UNIVERSIDAD DE MÁLAGA

E.T.S. INGENIERÍA INFORMÁTICA
UNIVERSIDAD DE MÁLAGA

GRADO EN INGENIERÍA DEL SOFTWARE

APLICACIÓN MÓVIL FÓRMULA 1 FANTASY

FORMULA 1 FANTASY MOBILE APP

Realizado por
IVÁN ROMERO ROMERO

Tutorizado por
GABRIEL JESÚS LUQUE POLO

Departamento
LENGUAJES Y CIENCIAS DE LA COMPUTACIÓN
UNIVERSIDAD DE MÁLAGA

MÁLAGA, julio de 2022

UNIVERSIDAD DE MÁLAGA

E.T.S. INGENIERÍA INFORMÁTICA
UNIVERSIDAD DE MÁLAGA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA
GRADUADO EN INGENIERÍA DEL SOFTWARE

APLICACIÓN MÓVIL FÓRMULA 1 FANTASY

FORMULA 1 FANTASY MOBILE APP

Realizado por
Iván Romero Romero

Tutorizado por
Gabriel Jesús Luque Polo

Departamento
Lenguajes y Ciencias de la Computación

UNIVERSIDAD DE MÁLAGA
MÁLAGA, JUNIO DE 2022

Fecha defensa: Julio de 2022

Resumen

Ante el creciente impacto mediático de la Fórmula 1 en los últimos meses, junto al aumento de interés entre un público cada vez más joven y digitalizado, surge la idea de crear un juego de simulación que resulte atractivo para los aficionados a esta categoría deportiva. De aquí parte la realización de este Trabajo Fin de Grado cuyo principal objetivo es la creación de una aplicación móvil que tenga el Campeonato Mundial de Fórmula 1 como temática principal.

Para realizar este proyecto, se siguen las distintas etapas del desarrollo de Software, como son Análisis, Diseño, Modelado, Implementación y Pruebas, hasta obtener un producto final que se adecúe a la propuesta original. Ésta se basa en crear un juego de tipo *fantasy*, con un sistema de ligas y puntos, donde los usuarios pueden competir entre ellos a través de la formación de equipos compuestos por pilotos y escuderías reales de la competición. Para ello, es necesario transformar los resultados reales de los grandes premios y trasladarlos a la aplicación desarrollada. Esto se consigue a través de un sistema complejo de cálculo de puntuaciones a partir de los datos en bruto de la clasificación y carrera que constituyen cada evento. Además, también se ofrece a los usuarios diversas estadísticas e información sobre cada uno de los miembros del campeonato, sumado a un sistema de valores que se puede traducir como la evaluación del rendimiento de cada piloto y escudería durante la temporada.

Palabras clave:

Android, Java, Flask, API REST, diseño de algoritmos, Fórmula 1.

Abstract

Given the growing media impact of Formula 1 in recent months, along with the increasing interest of an increasingly young and digitalised public, the idea of creating a simulation game that is attractive to fans of this sport appears. This is the starting point for this Final Degree Project, whose main objective is to develop a mobile application with the Formula 1 World Championship as the main theme.

To carry out this project, it is necessary to follow the different stages of software development, such as Analysis, Design, Modeling, Implementation and Testing, to obtain a final product similar to the original proposal. This is based on creating a fantasy-type game, with a league and points system, where users can compete against each other by forming teams made up of real drivers and constructors from the competition. To do this, it is necessary to transform the real results of the Grand Prix and transfer them to the developed application. This is achieved through a complex scoring system, which come from the raw qualifying and race data that constitute each event. In addition, the app provides the users with different statistics and information for each championship member, and a value system that can be considered as the performance evaluation of each driver and team during the season.

Keywords:

Android, Java, Flask, REST API, algorithm design, Formula 1.

Índice

1

Introducción

En este primer capítulo de la memoria se describen los elementos que han motivado su realización, los objetivos que se querían cubrir, la metodología utilizada y la organización del resto de capítulos.

1.1. Motivación

En los últimos meses, el interés por la Fórmula 1 ha aumentado considerablemente entre los aficionados al deporte. Si bien es cierto que se trata de un espectáculo conocido mundialmente, su popularidad durante la última década ha sufrido algunos altibajos que culminaron con una pérdida de más del 40% de aficionados. Sin embargo, la combinación de una serie de factores como la irrupción de jóvenes talentos, el regreso a la competición de algunas figuras conocidas, la unión con el público gracias al éxito de la serie documental "Formula 1: Drive to Survive" producida por Netflix, sumado al impacto positivo de la renovación de imagen en redes sociales, ha contribuido no sólo a que la Fórmula 1 recupere el interés de los aficionados, sino también a convertirse en un atractivo entre el público joven y de países que anteriormente no tenían tanta cercanía con este deporte [1]. En la Figura 1 se puede

observar un gráfico que ilustra su crecimiento en plataformas digitales respecto a otras competiciones deportivas de primer nivel internacional, y que confirma el impacto positivo de la marca entre un público más joven y más cercano a la tecnología.

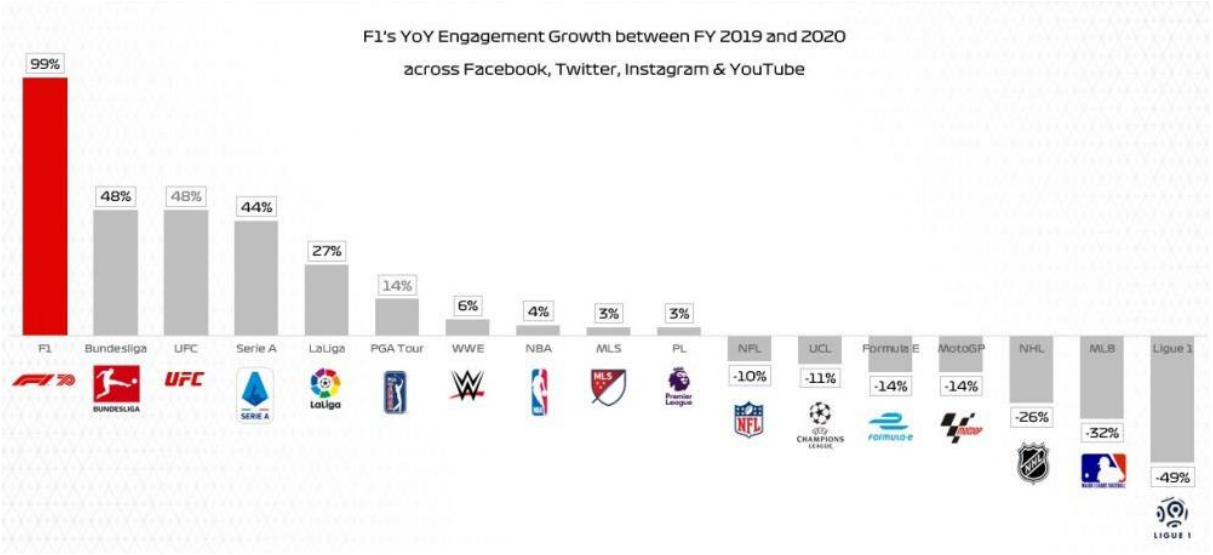

Figura 1. Crecimiento de la Fórmula 1 en plataformas sociales [2]

A su vez, el género de juego *fantasy* o simulación deportiva, donde los usuarios pueden formar un equipo virtual y competir entre ellos en función de los resultados reales de competiciones, es ya popular en otras disciplinas y alberga un gran de número de jugadores. A pesar de ello, no existe una aplicación móvil de este género dedicada exclusivamente a la Fórmula 1. Por ello, a raíz de dichas circunstancias, surge la idea de crear un juego fantasy relacionado con este deporte.

1.2. Objetivos

La finalidad de este trabajo es la creación de un juego de tipo *fantasy*, con un sistema de ligas y equipos, y cuya temática principal es el Campeonato Mundial de Fórmula 1. Por tanto, el objetivo es desarrollar una aplicación Android basada en un modelo cliente-servidor, donde la aplicación móvil actúa como interfaz de cara al usuario, mientras que el almacenamiento y tratamiento de datos se realizan en la parte del servidor. La comunicación entre estos dos componentes se lleva a cabo mediante

una API REST diseñada a medida de las funcionalidades de la aplicación. En cuanto a la aplicación, debe incluir las funcionalidades relacionadas con este género de juegos, cuya explicación se realizará de forma más detallada en el siguiente apartado. Por último, cabe destacar que además de la creación de la aplicación y de los objetivos técnicos, otra de las finalidades que se persiguen con la realización de este proyecto es afianzar conocimientos adquiridos en las distintas asignaturas del presente Grado de Ingeniería del Software, así como el aprendizaje de nuevas tecnologías y herramientas.

1.3. Contexto de la aplicación

Los juegos de deporte de tipo *fantasy* son espacios donde los usuarios compiten entre ellos, formando ligas y equipos virtuales que se acaban relacionando con elementos de competiciones reales, en este caso particular sobre Fórmula 1. Como ya se ha mencionado en la introducción, este género es popular entre los aficionados al deporte, en especial en competiciones de fútbol donde existen una gran variedad de aplicaciones que cuentan con cientos de miles de usuarios en nuestro país.

En el ámbito del motor, el número de aplicaciones se reduce respecto a otros deportes. Si bien es cierto que existen juegos con varias competiciones entre las que se encuentra la Fórmula 1, no existe hasta el momento ninguna aplicación móvil dedicada exclusivamente a dicha competición. Incluso la propia Fórmula 1 ha lanzado recientemente su propio *fantasy* en su web, aunque el problema radica en esto último, no es accesible mediante una aplicación móvil, lo que reduce drásticamente el número de jugadores que acceden al juego [3]. Es ahí donde surge la oportunidad y la idea en la que se basa este proyecto, ya que por lo general el público al que se destina este tipo de aplicaciones es gente joven y habituada a jugar desde su dispositivo móvil, como ocurren en el resto de juegos similares.

Una vez añadido el contexto donde se enmarca el proyecto, se procede a explicar de forma particular cómo funcionará el mismo. En primer lugar, los usuarios podrán crear ligas o unirse a ligas ya existentes. Cada miembro de la liga podrá formar un equipo para disputar cada evento, dicho equipo podrá ser modificado hasta el momento del inicio del evento, a partir de dicho instante quedará bloqueado. Se considera como evento la celebración de un gran premio de Fórmula 1 excluyendo entrenamientos, es decir, sólo clasificación y carrera. Cada piloto y escudería tendrá asignado un valor o precio variará en función de su rendimiento durante la temporada. Ese valor influye a la hora de formar los equipos ya que existirá un rango máximo de gasto y por tanto, un usuario no podrá elegir pilotos y escuderías cuya combinación de precios supere el valor límite establecido. Los puntos se repartirán al término de cada evento, momento en el que se actualizarán las ligas y se reabrirán los equipos.

1.4. Metodología

En este proyecto se sigue una metodología iterativa basada en Scrum y adaptada al trabajo de forma individual. Scrum es un marco de trabajo ágil que permite la colaboración entre grupos de desarrollo, y se caracteriza por el progreso incremental e iterativo, la gestión y organización del equipo y el uso de *sprints* de desarrollo. Gracias a esto, la metodología Scrum permite una gran flexibilidad ante los cambios, incentiva un ambiente de trabajo más colaborativo y proporciona un producto final de mayor calidad como resultado de la correcta aplicación de las iteraciones [4].

Tal y como se comenta anteriormente, el proceso se centra en el uso de *sprints* o iteraciones que agrupan las diferentes tareas según las funcionalidades requeridas en cada momento (ver Figura 2). En este caso, se han seleccionado las fases clásicas del desarrollo software, las cuales se van ejecutando en orden y de forma iterativa para cada conjunto de especificaciones, de forma que en cada iteración se desarrollan una selección de requisitos (por orden de prioridad) hasta completar la aplicación en su

totalidad. Como el desarrollo es individual, se han programado reuniones con el tutor cada tres semanas aproximadamente, dependiendo de los avances y de la carga de trabajo durante ese periodo, para realizar un seguimiento de las tareas realizadas.

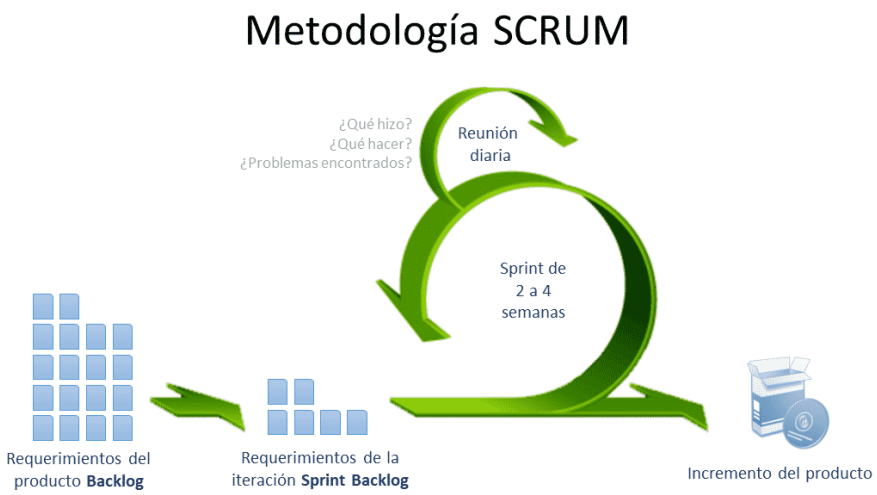

Figura 2. Metodología Scrum [5]

Entre las fases de trabajo, las etapas de Requisitos, Diseño, Implementación y Pruebas son la base de iteración siguiendo el modelo Scrum descrito previamente. Asimismo, se verán en detalle los procedimientos seguidos en cada una de estas fases a lo largo de los capítulos que componen este documento.

1.5. Estructura del documento

La presente memoria del Trabajo Fin de Grado realizado se compone de los siguientes siete capítulos que se describen a continuación:

- **Capítulo 1. Introducción.** En este primer capítulo se realiza una introducción a los contenidos que se abordan en este documento y se explica la motivación y los objetivos por los que surge el trabajo, así como el contexto de la aplicación. Además, se explica la metodología escogida para su desarrollo.

