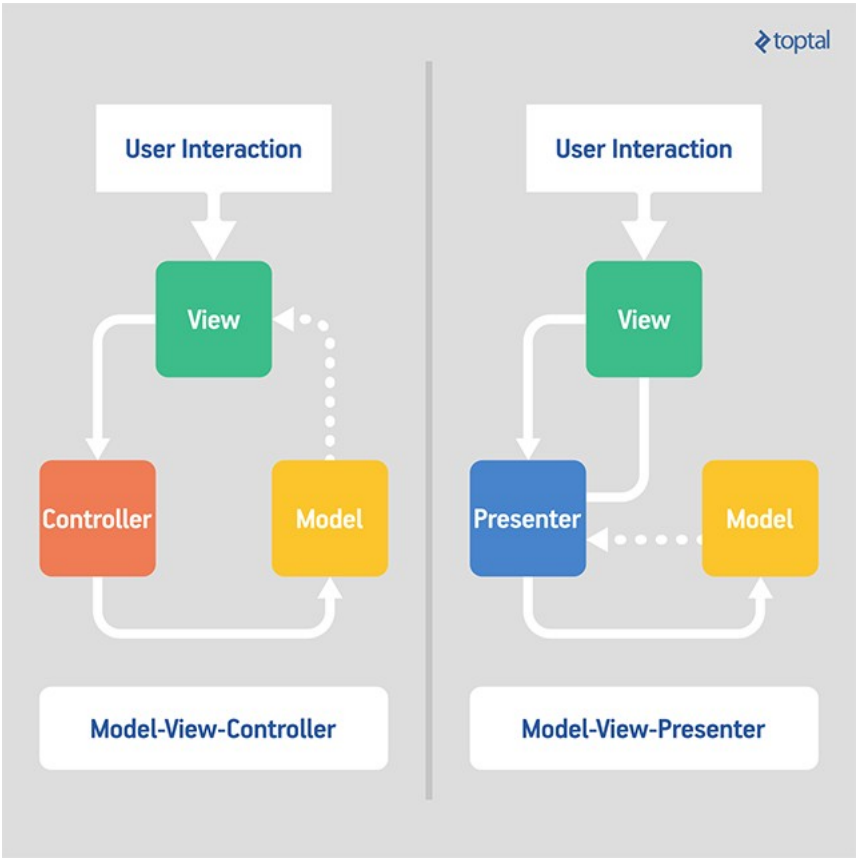

Figura 6.4: Comparación MVC y MVP

Fuente: https://medium.com/cr8resume/make-you-hand-dirty-with-mvp-model-view-presenter-eab5b5c16e42

Esta arquitectura como su nombre indica está separada en tres capas:

- **Model**: Es la capa encargada de realizar todas las acciones con los modelos de datos: llamadas a la API, cachear los datos, manejo de base de datos local... En nuestro caso se utilizará el patrón Repository (explicado posteriormente) para realizar esta capa.
- **View**: Es la capa encargada de mostrar los datos al usuario, normalmente esta capa está implementada mediante un Activity o un Fragment de Android.
- **Presenter**: Capa que hace de mediador entre la vista y el modelo. Recoge todos los datos proporcionados por el modelo y se los entrega en el formato necesario a la vista. Esta capa es la encargada de toda la lógica de presentación.

Esta es una de las arquitecturas más extendidas en el desarrollo de aplicaciones Android por ello es normal que existan ciertas discrepancias en la forma de utilizarlo entre desarrolladores, pero todas ellas coinciden de forma teórica en lo que se ha descrito.

6.5.2. Desarrollo de una funcionalidad

Durante esta sección se van a comentar los aspectos más importantes y el código referente a ellos a la hora de desarrollar una funcionalidad nueva, en este caso la de crear un nuevo entrenamiento. El desarrollo de una funcionalidad pasa por todas las etapas que se ven en la figura 6.4, en la parte del Model-View-Presenter, y podemos verlo de una forma más ampliada en la figura 6.5.

Primeramente es necesario definir la `Activity` (Vista) la cual va a alojar la funcionalidad (una funcionalidad más grande podría ocupar más de una `Activity`). En mi caso primeramente siempre me gusta empezar definiendo toda la parte visual de la pantalla y el comportamiento por defecto que tengan dentro del fichero de la `Activity`.