- **Capítulo 2. Tecnología y herramientas.** En esta sección se enumeran y detallan tanto las tecnologías como las diferentes herramientas utilizadas durante la elaboración de este proyecto.
- **Capítulo 3. Análisis y especificación de requisitos.** En dicho capítulo se enumeran los requisitos del sistema, en primer lugar los requisitos funcionales, y en segundo, los no funcionales. Además, se incorporan los casos de uso del sistema asociados a los requisitos funcionales descritos previamente.
- **Capítulo 4. Diseño y modelado.** Esta sección consta de la explicación de la arquitectura sobre la que se basa el sistema, el modelo de datos y sus relaciones y la descripción de los algoritmos creados para cumplir con las funcionalidades requeridas por el juego.
- **Capítulo 5. Desarrollo e implementación.** En este apartado se exponen aspectos relacionados con la etapa de implementación, tanto del desarrollo de la aplicación móvil, cómo del servidor y su despliegue.
- **Capítulo 6. Mantenimiento y pruebas.** En este capítulo se describen los procesos y tipos de pruebas escogidos para verificar el correcto funcionamiento de los elementos del sistema. Además, de una explicación de cómo debe mantenerse el mismo para su correcto funcionamiento.
- **Capítulo 7. Conclusiones y líneas futuras.** En el último apartado se exponen los resultados obtenidos tras la realización del proyecto, así como el aprendizaje fruto del desarrollo del mismo y las dificultades encontradas. Por último, se incluyen posibles mejoras y líneas futuras con las que continuar el trabajo realizado.

Además de los capítulos enumerados anteriormente, en este documento también se incluye un resumen de la propia memoria, un índice de contenidos, las referencias bibliográficas y los apéndices correspondientes a diferentes capítulos.

2

Tecnologías y Herramientas

A continuación se procede a describir las diferentes tecnologías y herramientas que se han utilizado para elaborar todos los elementos que componen el actual proyecto.

2.1. Tecnologías

2.1.1. Python

Python es un lenguaje de programación de alto nivel que apareció en la época de los noventa. Se trata de un lenguaje orientado a objetos que también soporta programación imperativa y en casos concretos, también funcional. Python es a día de hoy uno de los lenguajes más populares y demandados del mundo y cubre un gran sector de la industria en diferentes campos como Desarrollo Web, Ciencia de Datos, Inteligencia Artificial y un largo etcétera [6].

En este trabajo, se usa para implementar la parte del backend del sistema y su elección viene determinada por la gran cantidad de utilidades que incorpora, desde la generación de API REST hasta la gestión de bases de datos.

2.1.2. Flask

Figura 3. Logo Flask

Flask es un framework de Python para el desarrollo ágil y sencillo de aplicaciones web siguiendo el MVC (Modelo, Vista, Controlador), uno de los patrones clásicos del desarrollo software [7]. En el caso de este proyecto, se usa Flask en la parte del servidor para el intercambio de peticiones que se produce entre el propio servidor y los clientes, y se antepone a otros frameworks similares como Django debido a que es más útil y simple para las funcionalidades que se precisan en este sistema.

2.1.3. MongoDB Atlas y PyMongo


Figura 4. Logo MongoDB - Python

MongoDB es un sistema de base de datos de código abierto, orientado a documentos y por tanto, NoSQL. En MongoDB, los documentos que recogen los datos tienen un estilo similar a JSON.

Por su parte, MongoDB Atlas es la versión que ofrece un servicio global alojado en la nube, facilitando la conexión, la seguridad y la escalabilidad del sistema a través de la creación de clústeres de bases de datos. Es lo que actualmente se conoce como Database as a Services (DaaS) y es el motivo principal de la elección de este gestor de base de datos [8].

PyMongo es la biblioteca que permite conectar MongoDB con una aplicación desarrollada en Python. En el caso particular de este proyecto, es fundamental para el intercambio de datos entre el servidor y el clúster de almacenamiento alojado en Atlas.

2.1.4. Heroku

Heroku es una plataforma de servicios en la nube (PaaS, Platform as a Service) que se utiliza para alojar servidores y sus configuraciones, escalamiento y la administración del mismo. Se trata de uno de los entornos de despliegue más utilizado en los últimos años gracias a que posee versión gratuita, cuenta con contenedores Dynos, soporta una gran variedad de lenguajes como Node, Ruby, Java, Python, PHP… [9].

Heroku cuenta con varias formas de desplegar el servidor, una de ellas a través de Heroku CLI. La interfaz de línea de comandos (CLI) de Heroku permite crear y administrar aplicaciones directamente desde el terminal. Es una parte esencial del uso de esta plataforma y ha sido una de las formas de despliegue del software desarrollado en este proyecto, aunque como se ve a continuación no ha sido la única.

Figura 5. Logo Heroku

Además de Heroku CLI, Heroku ofrece diversas opciones para desplegar aplicaciones. Entre ellas se encuentra la posibilidad de sincronizar Heroku con un repositorio de GitHub, otra conocida herramienta que se describe más adelante en este mismo capítulo. De esta manera, la aplicación que se aloja en dicho repositorio se despliega de forma rápida y sencilla sin necesidad de instalaciones extras. La versatilidad que ofrece para realizar despliegues, sumado a la disponibilidad para crear servicios gratuitos, ha propiciado su elección para alojar el servidor de este proyecto.

2.1.5. Java / Android

Al igual que Python, Java es otro de los lenguajes de programación más extendidos y populares de la actualidad. Se trata de un lenguaje orientado a objetos, multiplataforma y muy versátil [10]. En relación a esta última característica, Java es uno de los lenguajes usado para el desarrollo de aplicaciones Android, y ese es el motivo de su inclusión en este proyecto.

Figura 6. Logo Android - Java

Android, por su parte, es un sistema operativo orientado a dispositivos móviles basado en el núcleo Linux. Fue diseñado especialmente para dispositivos móviles con pantalla táctil y a día de hoy es, con diferencia, el sistema operativo más utilizado en los teléfonos móviles. Esta es la razón principal por la que se ha elegido Android por encima de iOS u otros competidores, por su capacidad para llevar la aplicación a los usuarios finales [11].

2.1.6. Volley

Volley es una biblioteca HTTP que permite realizar peticiones en Android. Es una tecnología que se integra de forma sencilla con cualquier protocolo y que es compatible con diferente formatos de datos, como textos sin procesar, imágenes y JSON, siendo este último el formato utilizado para el intercambio de información entre el cliente y el servidor de la aplicación desarrollada. Se ha tomado la decisión de incluir Volley en la aplicación debido a su facilidad de implementación y la documentación presente en la web que facilita su uso [12].

2.2. Herramientas

Una vez descritas qué tecnologías se utilizan en este trabajo, se pasa a describir otras herramientas auxiliares que son esenciales, tales como entornos de desarrollo,

elementos para el control de versiones, y especialmente el API que se utiliza para obtener los datos con los que trabaja el sistema.

2.2.1. Visual Studio Code

Visual Studio Code es una aplicación para editar código fuente que fue lanzada a finales del año 2015 por Microsoft. Esta herramienta está disponible en Windows, Mac y Linux, e incluye características como depuración, sincronización con Git, control de sintaxis, refactorización de código, etc [13].

La compatibilidad de Visual Studio Code es alta, ya que soporta numerosos lenguajes de programación tales como C++, C#, .NET, Visual Basic, F#, Java, Python, Ruby y PHP. Esta variedad y su simplicidad de uso lo han llevado a convertirse en uno de los editores más utilizados en la actualidad, de hecho aún continúa en desarrollo y todo apunta a que se incrementarán sus funcionalidades y a que su popularidad seguirá en auge.

Figura 7. Logo VS Code

2.2.2. Android Studio

Android Studio es un IDE (Entorno de Desarrollo Integrado) basado en ItelliJ IDEA, que es también otro entorno de programación centrado en lenguaje Java. Android Studio es, de forma oficial, el entorno de desarrollo para aplicaciones Android. Consta de un editor de código y de las clásicas herramientas para desarrolladores que se incluyen de manera habitual en el resto de entornos, pero además de eso, cuenta con un sistema de compilación basado en Gradle, un emulador para simular las aplicaciones desarrolladas, integración con GitHub, herramienta que verá a continuación, herramientas de prueba y compatibilidad con Google Cloud Platform. Gracias a estas características, Android Studio se ha consolidado con firmeza como la herramienta dominadora en su sector y la referencia para sus competidores [14].

2.2.3. GitHub

GitHub es un servicio creado para alojar código de aplicaciones desarrolladas por usuarios o empresas. Consta de herramientas de administración de proyectos, incluyendo un potente sistema de control de versiones, Git, el cual permite recuperar versiones antiguas del proyecto, trabajar con varias ramas de desarrollo simultaneas e integración automática de cambios. De esta forma, Github se ha convertido en el portal principal de gestión y administración para Git, ya que también posibilita la colaboración entre desarrolladores, permitiendo la creación tanto de repositorios públicos como privados, e incluye herramientas útiles para los usuarios, como por ejemplo una herramienta de revisión de código, otra de anotaciones y comentarios en ficheros, e incluso, una Wiki personalizada para cada proyecto [15].

Figura 8. Logo GitHub

2.2.4. Postman

Postman es una herramienta multitarea que surgió como una extensión para Google Chrome pero que a día de hoy se ha convertido en una plataforma en sí misma con numerosas funcionalidades para sus usuarios. Entre las múltiples tareas que se pueden realizar en Postman encontramos el testeo de APIs, que es la característica por la que se incluye en el presente proyecto. Pero demás de las pruebas, Postman también permite gestionar la API, realizando un monitoreo y mantenimiento, y también es capaz de generar documentación útil sobre ella [16]. Todo ello basado en los conocidos métodos de intercambio de peticiones HTTP como son GET, POST, PUT, DELETE y PATCH.


Figura 9. Logo Postman

2.2.5. Ergast Developer API

Esgarst es una API abierta que ofrece un registro histórico de datos sobre Fórmula 1 para fines no comerciales. Además de la información en bruto que ofrece, cuenta también con documentación propia con referencias y ejemplos para realizar peticiones. Los datos se proporcionan en formato JSON, lo cual facilita la integración con la aplicación desarrollada [17]. Por tanto, pese a ser el recurso menos conocido de todos los nombrados anteriormente, Ergast Developer API es sin duda una herramienta imprescindible para el desarrollo de este trabajo, ya que su completitud destaca entre su escasa competencia, que o bien no ofrecen datos suficientes, o bien sólo disponen de versiones de pago.

3

Análisis y especificación de requisitos

En este capítulo se describen todos los elementos relacionados con el análisis de la aplicación que se pretende desarrollar. Se indican qué requisitos debe cumplir el sistema y qué casos de uso se han identificado.

3.1. Requisitos funcionales

En este apartado se describen y clasifican los requisitos funcionales fruto del análisis de las características que debe contener el sistema.

3.1.1. Usuario

- RF-USUARIO-1. Un usuario puede crear una cuenta en la aplicación introduciendo sus datos.

- RF-USUARIO-2. Un usuario puede iniciar sesión mediante nombre de usuario y contraseña.
- RF-USUARIO-3. Un usuario puede cerrar una sesión previamente iniciada.
- RF-USUARIO-4. Un usuario puede modificar los datos de su cuenta.

3.1.2. Liga

- RF-LIGA-1. Un jugador puede participar en una liga.
    - RF-LIGA-1.1. Un jugador puede crear una nueva liga.
    - RF-LIGA-1.2. Un jugador puede unirse a una liga ya existente, a través de un código de acceso.
- RF-LIGA-2. Un jugador puede ver el listado de ligas en las que participa.
- RF-LIGA-3. Un jugador puede consultar la clasificación de sus respectivas ligas.
- RF-LIGA-4. Los jugadores reciben las puntuaciones correspondientes a su equipo al término del evento.
- RF-LIGA-5. La clasificación se actualiza atendiendo a los datos y resultados reales de la competición, obtenidos de la API.
- RF-LIGA-6. Un jugador puede consultar el historial de puntos de cada participante en las ligas en las que forma parte.

3.1.3. Equipo

- RF-EQUIPO-1. Un jugador puede ver la alineación de su equipo.
- RF-EQUIPO-2. Un jugador puede alinear y modificar su equipo para disputar un evento.
    - RF-EQUIPO-2.1. Un jugador puede modificar su equipo seleccionando o deseleccionando elementos siempre que no supere el saldo máximo límite.
    - RF-EQUIPO-2.2. El sistema modificará el valor de los elementos de acuerdo a las puntuaciones obtenidas a lo largo de la temporada.

    - RF-EQUIPO-2.3. Un jugador puede alinear hasta cinco pilotos y una escudería.
    - RF-EQUIPO-2.4. Un jugador puede obtener puntos aunque su equipo no esté completo, siempre que este tenga al menos un elemento.
- RF-EQUIPO-3. Un jugador puede formar un equipo diferente en cada una de sus ligas.
- RF-EQUIPO-4. Los equipos permanecen bloqueados durante la disputa real del evento.
- RF-EQUIPO-5. Los equipos se liberan tras la celebración del evento y la correspondiente asignación de puntos y cambios de valores.

3.1.4. Evento

- RF-EVENTO-1. Un usuario puede consultar el calendario de eventos de la temporada.
    - RF-EVENTO-1.1. Un usuario puede conocer qué evento se está disputando en el momento actual.
- RF-EVENTO-2. Un usuario puede ver las puntuaciones obtenidas por los pilotos y escuderías en los eventos ya celebrados.
- RF-EVENTO-3. Un usuario puede visualizar la información principal de cada evento.
- RF-EVENTO-4. El sistema muestra datos e información personalizada al usuario sobre cada evento.

3.1.5. Componente

- RF-COMPONENTE-1. Un usuario puede consultar el listado de pilotos y de escuderías.
- RF-COMPONENTE-2. Un usuario puede ver las puntuaciones obtenidas por cada piloto y escudería en los distintos eventos en los que han participado.

- RF-COMPONENTE-3. Un usuario puede visualizar las estadísticas de cada piloto y escudería obtenidas en el juego.
- RF-COMPONENTE-4. El sistema muestra datos e información personalizada al usuario sobre cada piloto y sobre cada escudería.
- RF-COMPONENTE-5. Cada piloto y escudería modifica su valor en función del rendimiento obtenido en la celebración de cada gran premio.

3.2. Requisitos no funcionales

A continuación se muestra el listado de los requisitos no funcionales, los cuales se agrupan según el tipo de requisito.

3.2.1. Diseño

- RNF-DISEÑO-1. El sistema debe estar modularizado separando claramente el backend, frontend y el almacenamiento de datos.
- RNF-DISEÑO-2. La API abierta desde donde se obtiene la información y resultados necesarios será Ergast.
- RNF-DISEÑO-3. El usuario debe ser capaz de acceder al sistema usando un dispositivo Android.

3.2.2. Disponibilidad

- RNF-DISPONIBILIDAD-1. Los cambios realizados por los usuarios deben actualizarse en tiempo real.
- RNF-DISPONIBILIDAD-2. Las clasificaciones de las ligas, equipos y demás elementos del juego deben estar siempre disponibles para el usuario.

3.2.3. Implementación

- RNF-IMPLEMENTACIÓN-1. El sistema usará datos reales de escuderías y pilotos, y dichos datos deben actualizarse durante la temporada.

- RNF-IMPLEMENTACIÓN-1.1. Los puntos serán otorgados en base a los resultados de la clasificación y la carrera disputada en cada evento.
- RNF-IMPLEMENTACIÓN-1.2. El valor de mercado se modificará y ajustará al rendimiento de cada piloto y escudería, teniendo en cuenta tanto el último evento, como las estadísticas generales y otros factores de la temporada.

3.2.4. Usabilidad

- RNF-USABILIDAD-1. El sistema es robusto a comportamiento inesperados o erróneos del usuario, mostrándole la información oportuna.
- RNF-USABILIDAD-2. El sistema contará con internacionalización para visualizar la aplicación tanto en español como en inglés.

3.2.5. Seguridad

- RNF-SEGURIDAD-1. El sistema debe velar por la seguridad y privacidad de los datos de los usuarios.
  - RNF-SEGURIDAD-1.1. No se mostrarán datos asociados a otros usuarios, salvo el nombre completo de los participantes de las ligas en común.
  - RNF-SEGURIDAD-1.2. Las contraseñas se almacenarán de forma encriptada en la base de datos.

3.3. Casos de uso

Título	Creación de cuenta.
Actor	Usuario.
Descripción	Un usuario crea una cuenta en la aplicación introduciendo sus datos.
Pre-Condición	El usuario debe tener instalada la aplicación en un dispositivo Android.

Post-Condición	El usuario dispondrá de una nueva cuenta.
Requisito	RF-USUARIO-1
Escenario principal (Secuencia)	
1. El sistema muestra la pantalla de carga y la pantalla de inicio de sesión. 2. El usuario pulsa en Crear cuenta. 3. El sistema redirige al usuario a la pantalla con el formulario de creación de cuenta. 4. El usuario introduce sus datos. 5. El usuario pulsa en el botón Crear cuenta. 6. El sistema verifica los datos introducidos. 7. El sistema crea la cuenta a partir de los datos. 8. El sistema muestra un mensaje de confirmación. 9. El sistema redirige de nuevo al usuario a la pantalla de inicio de sesión.	
Escenario alternativo (Secuencia alternativa)	
7. El sistema detecta un error en los datos introducidos. 8. El sistema muestra un mensaje de error al usuario e indica el campo concreto donde se ha producido la incidencia. 9. El sistema conserva los datos introducidos y permanece en la misma pantalla.	

Tabla 1. Caso de uso. Creación de Cuenta

Título	Inicio de sesión.
Actor	Usuario.
Descripción	Un usuario inicia sesión con su nombre de usuario y contraseña.
Pre-Condición	El usuario debe abrir la aplicación en un dispositivo Android.
Post-Condición	El usuario accederá a su cuenta.
Requisito	RF-USUARIO-2
Escenario principal (Secuencia)	
1. El usuario abre la aplicación en su dispositivo Android. 2. El sistema muestra la pantalla de carga y la pantalla de inicio de sesión. 3. El usuario introduce su usuario y contraseña. 4. El usuario pulsa en el botón Iniciar sesión. 5. El sistema verifica los datos introducidos. 6. El sistema redirige al usuario a la pantalla con el menú principal.	

Escenario alternativo (Secuencia alternativa)
6. El sistema detecta un error en los datos introducidos. 7. El sistema muestra un mensaje de error al usuario e indica el campo concreto donde se ha producido la incidencia. 8. El sistema conserva los datos introducidos y permanece en la misma pantalla.

Tabla 2. Caso de uso. Inicio de sesión

Título	Cierre de sesión.
Actor	Usuario.
Descripción	Un usuario cierra su sesión previamente iniciada.
Pre-Condición	El usuario debe tener iniciada su sesión.
Post-Condición	La sesión del usuario se cerrará.
Requisito	RF-USUARIO-3
Escenario principal (Secuencia)	
1. El usuario se coloca en cualquiera de las pantallas de la aplicación. 2. El usuario pulsa en el icono de cerrar sesión en la barra superior. 3. El sistema cierra la sesión del usuario. 4. El sistema muestra un mensaje de confirmación. 5. El sistema redirige al usuario a la pantalla de inicio de sesión.	

Tabla 3. Caso de uso. Cierre de sesión

Título	Modificación de datos del usuario.
Actor	Usuario.
Descripción	Un usuario modifica los datos de su cuenta.
Pre-Condición	El usuario debe encontrarse en la pantalla del menú principal.
Post-Condición	Los datos del usuario serán modificados.
Requisito	RF-USUARIO-4
Escenario principal (Secuencia)	
1. El usuario pulsa en el icono de su cuenta en la barra superior. 2. El sistema muestra la pantalla con los datos de su cuenta. 3. El usuario modifica sus datos. 4. El usuario pulsa en el botón Actualizar. 5. El sistema verifica los datos introducidos.	

6. El sistema actualiza el perfil del usuario. 7. El sistema muestra un mensaje de confirmación. 8. El sistema redirige de nuevo al usuario a la pantalla de menú.
Escenario alternativo (Secuencia alternativa)
6. El sistema detecta un error en los datos introducidos. 7. El sistema muestra un mensaje de error al usuario e indica el campo concreto donde se ha producido la incidencia. 8. El sistema conserva los datos introducidos y permanece en la misma pantalla.

Tabla 4. Caso de uso. Modificación de datos del usuario

Título	Creación de liga.
Actor	Usuario.
Descripción	Un usuario crea una nueva liga.
Pre-Condición	El usuario debe haberse conectado de forma exitosa al sistema y encontrarse en la pantalla de ligas.
Post-Condición	El usuario creará la liga.
Requisito	RF-LIGA-1.1
Escenario principal (Secuencia)	
1. El usuario pulsa en el botón Añadir liga. 2. El sistema redirige al usuario a la pantalla de añadir liga. 3. El usuario introduce los datos de la nueva liga. 4. El usuario pulsa en el botón Crear liga. 5. El sistema verifica los datos introducidos. 6. El sistema crea una nueva liga. 7. El sistema muestra un mensaje de confirmación. 8. El sistema redirige de nuevo al usuario a la pantalla de ligas.	
Escenario alternativo (Secuencia alternativa)	
5. El sistema detecta un error en los datos introducidos. 6. El sistema muestra un mensaje de error al usuario. 7. El sistema conserva los datos introducidos y permanece en la misma pantalla.	

Tabla 5. Caso de uso. Creación de liga

Título	Unión a liga.
Actor	Usuario.
Descripción	Un usuario se une a una liga ya existente.
Pre-Condición	El usuario debe haberse conectado de forma exitosa al sistema y encontrarse en la pantalla de ligas.
Post-Condición	El usuario se unirá a la liga.
Requisito	RF-LIGA-1.2
Escenario principal (Secuencia)	
1. El usuario pulsa en el botón Añadir liga. 2. El sistema redirige al usuario a la pantalla de añadir liga. 3. El usuario introduce el código de la liga. 4. El usuario pulsa en el botón Unirme. 5. El sistema verifica los datos introducidos. 6. El sistema crea añade al usuario como participante de la liga. 7. El sistema muestra un mensaje de confirmación. 8. El sistema redirige de nuevo al usuario a la pantalla de ligas.	
Escenario alternativo (Secuencia alternativa)	
5. El sistema detecta un error en el código introducido. 6. El sistema muestra un mensaje de error al usuario. 7. El sistema conserva los datos introducidos y permanece en la misma pantalla.	

Tabla 6. Caso de uso. Unión a liga

Título	Listado de ligas.
Actor	Usuario.
Descripción	El sistema muestra el listado de ligas en las que participa el usuario.
Pre-Condición	El usuario debe haberse conectado de forma exitosa al sistema y encontrarse en la pantalla de menú.
Post-Condición	El usuario observará el listado.
Requisito	RF-LIGA-2
Escenario principal (Secuencia)	
1. El usuario pulsa en el botón Ligas en el menú principal. 2. El sistema redirige al usuario a la pantalla Ligas. 3. El sistema despliega un listado con las ligas en las que participa el usuario.	

Escenario alternativo (Secuencia alternativa)
3. El sistema no muestra nada si el usuario no pertenece a ninguna liga.

Tabla 7. Caso de uso. Listado de ligas

Título	Clasificación de liga.
Actor	Usuario.
Descripción	El usuario consulta la clasificación de una liga en la que participa.
Pre-Condición	El usuario debe haberse conectado de forma exitosa al sistema y encontrarse en la pantalla de ligas.
Post-Condición	El usuario observará la clasificación.
Requisito	RF-LIGA-3
Escenario principal (Secuencia)	
1. El usuario pulsa sobre una de las ligas que aparecen en el listado. 2. El sistema redirige al usuario a la pantalla de la liga seleccionada. 3. El sistema muestra un listado con los participantes de la liga y sus respectivas puntuaciones.	

Tabla 8. Caso de uso. Clasificación de liga

Título	Historial de puntos de participante.
Actor	Usuario.
Descripción	El usuario consulta el historial de puntos de un participante en una de las ligas.
Pre-Condición	El usuario debe haberse conectado de forma exitosa al sistema y encontrarse en la pantalla de ligas.
Post-Condición	El usuario observará el historial.
Requisito	RF-LIGA-6
Escenario principal (Secuencia)	
1. El usuario pulsa sobre una de las ligas en las que participa. 2. El sistema redirige al usuario a la pantalla de la liga seleccionada. 3. El sistema muestra un listado con los participantes de la liga y sus respectivas puntuaciones totales. 4. El usuario pulsa sobre alguno de los participantes de la liga. 5. El sistema redirige al usuario a la pantalla de puntos.	

6. El sistema muestra un listado con las rondas disputadas hasta el momento y la puntuación del participante seleccionado en cada uno de esos eventos.

Tabla 9. Caso de uso. Historial de puntos del participante

Título	Equipo.
Actor	Usuario.
Descripción	El usuario puede ver la alineación de su equipo en una liga.
Pre-Condición	El usuario debe haberse conectado de forma exitosa al sistema y encontrarse en la pantalla de ligas.
Post-Condición	El usuario observará su equipo.
Requisito	RF-EQUIPO-1, RF-EQUIPO-3
Escenario principal (Secuencia)	
1. El usuario pulsa sobre una de las ligas en las que participa. 2. El sistema redirige al usuario a la pantalla de la liga seleccionada. 3. El sistema muestra un listado con los participantes de la liga y sus respectivas puntuaciones totales. 4. El usuario pulsa sobre el botón Mi equipo. 5. El sistema redirige al usuario a la pantalla de equipo. 6. El sistema muestra la alineación del equipo del usuario para esa liga seleccionada y el valor del mismo.	

Tabla 10. Caso de uso. Equipo

Título	Fichaje miembro de equipo.
Actor	Usuario.
Descripción	El usuario puede alinear y modificar el equipo.
Pre-Condición	El usuario debe haberse conectado de forma exitosa al sistema y encontrarse en una de sus ligas.
Post-Condición	El usuario fichará un nuevo miembro.
Requisito	RF-EQUIPO-2, RF-EQUIPO-4
Escenario principal (Secuencia)	
1. El usuario pulsa sobre el botón Mi equipo. 2. El sistema redirige al usuario a la pantalla de equipo. 3. El sistema muestra la alineación del equipo del usuario para esa liga seleccionada y el valor del mismo. 4. El usuario pulsa en un hueco libre de su plantilla.	

5. El sistema redirige al usuario a la pantalla de pilotos o escuderías. 6. El usuario pulsa en el botón Fichar sobre el piloto o escudería que desea alinear. 7. El sistema verifica el fichaje. 8. El sistema añade el nuevo piloto o escudería al equipo y actualiza su valor. 9. El sistema redirige al usuario a la pantalla de su equipo.
Escenario alternativo (Secuencia alternativa)
7. El sistema detecta que el equipo está completo o que se supera el límite de dinero. 8. El sistema muestra un mensaje informativo al usuario. 9. El sistema redirige al usuario a la pantalla de su equipo.
Escenario alternativo 2 (Secuencia alternativa 2)
7. El sistema muestra bloqueados los miembros del equipo durante la disputa de un gran premio y no se puede fichar.

Tabla 11. Caso de uso. Fichaje miembro de equipo

Título	Venta miembro de equipo.
Actor	Usuario.
Descripción	El usuario puede vender y modificar el equipo.
Pre-Condición	El usuario debe haberse conectado de forma exitosa al sistema y encontrarse en una de sus ligas.
Post-Condición	El usuario venderá un miembro de su equipo.
Requisito	RF-EQUIPO-2, RF-EQUIPO-4
Escenario principal (Secuencia)	
1. El usuario pulsa sobre el botón Mi equipo. 2. El sistema redirige al usuario a la pantalla de equipo. 3. El sistema muestra la alineación del equipo del usuario para esa liga seleccionada y el valor del mismo. 4. El usuario pulsa en el botón Vender de alguno de los miembros de su equipo. 5. El sistema elimina el piloto o escudería del equipo del usuario. 6. El sistema actualiza los componentes del equipo y su valor.	
Escenario alternativo (Secuencia alternativa)	
4. El sistema muestra un equipo vacío y por tanto no deja vender.	
Escenario alternativo 2 (Secuencia alternativa 2)	

4. El sistema muestra bloqueados los miembros del equipo durante la disputa de un gran premio y no se puede vender.

Tabla 12. Caso de uso. Venta miembro de equipo

Título	Calendario de eventos.
Actor	Usuario.
Descripción	El usuario puede ver el calendario de grandes premios.
Pre-Condición	El usuario debe haberse conectado de forma exitosa al sistema y encontrarse en el menú principal.
Post-Condición	El usuario verá el calendario.
Requisito	RF-EVENTO-1
Escenario principal (Secuencia)	
1. El usuario pulsa en el botón Eventos en el menú principal. 2. El sistema redirige al usuario a la pantalla Eventos. 3. El sistema despliega un listado con los grandes premios de toda la temporada.	