Una vez terminada la base de la `Activity` pasamos a definir los métodos que necesitemos en el `Presenter` (figura 6.1), en este caso tendremos dos métodos, uno de ellos es el que llamará la `Activity` cuando el usuario pulse el botón para crear el entrenamiento y otro manejará el resultado de todo el proceso.

```
1   override fun createTraining(name: String, date: Long, distance: Double,
        movingTime: Int, averageSpeed: Double, maxSpeed: Double, comment:
        String, userId: Int) {
2
3     view?.showLoader()
4     view?.disableTouch()
5
6     val training = Training(1, name, date, distance, movingTime, averageSpeed
          , maxSpeed, comment, userId)
7     trainingsProvider.createTraining(
8       training, this::handleCreateTrainingResult
9     )
10  }
11
12  private fun handleCreateTrainingResult(result: CreateTrainingResult) {
13    view?.hideLoader()
14    when(result) {
15      is CreateTrainingResult.CreateTrainingSuccess -> view?.
            createTrainingSuccess()
16      is CreateTrainingResult.CreateTrainingFail -> {
17        view?.showError(R.string.error_create_training)
18        view?.enableTouch()
19      }
20    }
21  }
```

Listing 6.1: Métodos del Presenter

Como podemos observar en el método `createTraining` del `Presenter` (figura 6.1) se llama al `Provider` y se le pasa una instancia del entrenamiento que se desea crear. El `Provider` (figura 6.2) es una clase la cual va a realizar las llamadas a los repositorios que sea necesario, en nuestro caso se llaman al repositorio de nuestro servidor y al de la base de datos local. En el caso de solo tener un repositorio esta clase se podría omitir y llamar directamente al repositorio desde el `Presenter`. El patrón *Repository* es utilizado con la idea de que los datos sean transparentes para el cliente [9], es decir, que no tenga que preocuparse de si vienen por ejemplo de un servidor o de la base de datos local (como es nuestro caso).

Antes de continuar con la explicación del `Provider` se va a dar una breve explicación de los `Observable`, estos objetos son de la librería **RxJava** comentada anteriormente. Implementan el patrón `Observable` en el cual nos suscribimos a estos objetos reaccionando

a cualquier secuencia que nos devuelvan. Esto se utiliza a la hora de realizar llamadas asíncronas porque de esta forma no es necesario bloquear al usuario hasta recibir la respuesta.

Si observamos la cabecera del método `createTraining` del `Provider` (figura 6.2) podremos ver como tiene dos argumentos, el entrenamiento que se desea crear y la función la cual manejara el resultado de este método (que en nuestro caso es el método `handleCreateTrainingResult` del `Presenter`).

En el método primeramente se llama al caso de uso para crear un nuevo entrenamiento utilizando como repositorio nuestra API. En la línea 5 de la figura 6.2 podemos ver como se realiza el `subscribe` donde se esperara a recibir una respuesta. En este momento hay dos posibles casos, si todo se realiza correctamente en el caso de uso se pasa a utilizar el repositorio de la base de datos local y crear el entrenamiento, en el caso de que durante del caso de uso haya algún fallo este se notificara al `Presenter`. Si las dos llamadas a los casos de uso se realizan con éxito se notificará de esto al `Presenter`.

```
1   fun createTraining(training: Training, callback: (CreateTrainingResult) ->
        Unit) {
2     createTrainings.repository = apiTrainingRepository
3     with(training) {
4       var params = mutableListOf("1", name, date.toString(), distance.
            toString(), movingTime.toString(), averageSpeed.toString(),
            maxSpeed.toString(), comment, userId.toString(), stravaId.toString
            ())
5       createTrainings.invoke(params).subscribe({
6         when(it) {
7           is CreateTrainingResult.CreateTrainingSuccess -> {
8             createTrainings.repository = dbTrainingRepository
9             params[0] = it.response.data
10            createTrainings.invoke(params).subscribe({
11              callback(it)
12            })
13          }
14          is CreateTrainingResult.CreateTrainingFail -> {
15            callback(CreateTrainingResult.CreateTrainingFail(CreateTraining.
                  CREATE_TRAINING_FAIL))
16          }
17        }
18      })
19    }
20  }
```

Listing 6.2: Métodos del Provider

Como hemos comentado el encargado de llamar al repositorio es el caso de uso, que se trata de una clase que a parte de poder realizar alguna comprobación lógica lanza la llamada al repositorio de forma asíncrona a un nuevo hilo para así evitar que todas las operaciones se vayan a realizar en el hilo principal.

En la figura 6.3 podemos ver el método del repositorio que para crear el entrenamiento. En este método se realiza una llamada a la API y se devuelve un `Observable` el cual puede contener el entrenamiento indicando que se ha realizado la llamada correctamente o un código de fallo indicando que la llamada no se ha podido realizar o ha fallado al realizarse.

```
1   override fun createTraining(training: Training): Observable<
        CreateTrainingResult> =
2   Observable.create<CreateTrainingResult> { emitter ->
3     with(training) {
4       api.postCreateTraining(name, date, distance, movingTime, averageSpeed,
            maxSpeed, comment, stravaId)
```

```
            .subscribe(
              {
                emitter.onNext(CreateTrainingResult.CreateTrainingSuccess(it))
                emitter.onComplete()
              },
              {
                emitter.onNext(CreateTrainingResult.CreateTrainingFail(
                    CreateTraining.CREATE_TRAINING_FAIL))
                emitter.onComplete()
              }
            )
        }
    }
```

Listing 6.3: Repositorio de la API

Por último en la figura 6.4 podemos ver el código necesario para realizar la llamada a la API. Esta llamada se hace mediante la librería **Retrofit** la cual podemos ver que simplifica mucho todo el proceso. Retrofit está caracterizado por dos cosas, por una parte tenemos el `Manager` en el cual tendremos la configuración necesaria para poder llamar a nuestra API y por otra parte la definición de todos los servicios los cuales tienen la definición de todos los endpoints de nuestro servidor.

Como podemos ver en la figura 6.4 tenemos el método `postCreateTraining` el cual realiza llama al servicio de entrenamientos para poder crear el entrenamiento en el servidor. Si nos fijamos en el método podemos ver las llamadas `subscribeOn` y `observeOn` las cuales son las encargadas de lanzar la llamada en segundo plano y que posteriormente se devuelva la respuesta.

```
    // Metodo del API Manager que ejecuta la llamada
    fun postCreateTraining(name: String, date: Long, distance: Double,
        movingTime: Int,
                           averageSpeed: Double, maxSpeed: Double, comment:
                               String, stravaId: Int = 0) =
      trainingService.postCreateTraining(name, date, distance, movingTime,
          averageSpeed, maxSpeed, comment, stravaId)
          .subscribeOn(Schedulers.io())
          .observeOn(AndroidSchedulers.mainThread())!!

    // Definicion de la llamada en el servicio
    @FormUrlEncoded
    @POST(TRAINING)
    fun postCreateTraining(@Field(NAME) name: String,
                           @Field(DATE) date: Long,
                           @Field(DISTANCE) distance: Double,
                           @Field(MOVING_TIME) movingTime: Int,
                           @Field(AVERAGE_SPEED) averageSpeed: Double,
                           @Field(MAX_SPEED) maxSpeed: Double,
                           @Field(COMMENT) comment: String,
                           @Field(STRAVA_ID) stravaId: Int) : Observable<
                               Response>
```

Listing 6.4: Llamada a la API

En los dos métodos podemos observar como se realiza el uso de los `Observable` para que las llamadas a la API se realicen de forma asíncrona, es muy común utilizar las dos librerías (RxJava y Retrofit) simultáneamente ya que las dos se complementan muy bien.

Durante esta sección se ha realizado un breve resumen del proceso que tiene realizar una nueva funcionalidad de la aplicación. Hemos podido ver como se realiza una gran separación de todas las etapas (las cuales podemos en el diagrama de la figura 6.5), permitiéndonos así una gran modularidad en el proyecto que nos ofrece grandes beneficios

en cosas como el *testing* o a la hora de cambiar alguna de las tecnologías utilizadas en el proyecto.