Tabla 13. Caso de uso. Calendario de eventos

Título	Puntuaciones de eventos.
Actor	Usuario.
Descripción	El usuario puede ver los resultados de las puntuaciones de los eventos ya celebrados.
Pre-Condición	El usuario debe haberse conectado de forma exitosa al sistema y encontrarse en la pantalla de eventos.
Post-Condición	El usuario visualizará los resultados.
Requisito	RF-EVENTO-2
Escenario principal (Secuencia)	
1. El usuario pulsa sobre uno de los grandes premios del listado de eventos. 2. El sistema redirige al usuario a la pantalla de ese evento. 3. El usuario pulsa sobre el botón Puntuaciones. 4. El sistema redirige al usuario a la pantalla de puntuaciones. 5. El sistema lista los resultados del gran premio con los puntos obtenidos por cada una de las escuderías y pilotos que participaron.	
Escenario alternativo (Secuencia alternativa)	

5. El sistema no muestra resultados si el evento aún no se ha disputado.

Tabla 14. Caso de uso. Puntuaciones de eventos

Título	Visualizar evento.
Actor	Usuario.
Descripción	El usuario puede ver la información del evento y datos personalizados sobre el mismo.
Pre-Condición	El usuario debe haberse conectado de forma exitosa al sistema y encontrarse en el menú principal.
Post-Condición	El usuario visualizará la información del evento.
Requisito	RF-EVENTO-3, RF-EVENTO-4
Escenario principal (Secuencia)	
1. El usuario pulsa en el botón Eventos en el menú principal. 2. El sistema redirige al usuario a la pantalla Eventos. 3. El sistema despliega un listado con los grandes premios de toda la temporada. 4. El usuario pulsa sobre uno de los grandes premios del listado de eventos. 5. El sistema redirige al usuario a la pantalla de ese evento. 6. El sistema muestra la información acerca del gran premio, datos y estadísticas personalizadas y aleatorias sobre el evento.	

Tabla 15. Caso de uso. Visualizar evento

Título	Listado de pilotos y escuderías
Actor	Usuario.
Descripción	El usuario puede consultar el listado de pilotos y escuderías que compiten durante la temporada.
Pre-Condición	El usuario debe haberse conectado de forma exitosa al sistema y encontrarse en el menú principal.
Post-Condición	El usuario verá el listado de pilotos o escuderías.
Requisito	RF-COMPONENTE-1
Escenario principal (Secuencia)	
1. El usuario pulsa en el botón Pilotos o Escuderías en el menú principal. 2. El sistema redirige al usuario a la pantalla Pilotos o Escuderías.	

3. El sistema despliega un listado con los Pilotos o Escuderías que forman la parrilla.

Tabla 16. Caso de uso. Lisado de pilotos y escuderías

Título	Puntuaciones de pilotos y escuderías.
Actor	Usuario.
Descripción	El usuario puede ver los resultados de las puntuaciones de los pilotos y escuderías en eventos ya celebrados.
Pre-Condición	El usuario debe haberse conectado de forma exitosa al sistema y encontrarse en la pantalla de pilotos o escuderías.
Post-Condición	El usuario visualizará los resultados.
Requisito	RF-COMPONENTE-2
Escenario principal (Secuencia)	
1. El usuario pulsa sobre uno de los grandes premios del listado de componentes. 2. El sistema redirige al usuario a la pantalla de ese componente. 3. El usuario pulsa sobre el botón Puntuaciones. 4. El sistema redirige al usuario a la pantalla de puntuaciones. 5. El sistema lista los resultados del piloto o la escudería en cada uno de los eventos en los que ha participado.	
Escenario alternativo (Secuencia alternativa)	
5. El sistema no muestra resultados si el piloto o escudería no ha participado en ningún evento hasta la fecha.	

Tabla 17. Caso de uso. Puntuaciones de pilotos y escuderías

Título	Visualizar piloto y escudería.
Actor	Usuario.
Descripción	El usuario puede ver la información del piloto o escudería y datos personalizados sobre el mismo.
Pre-Condición	El usuario debe haberse conectado de forma exitosa al sistema y encontrarse en el menú principal.
Post-Condición	El usuario visualizará la información del componente.
Requisito	RF-COMPONENTE-3, RF-COMPONENTE-4
Escenario principal (Secuencia)	

1. El usuario pulsa en el botón Pilotos o Escuderías en el menú principal.
2. El sistema redirige al usuario a la pantalla Pilotos o Escuderías.
3. El sistema despliega un listado con los Pilotos o Escuderías que forman la parrilla.
4. El usuario pulsa sobre uno de los componentes del listado.
5. El sistema redirige al usuario a la pantalla de ese componente.
6. El sistema muestra la información acerca del piloto o escudería, datos y estadísticas personalizadas y aleatorias sobre el mismo.

Tabla 18. Caso de uso. Visualizar piloto y escudería

4

Diseño y modelado del sistema

Una vez detallado el análisis y la obtención de los requisitos, se procede a explicar la arquitectura y modelado del sistema, exponiendo los motivos que han llevado a tomar cada una de las decisiones de diseño.

4.1. Arquitectura de la aplicación

Tras estudiar las diferentes funcionalidades que debería tener la aplicación, se ha elegido una arquitectura cliente-servidor, debido a que se trata de un modelo de diseño que permite separar la lógica de la aplicación y la gestión de los datos, que se ejecuta en la parte servidora, de la interacción del usuario con la aplicación, que se realiza desde el lado del cliente. A su vez, esta arquitectura permite conectar varios dispositivos (clientes) de forma simultánea a los servicios ofrecidos, característica

imprescindible para poder implementar las funcionalidades que se pretenden desarrollar.

La imagen que aparece en la Figura 10 representa la arquitectura final del sistema y las relaciones entre cada componente, incluyendo la aplicación móvil (que se corresponde con el cliente), el servidor y el sistema de almacenamiento de datos, además de las tecnologías que implementan cada parte.

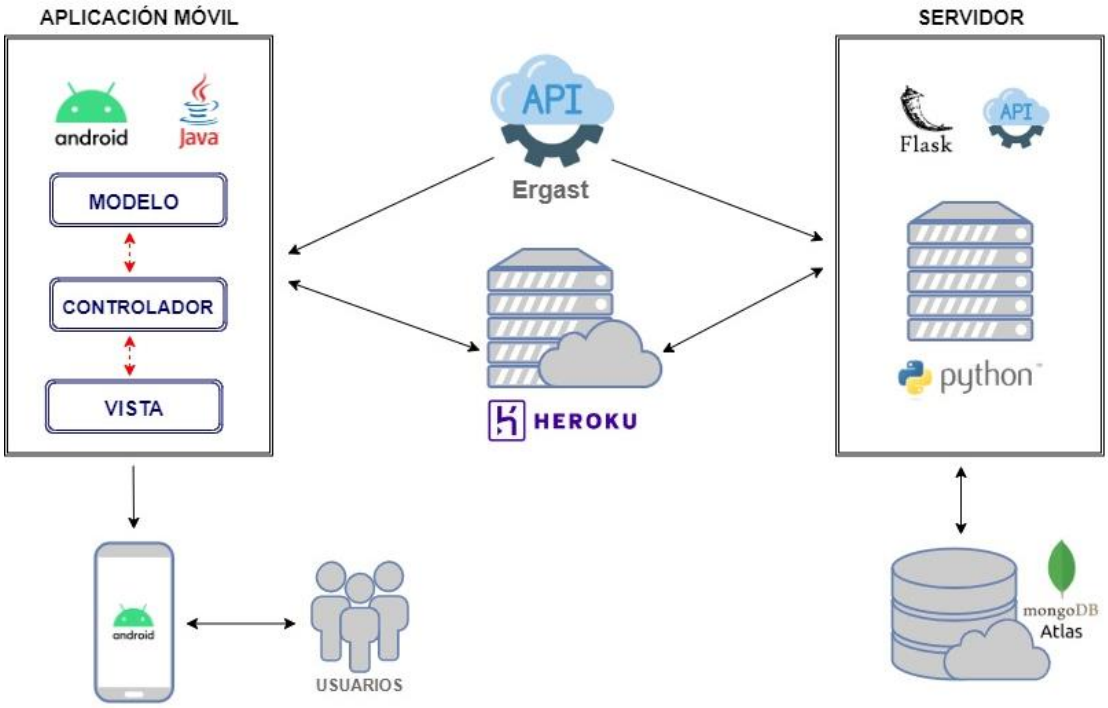

Figura 10. Diagrama de arquitectura

4.1.1. Servidor

El servidor está desarrollado en Python y contiene las principales funcionalidades lógicas de la aplicación, como son los algoritmos para el cálculo de puntos y para las modificaciones de los valores asignados a los componentes. Además, realiza la función de intermediario entre la base de datos y las aplicación móvil a través de una API REST propia. Para el desarrollo de la API se hace uso del framework Flask, que agiliza la creación de la aplicación que contiene los métodos del servicio web.

Los datos en bruto de los resultados de las competiciones los obtiene el servidor mediante consultas a la API REST de Ergast. Tras la recepción de los datos en formato JSON, la información se procesa en el propio servidor antes de actualizar resultados en base de datos.

En cuanto al alojamiento, el servidor se encuentra en la plataforma Heroku, con ello se consigue que esté activo de forma constante, y por ende, disponible para comunicarse con los distintos dispositivos de los usuarios.

4.1.2. Cliente

La parte del cliente está compuesta por una aplicación desarrollada en Android que será utilizada desde los dispositivos móviles de los usuarios. La aplicación recoge los datos tanto del propio servidor como también de la API Ergast, dependiendo de la finalidad de uso de dicha información. Por un lado, todos los datos referentes a los usuarios, ligas, puntuaciones y demás información que se alberga en la base de datos se demandan a través de los puntos de acceso que ofrece el servidor alojado en Heroku. Por otro lado, la información referente a estadísticas y datos aleatorios que se ofrecen como funcionalidad en la aplicación se recogen directamente de Ergast, debido a que su procesamiento es menor y puede realizarse directamente en el cliente sin necesidad de que el servidor actúe como intermediario, ya que esto último podría causar que se ralentice la llegada de la información hasta el usuario.

Dentro del propio cliente se utiliza otro patrón de arquitectura, el Modelo-Vista-Controlador (MVC), incluido para facilitar y formalizar el desarrollo de la aplicación móvil. El funcionamiento de este estilo de arquitectura es el que se describe a continuación: la parte del Modelo contiene la representación de los datos que se manejan en el sistema, aunque en este caso no incluye los mecanismos de persistencia del modelo de datos ya que de esa tarea se encarga el servidor. La Vista se corresponde con la interfaz de usuario, que muestra la información a los usuarios de la aplicación e

incluye la interacción con los mismos. Finalmente, el Controlador actúa como componente intermediario entre el Modelo y la Vista gestionando el traspaso de información entre ambos y ejecutando las operaciones lógicas necesarias, siendo necesario incluir en esta parte las comunicaciones y el intercambio de información con el servidor [18].

4.1.3. Almacenamiento de datos

Para finalizar con la arquitectura, se cuenta con un *cloud* en la plataforma MongoDB Atlas destinado al almacenamiento de datos de nuestro sistema. El funcionamiento de Atlas es similar al de una base de datos en MongoDB, pero con la diferencia de que las colecciones se encuentran alojadas en la nube. La decisión de combinar dos tecnologías como Flask y MongoDB surge de que en ambas partes se opera con un formato de datos similar a JSON y esto simplifica el poder trabajar con el modelo de datos sin necesidad de crear clases, mapeos complejos u otro subpatrón de arquitectura en la parte del servidor.

4.2. Modelado de datos

Para el modelo de datos se han diseñado un total de nueve tipos de componentes con sus atributos correspondientes. Este diseño se utiliza tanto en la parte del Modelo en el patrón MVC del cliente como también en las colecciones de documentos de la base de datos. Aunque el almacenamiento de datos no es relacional, existe una persistencia completa entre las tablas (colecciones) de la base de datos en Mongo y las clases Java de la aplicación en Android, respetando la totalidad de los atributos y las relaciones entre elementos.

Por otro lado, también se han diseñado los métodos necesarios para satisfacer las funcionalidades del sistema. Estos procedimientos o funciones se encuentran en su

mayoría dentro de los controladores del MVC, debido a que es en dichos componentes donde se produce la interacción entre aplicación y servidor.

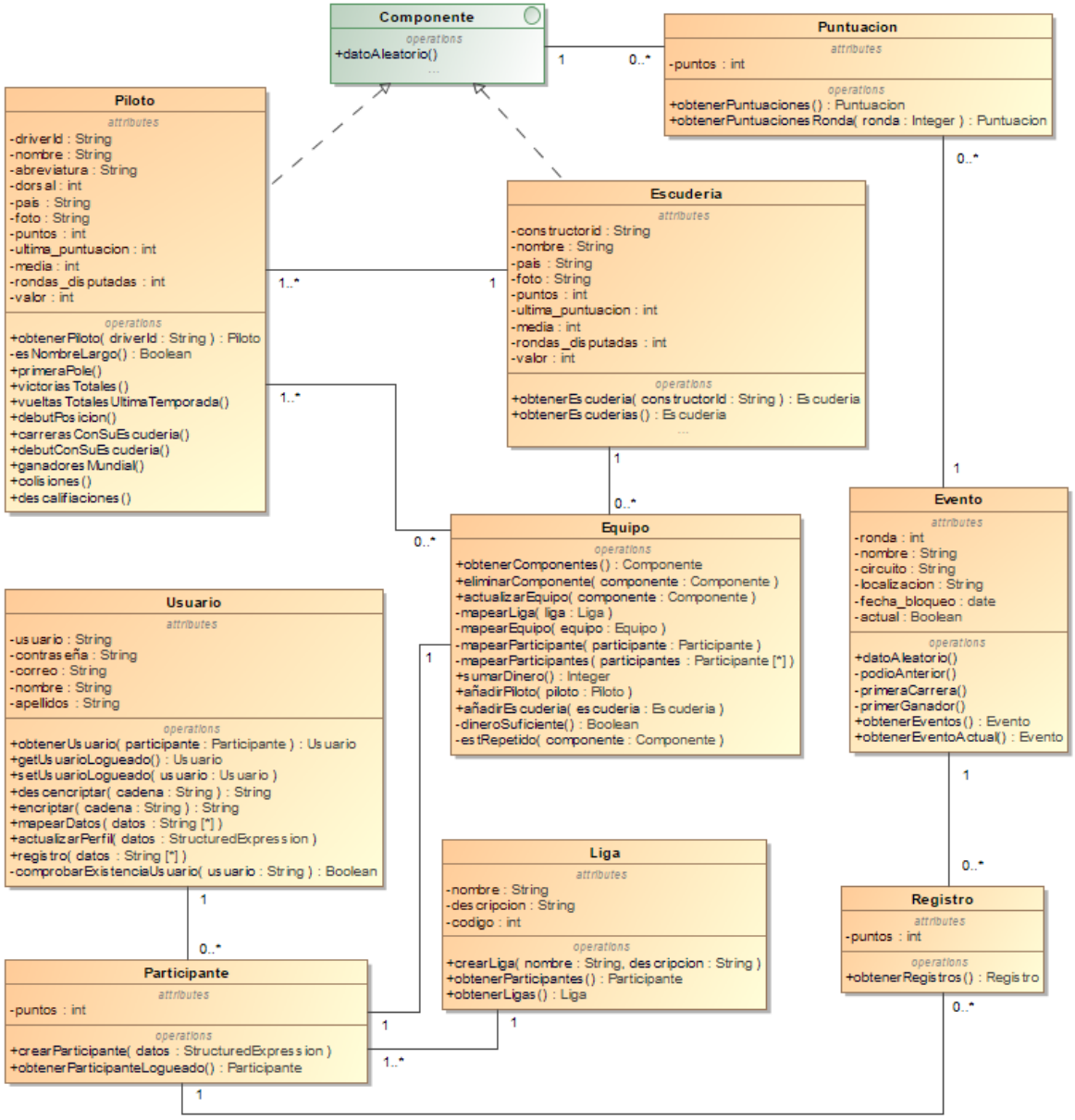

Figura 11. Diagrama de clases

Cada clase contiene los atributos y las relaciones necesarias para cumplir con el desarrollo de funcionalidades del juego. A continuación se procede a describir brevemente cada uno de los elementos del modelo de datos.