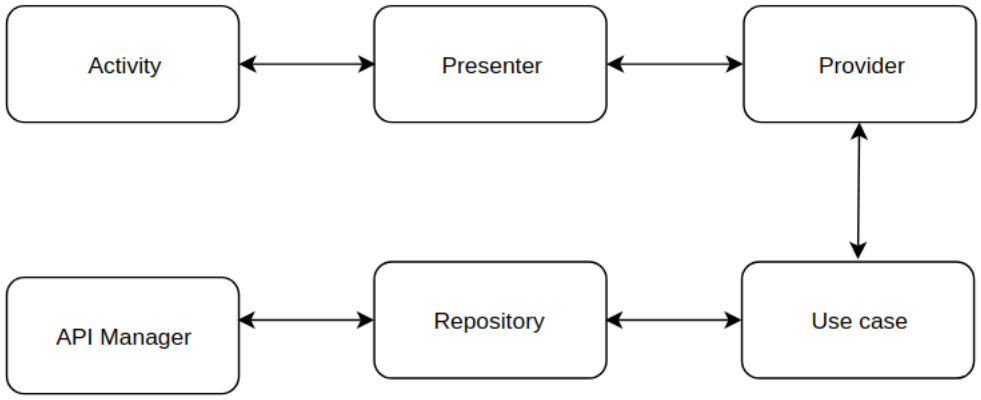

Figura 6.5: Diagrama de la estructura de una funcionalidad en Android

En Android realizar tests es más complicado en comparación con otros lenguajes como por ejemplo en *Ruby on Rails* como hemos visto anteriormente. Por esta razón dada que la dedicación de tiempo para poder realizar unos buenos tests es mucho más grande no se ha podido realizar. Las comprobaciones al realizar una funcionalidad nueva se han hecho manualmente, cosa que está bien pero no es suficiente. Si el proyecto se realizase con más tiempo una de las tarea para realizar más importante seria añadir tests.

CAPÍTULO 7
Resultado final

En esta sección vamos a ver el resultado final de todo el trabajo realizado durante el desarrollo del proyecto software. Se mostrarán imágenes del resultado de la aplicación ya que de la parte del servidor no existe ninguna forma visual de verlo, pero todo el comportamiento de la aplicación funciona gracias a su correcto funcionamiento.

Primeramente el usuario al entrar a la aplicación se encontrará con una pequeña pantalla de carga (figura 7.1) donde se muestra el logo de la aplicación. Durante esta carga se realiza la comprobación de si el usuario necesita iniciar sesión o puede continuar directamente a la pantalla principal.

Figura 7.1: Pantalla de carga (versión final)

En el caso de que el usuario entre por primera vez a la aplicación o no haya iniciado sesión se le proporcionará esas posibilidades con las pantallas de inicio de sesión (figura 7.2) o registro (figura 7.3).

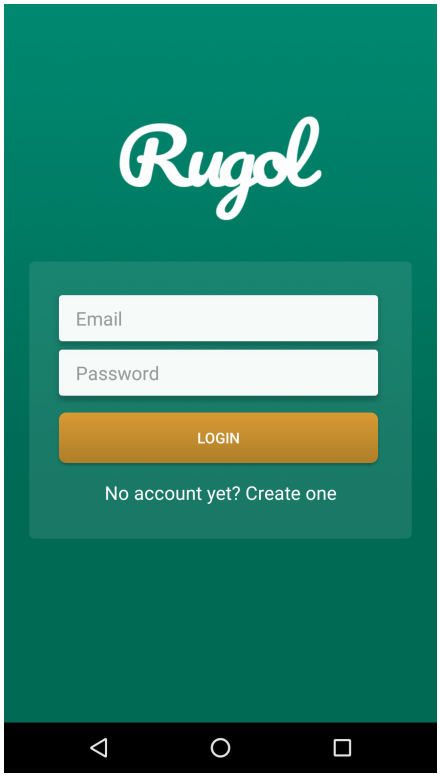

Figura 7.2: Pantalla inicio de sesión (versión final)

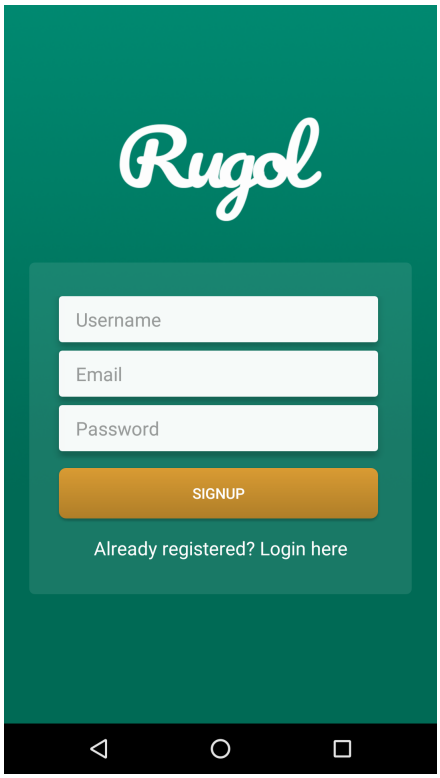

Figura 7.3: Pantalla registro (versión final)

Una vez el usuario haya iniciado sesión o se haya registrado verá la pantalla de inicio. En la parte inferior de la pantalla hay una barra de navegación por la cual el usuario podrá navegar por las distintas secciones de la aplicación.

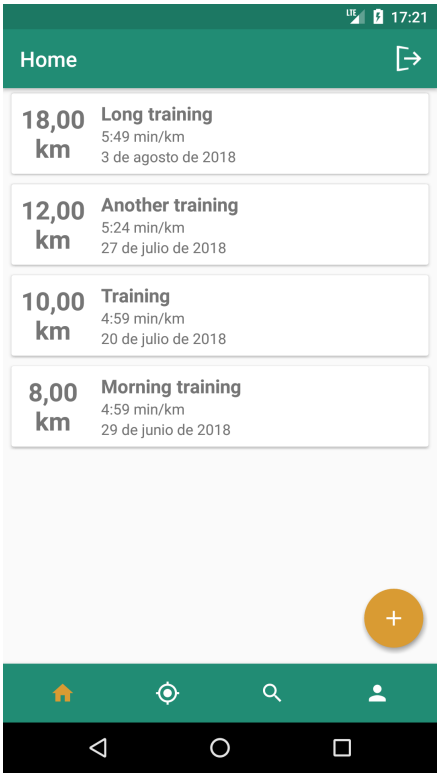

Figura 7.4: Pantalla con la lista de entrenamientos (versión final)

La primera sección contiene la lista de entrenamientos (figura 7.4) realizados por el usuario. Mediante el botón situado a la parte inferior derecha el usuario podrá crear nuevos entrenamientos o iniciar sesión en su cuenta de Strava e importarlos desde ahí.

En la segunda sección el usuario podrá gestionar sus objetivos (figura 7.5). Esta sección tiene dos partes, en la parte de arriba se encuentran los objetivos que tiene actualmente el usuario y en la parte de abajo se encuentran los objetivos que ya ha realizado. Mediante el botón de la parte inferior derecha el usuario podrá crear nuevos objetivos. Una vez el objetivo haya sido cumplido el usuario puede completarlos (manteniendo pulsado un objetivo en progreso) para así poder realizar nuevos objetivos sobre la misma distancia.

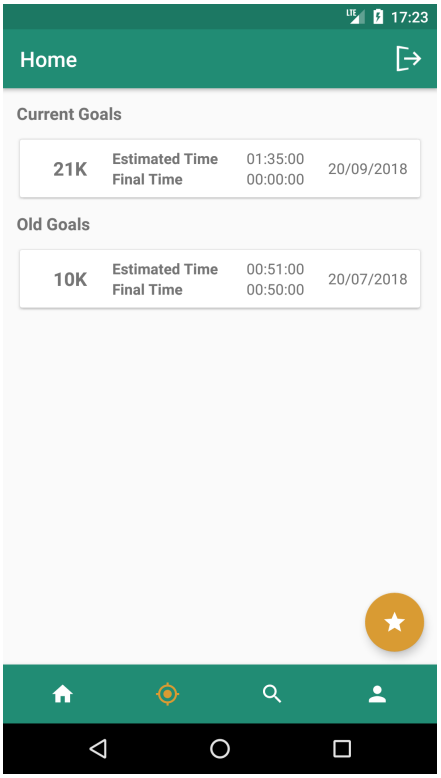

Figura 7.5: Pantalla con la lista de objetivos (versión final)