4.2.1. Usuario

Los usuarios se corresponden con los jugadores de la aplicación, los cuales pueden registrarse, iniciar sesión, participar en ligas, etc. En cada usuario se almacena la información del perfil, teniendo en cuenta siempre que la contraseña debe estar cifrada.

4.2.2. Piloto

La clase `Piloto` extiende de la interfaz `Componente` y representa a los pilotos que componen el campeonato de Fórmula 1. De cada uno de ellos se almacena la tanto la información acerca del piloto como los datos referentes al juego, los cuales se van actualizando con el paso del tiempo. Además, añaden las funciones para extraer estadísticas personalizadas para cada uno de ellos.

4.2.3. Escudería

Escudería es otro elemento similar a `Piloto` pero representando en este caso a los equipos de la competición real. Contiene atributos y métodos similares a los de la clase `Piloto`, ya que a efectos del juego ambos tienen una funcionalidad prácticamente idéntica, de ahí de la existencia de la interfaz `Componente` para que se puedan referenciar ambas clases a la vez cuando sea necesario. Para simplificar un poco el diagrama de clases se han obviado algunos métodos privados de generación de datos aleatorios en `Escudería`, ya que son similares a los que sí aparecen en `Piloto`.

4.2.4. Liga

La clase `Liga` representa las competiciones que se crean los usuarios para jugar entre ellos. Contiene el nombre, la descripción de la liga y el código de acceso para nuevos usuarios, además del conjunto de participantes que pertenecen a cada liga.

4.2.5. Equipo

Un equipo es una formación o alineación de componentes que pertenece a un usuario en una determinada liga. Es importante diferenciar este elemento de una escudería, ya que en ocasiones podría llevar a confusión al hablar de "equipo" en ambas situaciones. Por tanto, un `Equipo` se relaciona con una `Escudería`, con cinco elementos de la clase `Piloto` y con el `Participante` al que pertenece, y contiene los métodos necesarios para su modificación, cálculo de valor total y demás funcionalidades.

4.2.6. Participante

`Participante` es una clase que hace de intermediaria entre los usuarios y las ligas. Representa la participación de un usuario en una liga con un equipo concreto. A su vez, también almacena la puntuación actual que lleva dicho participante en la liga, la cual que se va actualizando al término de cada evento celebrado.

4.2.7. Evento

Cada `Evento` representa un Gran Premio del campeonato y en él se recoge información acerca de dicho evento. También se controla en todo momento el evento actual, es decir, el próximo Gran Premio que se va a celebrar o que ya se está celebrando, además de la fecha exacta en la que se debe bloquear la formación de equipos antes del comienzo del mismo.

4.2.8. Registro

`Registro` es una tabla intermedia cuya finalidad es almacenar el historial de puntos de los participantes en cada evento, ya que la clase `Participante` sólo tiene constancia del total de puntos en el momento actual pero no de cómo se han ido consiguiendo.

4.2.9. Puntuación

Por último, `Puntuación` es otra clase similar a `Registro`, sólo que ésta hace de intermediaria entre los eventos y los componentes. Al igual que en el caso anterior, se utiliza para almacenar la secuencia de puntuaciones de pilotos y escuderías en cada uno de los eventos ya celebrados.

4.3. Sistema de puntuación y cambios de valor

En la aplicación existen dos funcionalidades que destacan por encima del resto, como son el sistema de puntuación y el sistema de cambios de valor. Esto se debe a que son dos mecanismos cruciales para el desarrollo, tanto por su trascendencia en cuánto al trascurso del juego, como por su importancia a la hora de diseñar y programar sus algoritmos. Por ello, se procede a explicar detalladamente en qué consiste cada uno de estos procedimientos que encabezan la lógica de la aplicación.

4.3.1. Sistema de puntuación

Para el correcto funcionamiento del juego es esencial trasladar los resultados reales de la competición a puntuaciones ficticias en la aplicación. Como consecuencia, es necesario el diseño de un algoritmo que sea capaz de convertir los datos en bruto del evento en puntos que puedan ser asignados a los pilotos y escuderías, y por ende también a los participantes de las ligas. Para hacer posible esta transformación, se han seleccionado una serie de criterios que asignan puntos en función de los resultados extraídos de la API externa. Esos datos sin procesar que se extraen desde Ergast se corresponden con la parrilla de clasificación celebrada el sábado, el resultado de la carrera disputada el domingo (ambos son dos listados en bruto con un orden desde la primera hasta la vigésima posición de parrilla) y el resultado de la vuelta rápida. Además, para determinar los puntos no solo se tiene en cuenta el orden, sino que

también es necesario procesar esos resultados para obtener otros criterios de puntuación a partir de ellos.

Por otra parte, no sólo es necesario tener claro cuáles son los criterios de puntuación, sino también determinar cuántos puntos se deben otorgar en cada caso para que el sistema de puntuación sea justo y se mantenga la competitividad y la igualdad en las ligas durante el trascurso de la temporada.

Como resultado del análisis de todos los factores y criterios de calificación, y partiendo de la información en bruto que se obtiene de la API Ergast, se ha diseñado sistema de puntuación que aparece en las siguientes tablas. Estos valores se han refinado durante el desarrollo del proyecto y pruebas sobre las puntuaciones que se iban consiguiendo conforme ocurrían los eventos.

CLASIFICACIÓN		
POSICIÓN	**PILOTO**	**ESCUDERÍA**
1º	40	20
2º	38	19
3º	36	18
4º	34	17
5º	32	16
6º	30	15
7º	28	14
8º	26	13
9º	24	12
10º	22	11
11º	20	10
12º	18	9
13º	16	8
14º	14	7
15º	12	6
16º	10	5

17º	8	4
18º	6	3
19º	4	2
20º	2	1

Tabla 19. Sistema de puntuación. Clasificación

CARRERA		
POSICIÓN	**PILOTO**	**ESCUDERÍA**
1º	60	40
2º	55	38
3º	50	36
4º	45	34
5º	40	32
6º	36	30
7º	32	28
8º	28	26
9º	24	24
10º	21	22
11º	18	20
12º	15	18
13º	12	16
14º	10	14
15º	8	12
16º	6	10
17º	4	8
18º	3	6
19º	2	4
20º	1	2

Tabla 20. Sistema de puntuación. Carrera

BONUS PILOTOS	
BATIR COMPAÑERO DE EQUIPO	
Entre 1 y 3 posiciones	+3

Entre 4 y 7 posiciones	+6
Entre 8 y 11 posiciones	+9
Entre 12 y 20 posiciones	+12
ADELANTAMIENTOS	
Por cada posición ganada	+2
EXTRAS	
Descalificación	-10
Finalizar sin ser doblado	+5
Vuelta rápida	+10

Tabla 21. Sistema de puntuación. Bonus pilotos

BONUS ESCUDERÍAS	
COMPLETAR CARRERA	
Un coche	+5
Dos coches	+15
EXTRAS	
Descalificación	-5
Vuelta rápida	+10

Tabla 22. Sistema de puntuación. Bonus escuderías

4.3.2. Cambios de valor

El valor de un piloto o escudería tiene una doble finalidad dentro del juego. Por un lado, el valor actúa como precio a la hora de fichar a un nuevo integrante para el equipo, siendo un elemento clave en la formación de la plantilla ya que como se indica en los requisitos se dispone de un rango limitado de precio. Pero por otro lado, el valor es un indicativo del rendimiento del piloto o escudería durante la temporada, es el equivalente a una puntuación numérica que evalúa el trabajo real realizado en la pista y la productividad en cuanto a puntos de dicho componente.

Tras la celebración de cada evento, es necesario realizar modificaciones en los valores de los pilotos y escuderías en función de su puntuación obtenida. Pero este no

debe ser el único criterio a seguir, ya que una sola actuación no puede influir drásticamente en el valor del componente, es necesario hacer un estudio de la evolución de las puntuaciones del mismo y valorar como de importante debe ser su modificación.

Por tanto, para implementar el sistema de cambios de valor es necesario diseñar un algoritmo que tenga en cuenta diferentes factores de puntuación que puedan formar un sistema justo. Esas variables son la puntuación media del componente durante la temporada, la nueva puntuación obtenida en el evento actual, la última puntuación correspondiente al anterior gran premio y el valor actual del componente. Además, también hay que considerar el máximo de puntos posibles que puede obtener un componente en caso de hacer un gran premio perfecto. Como el sistema de puntuación diferencia entre pilotos y escuderías, el valor máximo es diferente para cada caso.

Una vez formalizadas las variables que deben tenerse en cuenta, el cálculo de la modificación del valor no es trivial. Hay que obtener nuevas variables como la diferencia de puntos que ha obtenido el componente en este evento respecto a su media de la temporada, para evaluar si está por encima o por debajo de su rendimiento actual. También hay que calcular la diferencia de puntuación en relación al anterior evento, para estimar si se trata de un posible cambio de tendencia en la puntuación o de un hecho aislado. Y con la combinación de ambas se obtiene la diferencia de rendimiento en el evento actual respecto a la competición en general.

- Diferencia respecto al último evento (Diferencia última) = Nueva puntuación – Última puntuación
- Diferencia respecto a la media de la temporada (Diferencia media) = Nueva puntuación – Media
- Diferencia respecto al rendimiento general (Diferencia) = (Diferencia última * 0,3 + Diferencia media * 0,7) / 10
- Ajuste = 2 - (Nueva puntuación / Máximos puntos posibles)

- Porcentaje = (100 + Diferencia * Ajuste) / 100

Tras realizar los ajustes necesarios, se obtiene un porcentaje que es el indicativo de si el piloto o la escudería está por encima o por debajo de su rendimiento (indicado sobre 100) y por tanto hay que realizar una subida o bajada de valor. Para ello, la diferencia de valor, que puede ser un incremento o una disminución (cantidad positiva o negativa), se calcula aplicando el porcentaje al valor actual del componente. Aunque la cosa no queda ahí, también se establece un rango mínimo y máximo de modificación para que no haya grandes fluctuaciones.

- Si porcentaje > 100 :

  Diferencia de valor = MÍN(Valor actual * Porcentaje - Valor actual, 3)

- Si porcentaje < 100 :

  Diferencia de valor = MÁX(Valor actual * Porcentaje - Valor actual, -3)

Finalmente, para obtener un algoritmo aún más justo y complejo, se aplica un segundo ajuste antes de corregir el valor del componente. En esta ocasión se valora el valor actual que tiene el piloto o la escudería para distinguir tres casos especiales. Si el componente ya se encuentra en un valor de precio muy bajo y debe perder aún más valor, este descenso será inferior que para el resto de componentes. De igual forma, si el componente tiene un valor que roza máximos y debe seguir incrementándose, esta subida también será controlada mediante una reducción. Sin embargo, en el caso inverso a la primera opción, las subidas de valor de los componentes que están en precios mínimos será aún más notoria, con el fin de ayudar a la competitividad en el juego. Los valores pueden oscilar entre un rango con límite inferior y superior fijo, situado entre 5 y 35.

- Rango de valores : [5M – 35M]
- Caso general :

  Nuevo valor = Valor actual + Diferencia de valor

- Rango [5M-10M] en caso de bajada :

  Nuevo valor = Valor actual + Diferencia de valor * 0,6

- Rango [5M-10M] en caso de subida :

  Nuevo valor = Valor actual + Diferencia de valor * 1,4

- Rango [30M-35M] en caso de subida :

  Nuevo valor = Valor actual + Diferencia de valor * 0,4

5

Desarrollo e Implementación

En este capítulo se explica cómo se ha desarrollado el sistema desde cero, describiendo los procedimientos que se han seguido y los mecanismos de implementación. Los contenidos se dividen en apartados siguiendo las iteraciones que se han realizado, incluyendo capturas de ciertas partes del código, que no son más que una pequeña demostración del desarrollo. En cuanto a las funcionalidades que se van a explicar, todas han seguido una misma secuencia de implementación, comenzando por el diseño de la interfaz de usuario en Android Studio, en caso de que esta sea necesaria, seguido de la programación de la funcionalidad en sí misma, y finalizando con su integración con el resto de componentes, ya sean del propio cliente o del servidor. Adicionalmente en cada iteración se han realizado las pruebas oportunas para comprobar su corrección, pero éstas se detallarán en profundidad en el siguiente capítulo.

5.1. Iniciación del proyecto

Antes de comenzar con el desarrollo de las funcionalidades especificadas en los requisitos, se han creado e inicializado todos los componentes que forman la arquitectura del sistema.

5.1.1 Creación del proyecto cliente y proyecto servidor

En primer lugar, se han creado los respectivos proyectos en Android Studio y en Visual Studio Code para la parte de la aplicación y del servidor.

Por un lado, en Android Studio se ha iniciado un proyecto desde cero con una versión 31 del SDK (Kit de desarrollo de software que genera un conjunto de herramientas útiles para la programación), aunque como versión mínima se ha establecido la 21 para lograr una mayor compatibilidad con los dispositivos de los usuarios. Una vez iniciado el proyecto, se han creado los paquetes correspondientes para implementar el Modelo-Vista-Controlador que se irán completando más adelante. (en la Figura 12 se aprecia el resultado final). Sin embargo, en el caso de las clases del Modelo, sí que se han implementado en este instante, porque ya se tiene constancia del número de clases y sus atributos gracias al análisis y modelado que se ha realizado anteriormente.