La tercera sección se trata de la pantalla de búsqueda (figura 7.6), en esta el usuario podrá rellenar un formulario con los datos que desea que cumplan los corredores que quiera buscar. Una vez realizada la búsqueda el usuario podrá ver una lista de usuarios los cuales cumplen las condiciones, podrá ver cada uno de los perfiles de las personas (figura 7.7) y realizar una comparación (entre el usuario y la persona que esta observando) mediante gráficas de de los entrenamientos realizados y distancia de cada mes en el ultimo año (figura 7.8).

Por último se encuentra la sección del perfil de usuario (figura 7.9), en esta sección el usuario podrá ver datos sobre los entrenamientos que ha realizado.

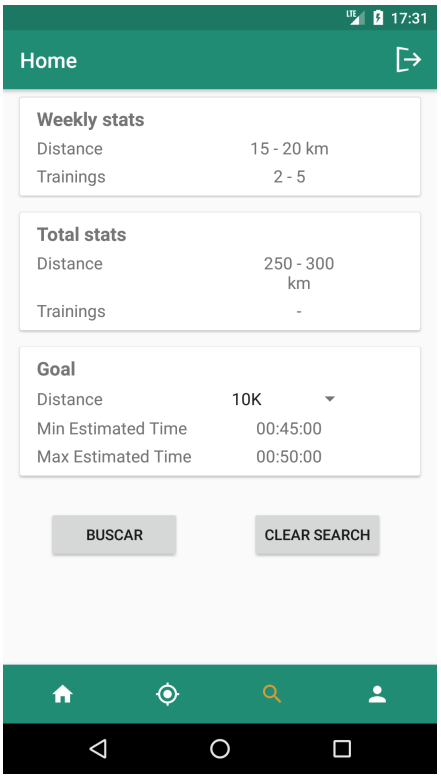

Figura 7.6: Pantalla para buscar usuarios (versión final)

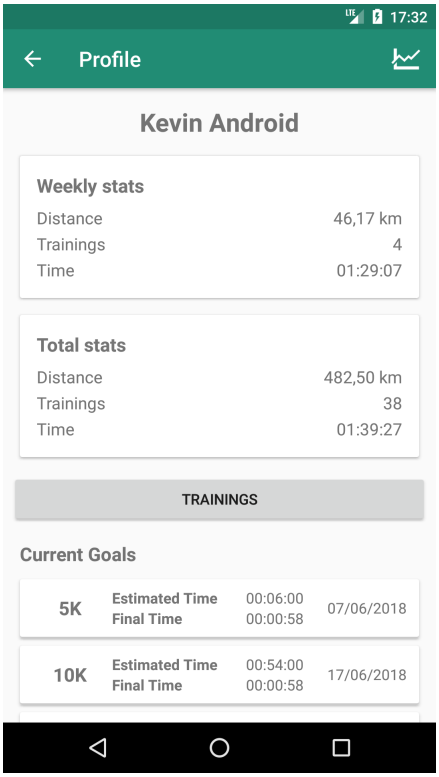

Figura 7.7: Pantalla con el perfil de un resultado de la búsqueda (versión final)

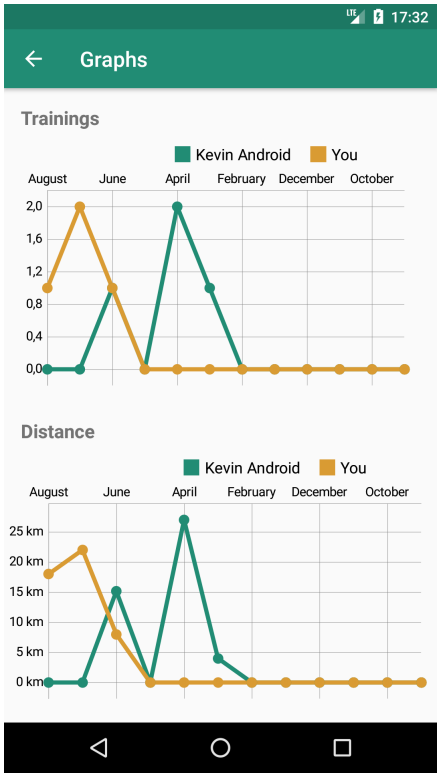

Figura 7.8: Pantalla con gráficas comparativas entre usuarios (versión final)