Por otro lado, en paralelo con la actividad anterior, se ha creado un proyecto Python en Visual Studio Code para el desarrollo del servidor (ver Figura 13). En él se encuentra la clase principal del servidor, que es `app.py`, y se han instalado las bibliotecas necesarias para la programación de la parte servidora, como son Flask, bson, request, PyMongo, jsonify, etc.

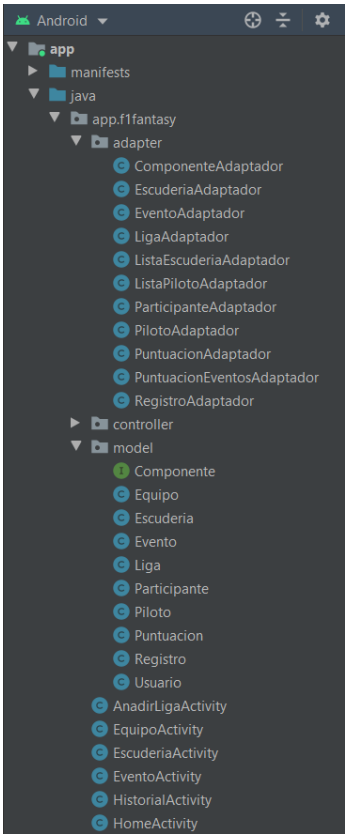

Figura 12. Desarrollo. Proyecto Android

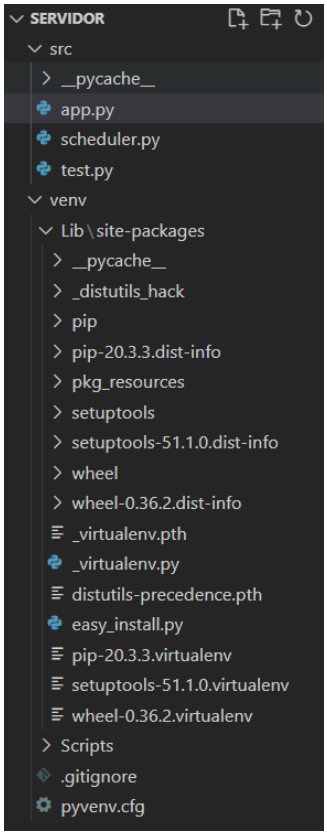

Figura 13. Desarrollo. Proyecto Python

5.1.2. Base de datos

En tercer lugar, se ha creado un clúster gratuito de Amazon Web Services (AWS) en MongoDB Atlas alojado en la región de Irlanda. En su interior, se ha formado una base de datos `fantasy` donde se encuentran las colecciones con los documentos de datos. En el caso de `Pilotos`, `Escuderías` y `Eventos`, se han inicializado los documentos con la información básica referente a cada componente, quedando así un registro para cada piloto, escudería y evento. El resto se colecciones se inicializan vacías a la espera de la generación de datos posterior.

Posteriormente , se ha conectado la aplicación del servidor con la base de datos a través de `MongoClient` de la biblioteca PyMongo y de la URL proporcionada por Atlas donde se aloja la base de datos tal y como se ve en la Figura 14.

```
######################  Inicicación  ######################

app = Flask(__name__)

url_mongo_atlas = "mongodb+srv://ivan:" + os.environ['PASS'] + "@cluster0.qecwv.mongodb.net/test?retryWrites=true&w=majority"
client = pymongo.MongoClient(url_mongo_atlas)
mongo = client.get_database('fantasy')
```

Figura 14. Desarrollo. Inicialización PyMongo

5.1.3. Peticiones con Volley

Como se ha comentado en la descripción de las tecnologías, las peticiones entre cliente y servidor se realizan a través de Volley. Esta biblioteca se ha añadido como dependencia al proyecto en Android Studio y se utiliza en cada uno de los intercambios de información entre la aplicación y la API REST, ya sea la propia o la de Ergast.

El uso de Volley una característica común para todos los métodos que realizan peticiones desde la aplicación, en concreto desde las clases controladoras del MVC. Por ello, a continuación se procede a explicar cómo se realiza una petición usando esta biblioteca, y de aquí en adelante, cuando se nombre un intercambio de información entre cliente y servidor o una consulta a la API REST, se conocerá que se realiza de la siguiente forma (ver Figura 15).

En primer lugar, es necesario tener la dirección URL donde se encuentra el punto de acceso de la consulta que se desea realizar. Inicialmente se crea una cola de peticiones (`Request Queue`) para el contexto actual de la aplicación, es decir, la actividad en la que se halla la aplicación. Posteriormente, se ejecuta una llamada indicando del tipo de petición y el formato del resultado de la petición, en este caso un array de objetos de tipo JSON, y se espera la respuesta. En caso de éxito se pasa al método correspondiente `onResponse` y en caso de error se ejecuta `onErrorResponse`. El resultado de la petición se almacena en una variable del mismo tipo del formato del resultado del `Request`.

```
public static void obtenerUsuario(Participante participante, final VolleyCallBack callBack){
    String url = "https://fantasyf1-ivan.herokuapp.com/usuarios/usuario/" + participante.getUsuario();
    RequestQueue mQueue = Volley.newRequestQueue(context);

    JsonArrayRequest request = new JsonArrayRequest(Request.Method.GET, url,  jsonRequest: null,
            new Response.Listener<JSONArray>() {
                @Override
                public void onResponse(JSONArray response) {
                    try {
                        auxUsuario = new Usuario(response.getJSONObject( index: 0));
                        callBack.onSuccess();
                    } catch (JSONException e) {
                        e.printStackTrace();
                    }
                }
            }, new Response.ErrorListener() {
        @Override
        public void onErrorResponse(VolleyError error) { error.printStackTrace(); }
    });

    mQueue.add(request);
}
```

Figura 15. Desarrollo. Volley request

En cuanto a la petición, con eso ya es suficiente para obtener los resultados, pero en este caso se precisa también de otro mecanismo que es `VolleyCallback`. Este se crea en la actividad desde la que se llama al método de consulta, y hasta que no se recibe un aviso de éxito a través del `callback` no se continúa con la ejecución del código de la actividad. Esto es necesario ya que la actividad (la pantalla de la aplicación) sigue ejecutándose después de realizar la petición, y puede darse el caso de que se requieran valores o información que aún no han sido retornados y por tanto nunca lleguen a mostrarse, y con el uso del `callback` se soluciona este y otros posibles conflictos de sincronización.

```
LigaControlador.obtenerUsuario(item, new VolleyCallBack() {
    @SuppressLint("SetTextI18n")
    @Override
    public void onSuccess() {
        textoUsuario.setText(LigaControlador.auxUsuario.getNombre() + " " + LigaControlador.auxUsuario.getApellidos());
    }
});
```

Figura 16. Desarrollo. Volley callback

5.2. Primera iteración

En esta iteración se incluye el desarrollo de la API REST y la implementación de las primeras funcionalidades en la aplicación. Estas funcionalidades incluyen la creación y acceso del usuario y las visualizaciones de la información sobre pilotos, escuderías y eventos (que son elementos más estáticos).

5.2.1. API REST

Con la arquitectura ya asentada y los proyectos iniciados, lo primero que se ha desarrollado ha sido la API REST propia. Se han creado los métodos para realizar las peticiones HTTP necesarias para introducir y extraer información de la base de datos.

Para la colección `Usuario` se ha creado un CRUD con operaciones de creación, modificación (ejemplo de la Figura 17) y obtención, a excepción de la eliminación de usuarios ya que no es una característica necesaria en la aplicación. En el caso de Liga ocurre exactamente lo mismo que en Usuario, mientras que para `Piloto` y `Escudería` se han desarrollado métodos de obtención y modificación únicamente, debido a que éstos ya están creados desde el momento de inicialización de la base de datos.

```python
@app.route('/usuarios/<id>', methods=['PUT'])
def update_usuario(id):
    nombre = request.json['nombre']
    apellidos = request.json['apellidos']
    usuario = request.json['usuario']
    clave = request.json['clave']
    respuesta = request.json['respuesta']
    correo = request.json['correo']

    if nombre and usuario and clave and correo:
        mongo.usuarios.update_one({'_id': ObjectId(id)}, {'$set': {
            'nombre': nombre,
            'apellidos': apellidos,
            'correo': correo,
            'clave': clave,
            'respuesta': respuesta,
            'usuario': usuario
        }})
        response = jsonify({'mensaje': 'Usuario actualizado correctamente'})
        return response
    else:
        return not_found()
```

Figura 17. Desarrollo. PUT Usuario

Para las colecciones restantes, se han creado solamente procedimientos de obtención (de tipo GET), ya que los nuevos registros no se hacen a través de la API REST sino que son funcionalidades incluidas dentro de la actualización de puntos que se realiza más adelante. Además de las operaciones nombradas, también existen consultas personalizadas para obtener resultados con filtros, como por ejemplo para la obtención de pilotos a través de su identificador de Ergast (que también se almacena en la base de datos) o para obtener las puntuaciones de determinadas rondas, entre otras consultas.

5.2.2. Inicio de sesión, registro y perfil

La dos primeras funcionalidades que se han desarrollado están relacionadas entre sí, y son el inicio de sesión y el registro en la aplicación (ver ejemplo en la Figura 18). En primer lugar, se ha diseñado la interfaz de usuario de ambas pantallas para obtener los datos necesarios del usuario.

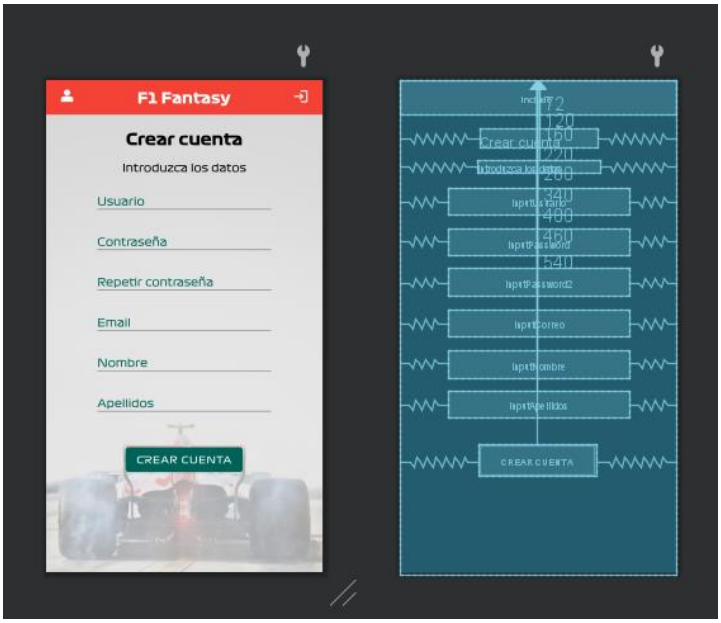

Figura 18. Desarrollo. Interfaz registro

En el caso de registro, se recogen los datos del nuevo usuario por pantalla, se comprueba la validez de los campos y se realiza un envío al servidor para se encargue

del proceso de creación. Previamente, se realiza una comprobación de existencia del usuario, donde el servidor realiza una búsqueda e informa a la aplicación si ha encontrado o no coincidencias para el valor introducido por el usuario. En caso de que los datos estén correctos, cuando el servidor los recibe se encarga de la creación de un nuevo usuario en la base de datos.

Para el inicio de sesión, la aplicación recoge el usuario y contraseña, en función de si coinciden o no con la información que consulta el servidor a base de datos, se aprueba o se rechaza la operación. En cuanto al perfil del usuario, se ha desarrollado una pantalla de modificación de información, que es una funcionalidad similar al proceso de registro pero para un usuario con cuenta ya existente e iniciada.

5.2.3. Pilotos, escuderías y eventos

Se ha diseñado un menú principal desde el cual se puede acceder a las diferentes secciones que componen el juego: Pilotos, Escuderías, Eventos y Ligas. Se ha desarrollado una pantalla donde se muestra un listado con todos los pilotos que forman parte de la competición. Para cada listado que aparece en la aplicación, ha sido necesario crear un ítem y un adaptador especial (como se puede ver en la Figura 19) para mostrar información personalizada en cada una de sus filas.

```
@Override
public View getView(int i, View view, ViewGroup viewGroup) {
    Piloto item = getItem(i);
    view = LayoutInflater.from(context).inflate(R.layout.item_piloto_2,  root: null);

    TextView textoNombre = (TextView) view.findViewById(R.id.pilotoNombre);
    TextView textoValor = (TextView) view.findViewById(R.id.pilotoValor);
    TextView textoPuntos = (TextView) view.findViewById(R.id.pilotoPuntos);
    TextView textoEscuderia = (TextView) view.findViewById(R.id.pilotoEscuderia);
    ImageView imagen = (ImageView) view.findViewById(R.id.pilotoImagen);

    textoNombre.setText(item.getNombre());
    textoValor.setText(String.valueOf(item.getValor()) + "M €");
    textoPuntos.setText(String.valueOf(item.getPuntos()) + " pts");
    textoEscuderia.setText(item.getEscuderia());
    Picasso.with(context).load(item.getFoto()).into(imagen);

    return view;
}
```

Figura 19. Desarrollo. Adaptador listado de pilotos

Si se pulsa sobre uno de los componentes, se abre otra sección donde se muestra información personalizada. La primera parte de esta sección se obtiene a través de la información predefinida que hay sobre cada piloto en la base de datos. A continuación, aparecen los datos y estadísticas relacionados con las puntuaciones del juego, que también se extraen de la base de datos. Por último, aparece un dato aleatorio acerca de este piloto, el cual se extrae a partir de una consulta a la API externa y se procesa para convertirlo en un mensaje presentable para el usuario. Además, la interfaz de usuario se complementa con imágenes y fotografías que complementen toda esta información y hagan más atractiva la sección.

Las fotografías y grafismos se muestran gracias al uso de una biblioteca llamada Picasso, que facilita la inclusión de imágenes a partir de una URL sin necesidad de descargar los archivos en la aplicación.

Las secciones de Eventos y Escuderías se han desarrollado de forma similar y empleando las mismas técnicas al caso de Piloto, ya que las funcionalidades son parecidas.

5.3. Segunda iteración

En esta segunda iteración se han desarrollado las funcionalidades relacionadas con la creación de ligas y la formación de equipos.

5.3.1. Nueva liga

Existen dos mecanismos para que el usuario participe en una liga, la creación desde cero de una nueva liga y la unión a una ya existente. Para ello, se ha desarrollado una pantalla donde el usuario dispone de ambas opciones. Con los datos que se recogen de los formularios, se realizan peticiones al servidor para almacenar en base de datos la información sobre la nueva liga con un único participante, o para añadir un nuevo participante a una determinada liga, en función de la elección del usuario.