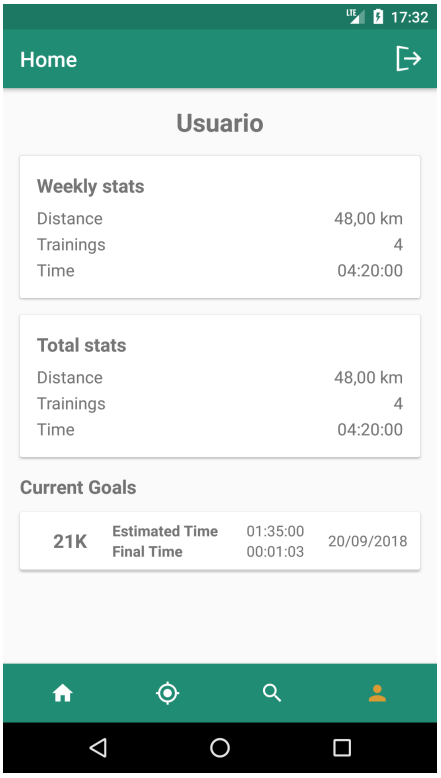

Figura 7.9: Pantalla con el perfil del usuario (versión final)

CAPÍTULO 8
Conclusiones

Durante todo el documento se ha intentado hacer hincapié en la importancia de dos cosas, utilizar una buena metodología de trabajo y una buena arquitectura software en un proyecto.

Aunque la metodología utilizada en el proyecto no se haya seguido por completo como se ha presentado (dado que está enfocada más a equipos) se ha querido presentar dada la importancia que está recibiendo en mucho equipos de desarrollo de todo el mundo. Actualmente es muy común encontrarse a equipos que trabajan con esta metodología, ya sea scrum o kanban, en vez de metodologías más antiguas en las que el producto se desarrollaba de forma completa sin modificaciones. Como se ha mencionado en el documento esto es debido a que vivimos actualmente en un mundo muy dinámico y cualquier proyecto debe estar preparado para estos cambios.

Tanto el cambio de metodología como el uso de una nueva arquitectura son dos cosas las cuales requieren un cierto tiempo de aprendizaje y de adaptación para realizarlos correctamente. A primera vista para muchas personas y equipos de desarrollo pueden parecerles que esta perdida de tiempo no da ningún beneficio y por eso en muchos casos no se llega a realizar pero la realidad es que una vez se obtiene el conocimiento la productividad y la calidad del producto es mucho más alta una vez se supera la barrera inicial.

También hemos podido ver como no en todas las áreas se utiliza las mismas arquitecturas, cada tiene una o unas arquitecturas que más le conviene a ese tipo de proyecto. En este proyecto hemos podido ver como en la parte del servidor hecho con *Ruby on Rails* la arquitectura utilizada es el *Modelo Vista Controlador*, por otra parte en la aplicación móvil Android la arquitectura que se ha utilizado es el *Modelo Vista Presentador* aunque también existen otras arquitecturas como el *Modelo Vista Modelo de Vista*.

A la hora de utilizar cualquier tipo de arquitectura no existe una sola única forma de realizarlo y es por eso que también es necesario dedicar tiempo a conocer la mayor cantidad de herramientas posibles para poder ajustar cualquier arquitectura a nuestras necesidades, ya que podría existir la posibilidad que aplicar cierta arquitectura que no conviene a nuestro proyecto nos cause más problemas que beneficios.

CAPÍTULO 9
Relación del trabajo con los estudios cursados

Los estudios cursados tienen una gran importancia para el desarrollo de un proyecto software, en él se puede ver la implicación de muchas de las asignaturas cursadas durante los cuatro años del grado.