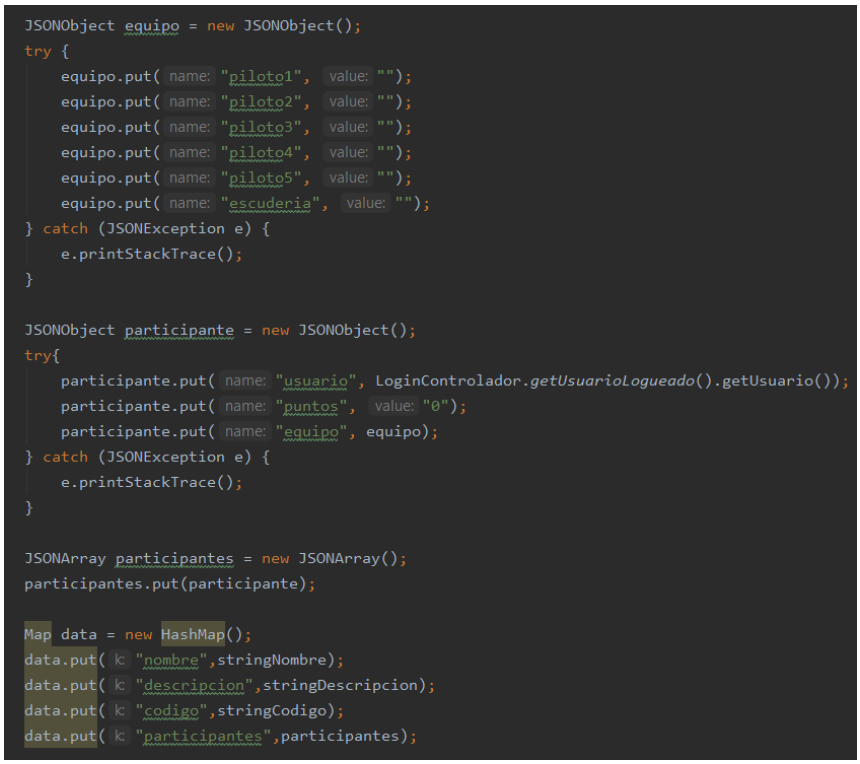

Figura 20. Desarrollo. Mapeo de una nueva liga

Para poder enviar los datos en la consulta, es necesario mapearlos para posteriormente convertirlos a formato JSON y enviarlos mediante Volley. En la imagen anterior (ver Figura 20) se puede observar el proceso de mapeo de datos para una nueva liga, donde se inicializa el equipo del usuario y se le añade como primer participante.

5.3.2. Formación de equipo

Para que los participantes de las ligas puedan formar sus equipos, se han añadido los botones correspondientes para comprar y vender pilotos y escuderías. Se ha tenido en cuenta que para aquellos componentes que ya formen parte del equipo hay que restringirles la opción de fichaje, comprobando previamente el equipo del usuario. El detalle más importante a la hora de realizar una nueva incorporación es que la suma total de valor del equipo no supere el límite superior.

Hay que tener en cuenta que cada vez que se realiza una modificación en el equipo, también es necesario actualizar la liga a la que pertenece ese equipo, ya que por la arquitectura de la base de datos, los participantes contienen cada uno de ellos un equipo y a su vez las ligas contienen un listado de participantes. Por ello, hay que mapear no sólo la información del equipo sino también la de la liga, tal y como se explica en el apartado anterior. Posteriormente, el servidor se encarga de la actualización de la liga completa en base de datos, incluyendo participantes, y por ende sus equipos.

```python
@app.route('/ligas/<id>', methods=['PUT'])
def update_liga(id):
    nombre = request.json['nombre']
    descripcion = request.json['descripcion']
    codigo = request.json['codigo']
    participantes = request.json['participantes']

    if nombre and codigo:
        mongo.ligas.update_one({'_id': ObjectId(id)}, {'$set': {
            'nombre': nombre,
            'descripcion': descripcion,
            'codigo': codigo,
            'participantes': participantes
        }})
        response = jsonify({'mensaje': 'Liga actualizada correctamente'})
        return response
    else:
        return not_found()
```

Figura 21. Desarrollo. Actualización de la liga

5.4. Tercera iteración

En esta iteración se ha realizado la lógica de reparto de puntos y de modificaciones de valor. Por tanto, la mayor parte del trabajo de esta etapa recae en el lado del servidor.

5.4.1. Reparto de puntos

Se ha creado una nueva clase `scheduler.py` al margen de la clase principal del servidor, la cual contendrá todo el código necesario para realizar la actualización tanto de puntos como de valores tras la celebración de un evento. De esta forma se podrá

ejecutar estas funciones en un momento concreto y de forma automática, tal y como se explicará en la siguiente iteración.

Para aplicar el reparto de puntos, en primer lugar hay que extraer los datos de los resultados tanto de la clasificación como de la carrera y almacenarlos en arrays auxiliares para posteriormente asignar los puntos. Para lograrlo, hay que realizar una consulta a la API externa Ergast en relación al evento celebrado y obtener aquella información que es útil, que se trata simplemente de un listado de pilotos y escuderías ordenados por de primero a último. A continuación, en la Figura 22 se muestra un ejemplo para el caso de la clasificación.

```
def get_Puntos(ronda_actual, evento_actual):
    url = 'http://ergast.com/api/f1/2022/' + ronda_actual + '/qualifying.json'
    response = requests.get(url).json()

    resultado_clasificacion = dict()
    resultado_clasificacion_escuderias = dict()

    MRData = response['MRData']
    RaceTable = MRData['RaceTable']
    Races = RaceTable['Races'][0]

    for i in range(20):
        QualifyingResults = Races['QualifyingResults'][i]
        Driver = QualifyingResults['Driver']
        driverId = Driver['driverId']
        Constructor = QualifyingResults['Constructor']
        constructorId = Constructor['constructorId']

        resultado_clasificacion[i] = driverId
        resultado_clasificacion_escuderias[i] = constructorId

    puntuaciones_clasificacion_pilotos = get_puntos_clasificacion_pilotos(resultado_clasificacion)
    puntuaciones_clasificacion_escuderias = get_puntos_clasificacion_escuderias(resultado_clasificacion_escuderias)
```

Figura 22. Desarrollo. Resultados clasificación

Cabe destacar que parte de los puntos que se repartirán, concretamente aquellos que se otorgan por posición, se han inicializado al principio de la clase.

```
########################  Puntos  ########################

PUNTOS_CLASIFICACION_PILOTOS = [40,38,36,34,32,30,28,26,24,22,20,18,16,14,12,10,8,6,4,2]
PUNTOS_CLASIFICACION_ESCUDERIAS = [20,19,18,17,16,15,14,13,12,11,10,9,8,7,6,5,4,3,2,1]
PUNTOS_CARRERA_PILOTOS = [60,55,50,45,40,36,32,28,24,21,18,15,12,10,8,6,4,3,2,1]
PUNTOS_CARRERA_ESCUDERIAS = [40,38,36,34,32,30,28,26,24,22,20,18,16,14,12,10,8,6,4,2]
MAXIMO_PUNTOS_PILOTOS = 127
MAXIMO_PUNTOS_ESCUDERIAS = 80
```

Figura 23. Desarrollo. Puntuaciones

Una vez se han extraído los resultados en bruto, se debe comenzar su procesamiento e ir transformándolos en puntos, teniendo en cuenta que para esta labor no sólo es necesario el orden extraído anteriormente, sino también alguna información extra como la vuelta rápida, la posición del compañero de equipo, etc. Esto se consigue siguiendo el sistema de reparto de puntos que se ha explicado anteriormente en el Apartado 4.3.1. En la siguiente imagen (Figura 24) se puede observar una pequeña muestra de la programación del reparto de puntos para el caso de pilotos, concretamente de la puntuación obtenida de la carrera.

```
puntos = 0
puntos += PUNTOS_CARRERA_PILOTOS[i]

val_list = list(resultado_carrera.values())

posicion_compañero = val_list.index(compañero)
diferencia = posicion_compañero - i

if diferencia >= 1 and diferencia <= 3:
    puntos += 3
elif diferencia >= 4 and diferencia <= 7:
    puntos += 6
elif diferencia >= 8 and diferencia <= 11:
    puntos += 9
elif diferencia >= 11 and diferencia <= 20:
    puntos += 12

val_list_clasificacion = list(resultado_clasificacion.values())
posicion_clasificacion = int(val_list_clasificacion.index(piloto))
diferencia_puestos = posicion_clasificacion - i

if diferencia_puestos > 0:
    puntos += diferencia_puestos * 2

if status_carrera[i] == 'Disqualified':
    puntos -= 10
elif status_carrera[i] == 'Finished':
    puntos += 5

if vuelta_rapida == piloto:
    puntos += 10

puntuaciones_carrera[piloto] = puntos
```

Figura 24. Desarrollo. Sistema de puntuación

5.4.2. Modificaciones de valor

Como se ha visto en el Apartado 4.3.2, se ha creado un algoritmo para realizar los cambios de valor de los componentes tras la celebración de un Gran Premio. En

este caso, la implementación del mismo es sencilla, ya que la complejidad residía principalmente en su diseño. Para desarrollarlo simplemente se han trasladado las instrucciones en pseudocódigo a lenguaje Python, quedando se la siguiente forma (ver Figura 25).

```python
def modificacion_valor(media, nueva_puntuacion, ultima_puntuacion, valor_actual, MAXIMO_PUNTOS):
    diferencia_ultima = nueva_puntuacion - ultima_puntuacion
    diferencia_media = nueva_puntuacion - media
    diferencia = (diferencia_ultima * 0.3 + diferencia_media * 0.7) / 10
    ajuste = 2 - (nueva_puntuacion / MAXIMO_PUNTOS)
    porcentaje = (100 + diferencia * ajuste) / 100

    if porcentaje > 1:
        diferencia_valor = min(valor_actual * porcentaje - valor_actual, 3)
    elif porcentaje < 1:
        diferencia_valor = max(valor_actual * porcentaje - valor_actual, -3)
    else:
        diferencia_valor = 0

    if valor_actual < 10 and diferencia_valor < 0:
        nuevo_valor = max((valor_actual + diferencia_valor * 0.6), 5)
    elif valor_actual < 10 and diferencia_valor > 0:
        nuevo_valor = max((valor_actual + diferencia_valor * 1.4), 5)
    elif valor_actual > 30 and diferencia_valor > 0:
        nuevo_valor = min((valor_actual + diferencia_valor * 0.4), 35)
    else:
        nuevo_valor = (valor_actual + diferencia_valor)

    return round(nuevo_valor, 1)
```

Figura 25. Desarrollo. Algoritmo de modificación de valor

Una vez realizada la asignación de puntos y las modificaciones de valor, el último paso es actualizar la información de los pilotos y escuderías en la base de datos, además de sumar los puntos a los participantes según los miembros de su equipo. También se almacenan los registros de puntuaciones asociados a cada componente para así poder mostrarlos en el historial.

5.5. Cuarta iteración

En esta última iteración se han refinado las funcionalidades desarrolladas en las anteriores y se han complementado algunas de ellas, destacando la automatización de la actualización de puntos y la compleción de los datos aleatorios ofrecidos a partir de

Ergast. Además, se ha mejorado la aplicación móvil gracias a la incorporación de algunos elementos visuales.

5.5.1. Actualización de puntos

La actualización de puntos que se realiza al término de cada evento, podrá realizarse de forma automática sin necesidad de ejecutar manualmente las funciones que se encuentran en el servidor. Para ello, se ha utilizado una funcionalidad de Heroku llamada `scheduler`, que simplemente es una herramienta para programar tareas y que se ejecuten en un momento determinado. La plataforma ofrece varias opciones similares para programar estas tareas, aunque la gran mayoría son funcionalidades no gratuitas. Por tanto, se ha seleccionado una de ellas en su versión de prueba para comprobar que funciona correctamente y que se produce la actualización en el servidor de forma adecuada.

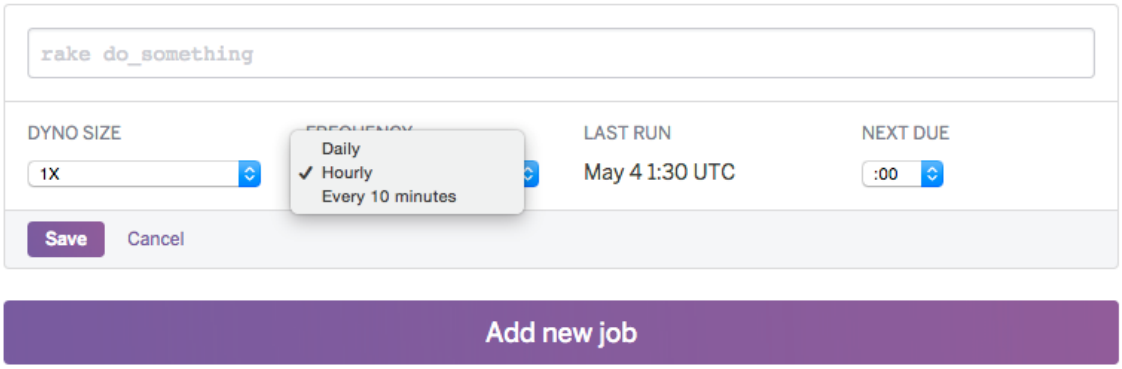

Figura 26. Desarrollo. Advanced scheduler Heroku

En esta herramienta, se indican las fechas en las que se desean realizar las actualizaciones, las cuales se saben de antemano ya que el calendario de la temporada queda definido antes de su comienzo. Luego, en cada uno de los instantes programados, se ejecutará la clase `scheduler.py` programada en el servidor de la aplicación y que, como ya se ha visto en las iteraciones anteriores, contiene el código necesario para realizar las trasformaciones de puntos y valores correspondientes.

5.5.2. Estadísticas aleatorias

Una de las funcionalidades que aparece en la aplicación, es la inclusión de datos históricos aleatorios sobre un determinado piloto, escudería o evento. Como ya se ha comentado, esto se realiza procesando información en bruto extraída de la API Ergast y modificándola para mostrarla en una sección de curiosidades sobre el elemento que se está visualizando. Para conseguir esto, ha sido necesario programar diferentes funciones que soliciten los datos en bruto a través de cada punto de acceso correspondiente, que posteriormente extraigan los valores necesarios en cada caso, y que finalmente estructure los mensajes para que esto tenga sentido.