Desde el comienzo del grado ya podemos ver asignaturas las cuales son muy importantes como son *Introducción a la informática y a la programación* y *Programación* ya que son las que asientan la base de la programación en los estudiantes, los cuales la mayoría es la primera vez que tienen contacto con el mundo de la programación. Durante los siguientes cursos podemos observar algunas asignaturas las cuales han mostrado conceptos útiles a la hora de un desarrollo software:

- *Estructuras y datos y algoritmos*: Como manejar los datos dentro de nuestro proyectos de una forma eficiente según el caso.
- *Interfaces persona computador*: Como estructurar la interfaz de usuario de una forma adecuada.
- *Redes de computadores*: Conceptos básicos para la realización de un servidor en internet.
- *Ingeniería del software*: Conceptos básicos para la realización de cualquier tipo de proyecto software.
- *Bases de datos*: Como estructurar nuestra información mediante una base de datos relacional.

En conclusión, como podemos observar a lo largo de la carrera podemos observar diferentes asignaturas las cuales sirven de utilidad a cualquier persona que ha cursado el grado que quiera realizar un proyecto software.

CAPÍTULO 10
Trabajos futuros

Durante este trabajo hemos podido ver como se ha realizado un proyecto software el cual proporciona unas funcionalidades básicas para el usuario, existe una gran cantidad de funcionalidades nuevas (y mejoras de las actuales) las cuales proporcionarían una gran experiencia a los usuarios. Como hemos comentado durante este documento la realización de un proyecto software conlleva una gran inversión de tiempo motivo por el cual la mayoría de proyectos potenciales que podemos observar en el mercado se ha realizado por grandes equipos de desarrollo, aun así durante este trabajo he querido proporcionar lo que podría ser la base de una herramienta más amplia.

A continuación se va a detallar una lista de algunos de los trabajos que se podrían realizar para seguir mejorando el proyecto:

- Mejora de los proyectos a nivel de código. Esta mejora está presente en cualquier en todos los proyectos software ya que continuamente podemos mejorar la estructura o calidad de nuestro código. Normalmente esto es algo que no se realiza tanto como se debería dado a la limitación de tiempo.
- Mejora de los diseños de la aplicación.
- Posibilidad de grabar en detalle un entrenamiento desde la aplicación, es decir, grabar la distancia recorrida, velocidad, recorrido...
- Proporcionar una manera más detallada de ver los entrenamientos por ejemplo mostrando en un mapa la ruta recorrida durante el entrenamiento.
- Permitir que los usuarios se sigan entre ellos. De esta forma un usuario podrá ver lo que van haciendo sus amigos o gente a la que considera de interés.
- Crear un sistema de retos. A parte de los objetivos que hay dentro de a aplicación crear un sistema de retos más pequeños para el usuario para motivarle aun más a seguir entrenando.
- Crear grupos de entrenamiento. Poder ofrecer la posibilidad de crear grupos donde los usuarios puedan intercambiar entrenamientos para realizarlos de forma común o que puedan reunirse y realizarlos de forma conjunta.

Bibliografía

[1] Architecting Android...The clean way? Consultar en https://fernandocejas.com/2014/09/03/architecting-android-the-clean-way/

[2] Desarrollo ágil Consultar en https://medium.com/@LazaroIbanez/a-quick-overview-to-agile-5c87ffc9e0f2

[3] To agility and beyond: The history—and legacy—of agile development Consultar en https://techbeacon.com/agility-beyond-history%E2%80%94-legacy%E2%80%94-agile-development

[4] What is the Definition of Done (DoD) in Agile? Consultar en https://www.solutionsiq.com/resource/blog-post/what-is-the-definition-of-done-dod-in-agile/

[5] Ruby on Rails. Consultar en https://es.wikipedia.org/wiki/Ruby_on_Rails

[6] Metaprogramación. Consultar en https://es.wikipedia.org/wiki/Metaprogramación

[7] MVC, MVP and MVVM Design Pattern. Consultar en https://medium.com/@ankit.sinhal/mvc-mvp-and-mvvm-design-pattern-6e169567bbad

[8] Keeping it DRY. Consultar en https://medium.com/adventures-in-code/keep-it-dry-b9e062149447

[9] The evolution of the Repository pattern - Be aware of over abstraction. Consultar en http://hannesdorfmann.com/android/evolution-of-the-repository-pattern

[10] Model-View-Presenter: Android guidelines Consultar en https://medium.com/@cervonefrancesco/model-view-presenter-android-guidelines-94970b430ddf