Se han programado diferentes métodos para cada uno de los casos (pilotos, escuderías y eventos). Cada uno con sus particularidades y con consultas a distintas secciones de información de la API. Un ejemplo para el caso de piloto sería el que aparece a continuación en las imágenes (ver figuras 27 y 28), donde se puede ver el método que hace posible la transformación, y su posterior resultado visual en la aplicación. En este caso se trata de una estadística que involucra al piloto que se está visualizando, pero también a su actual escudería, ya que el objetivo es mostrar la información de su debut en ella.

```java
private static void debutConSuEscuderia(Piloto piloto, final VolleyCallBack callBack) {
    String escuderiaid = PilotoControlador.obtenerEscuderiaId(piloto.getEscuderia());
    String url = "https://ergast.com/api/f1/drivers/" + piloto.getDriverid() + "/constructors/" + escuderiaid + "/results.json";
    RequestQueue mQueue = Volley.newRequestQueue(context);

    JsonObjectRequest request = new JsonObjectRequest(Request.Method.GET, url, jsonRequest: null,
            new Response.Listener<JSONObject>() {
                @Override
                public void onResponse(JSONObject response) {
                    try{
                        JSONArray jsonArray = response.getJSONObject("MRData").getJSONObject("RaceTable").getJSONArray( name: "Races");
                        JSONObject carrera = jsonArray.getJSONObject( index: 0);
                        JSONObject circuito = carrera.getJSONObject("Circuit");
                        String localizacion = circuito.getJSONObject("Location").getString( name: "locality");
                        String anio = carrera.getString( name: "season");

                        mensaje = context.getResources().getString(R.string.sabiasQue) + " " + piloto.getNombre() + " " +
                                context.getResources().getString(R.string.debutEscuderia) + " " + piloto.getEscuderia() + " " +
                                context.getResources().getString(R.string.debutEscuderia1) + " " + localizacion + " " +
                                context.getResources().getString(R.string.debutEscuderia2) + " " + anio + ".";

                        callBack.onSuccess();
                    } catch (JSONException e) {
                        e.printStackTrace();
                    }
                }
            }, new Response.ErrorListener() {
        @Override
        public void onErrorResponse(VolleyError error) { error.printStackTrace(); }
    });
```

Figura 27. Desarrollo. Dato aleatorio

Figura 28. Desarrollo. Sabías que...?

6

Pruebas y mantenimiento

En este capítulo se describen los diferentes procedimientos de prueba que se han realizado en el sistema, tanto en la aplicación cliente como en el servidor. Se han implementado diferentes tipos de pruebas con el objetivo de cubrir el máximo número de componentes que forman todo el sistema, teniendo también en cuenta la experiencia de manejo de los usuarios. Además, se explica brevemente algunos aspectos básicos necesarios para un correcto mantenimiento.

6.1. Pruebas REST API

Una vez que han sido creados los servicios que forman la API REST propia de la aplicación, es necesario probar que estos funcionan correctamente, tanto en funcionalidad como en localización a través de su rutas o puntos de acceso (*endpoints*). Para ello se ha usado la herramienta de pruebas Postman.

Se ha creado una colección de pruebas con un conjunto de paquetes, los cuales contienen cada uno de los servicios ofrecidos por la API REST. Los paquetes están

divididos por clases o tipos de datos que hay en la aplicación (`Usuario`, `Evento`, `Piloto`…) y en cada paquete se testean los diferentes tipos de consulta para cada componente. En resumen, se han probado un total de treinta y una consultas con sus respectivos *endpoints*. En la imagen de la Figura 29 que se muestra a continuación, se puede observar la estructura de los paquetes de pruebas y algunas de las consultas para los paquetes de usuarios y escuderías.

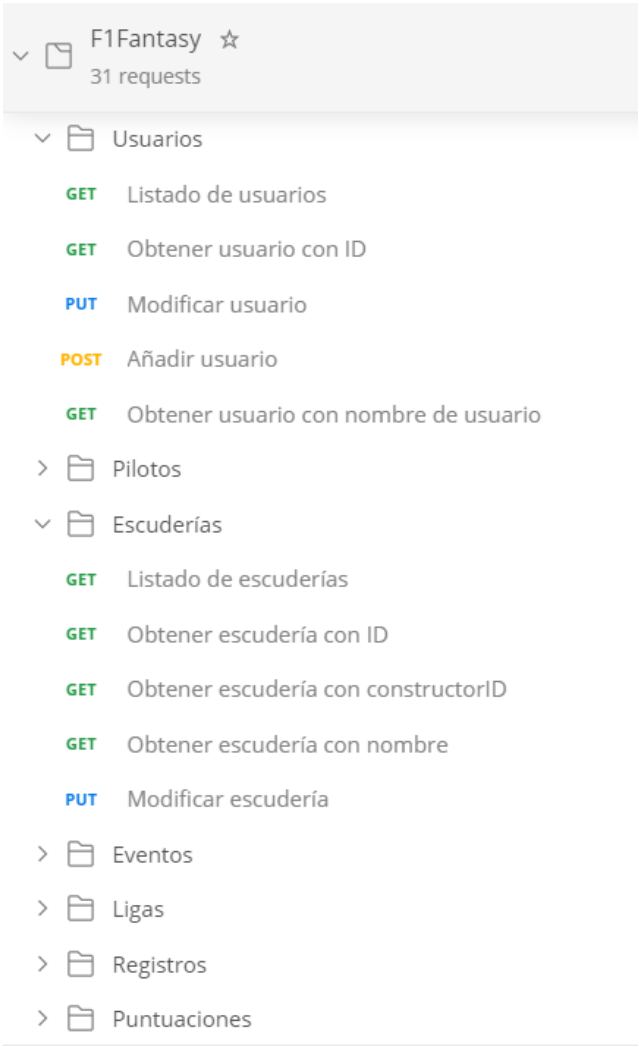

Figura 29. Pruebas API REST

El proceso de prueba de cada servicio es bastante simple e intuitivo. Se indica la dirección URL donde se encuentra alojado el servicio, en este caso todas parten de una misma dirección que apunta al servidor alojado en Heroku y que es la siguiente "https://fantasyf1-ivan.herokuapp.com". A continuación, se añade en la dirección el

nombre de la colección de datos, el nombre del método a probar y se indica el tipo de petición (GET, POST, PUT). En caso de que sea necesario enviar información, se añade un objeto JSON como en el cuerpo de la petición (ver Figura 30).

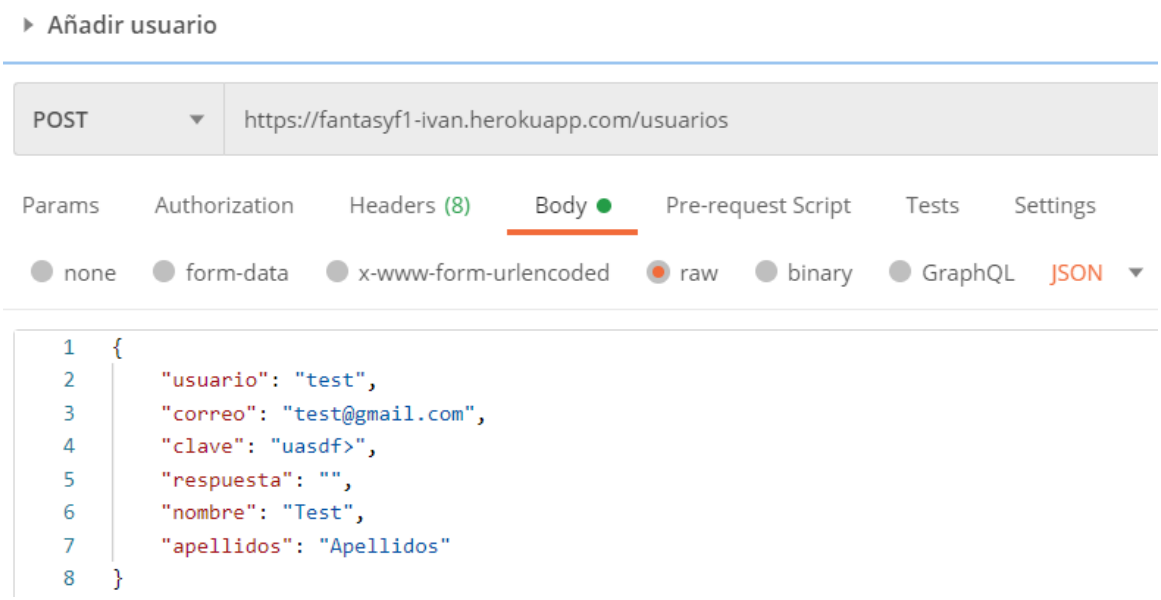

Figura 30. Ejemplo Test Postman. POST Añadir Piloto

Cuando el servidor procesa la consulta y envía una respuesta hay que comprobar que el resultado es positivo y que se recibe un código de categoría 200, que es de tipo satisfactorio. Además de eso, en caso de que se devuelva información en formato JSON, hay que ratificar que ésta es la esperada para la consulta realizada, como se puede observar en el ejemplo de la Figura 31 cuando se consulta el listado con el historial de puntos de los pilotos.

Cabe mencionar que a través de Postman también se han probado todos los puntos de acceso y servicios ofrecidos por la API abierta Ergast. Estos no se han almacenado en ningún paquete de pruebas ya que no se trata de un desarrollo propio, pero aun así se ha comprobado el correcto funcionamiento de cada uno de ellos durante la implementación de las funcionalidades que así lo requerían. Además, ha sido de gran utilidad para comprobar no solo la validez de los servicios sino también los resultados

ofrecidos para cada consulta, siendo así un gran complemento a la propia documentación de Ergast.

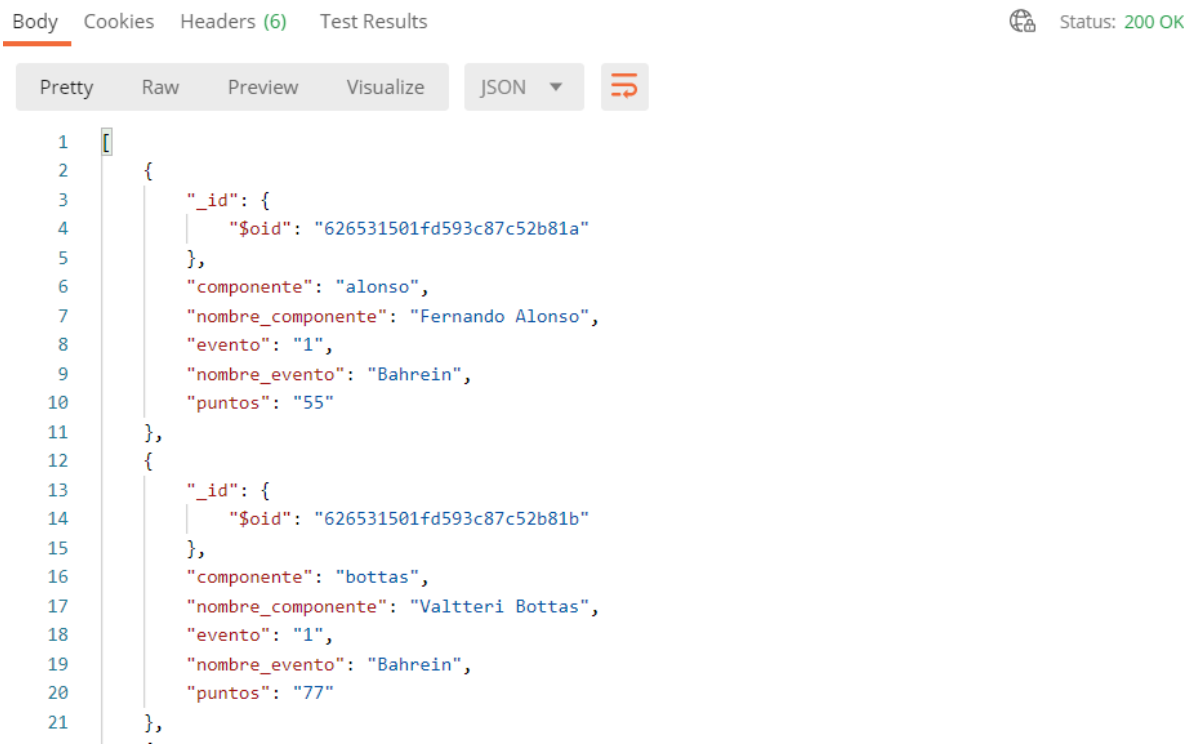

Figura 31. Ejemplo Test Postman. GET Historial Puntos

6.2. Pruebas unitarias

Un test unitario es un tipo de prueba donde se analiza el correcto funcionamiento un método o un bloque de código independiente, es decir, se prueba una sección aislada que no depende de otros componentes. Este tipo de pruebas se han incluido tanto en la parte de la aplicación en Java como en el backend desarrollado en Python. Aunque ambas funcionan de forma similar y se basan en el mismo concepto, se procede a explicar por separado en dos apartados diferentes debido a que las bibliotecas usadas en cada caso son distintas.

6.2.1. JUnit y Mockito

Para las pruebas unitarias en Android se ha usado JUnit y Mockito, que son, respectivamente, la biblioteca y el framework por excelencia para la creación de tests unitarios en Java.

JUnit permite crear clases de test para comprobar el funcionamiento de cada método y comparar los resultados obtenidos con los esperados. Para ello, principalmente se han usado los `Asserts` que proporciona esta biblioteca y que se usan para verificar que el contenido que devuelve el bloque de código testeado coincide con un objeto, valor o premisa esperada. Además, incluye una serie de etiquetas para comprobar el lanzamiento de excepciones o para tener métodos comunes que se ejecutan antes o después de cada test, para evitar la repetición de código, aunque en este caso solo se han usado algunas de estas opciones [19].

Mockito es un framework que permite la simulación de objetos que no existen realmente, lo que se conoce como `mock`. Es vital para la creación de ciertas pruebas unitarias debido a que facilita la inclusión de otros objetos y métodos que son necesarios para probar la función actual pero que en realidad no importa su funcionamiento interno, tan sólo la simulación de su comportamiento para poder testear el método que se está probando [20].

La combinación de ambas ha permitido la creación de tests unitarios para probar las diferentes funcionalidades desarrolladas, concretamente los métodos que se encuentran en los controladores, que son aquellos en los que tiene sentido realizar este tipo de pruebas, como se puede ver en el ejemplo de la Figura 32. Han sido de gran utilidad para detectar y prevenir errores de programación que no se habían tenido en cuenta durante las distintas etapas de desarrollo.

```java
@Test
public void obtenerParticipanteLogueado() {
    Usuario usuario = mock(Usuario.class);
    when(usuario.getUsuario()).thenReturn("test");
    LoginControlador.setUsuarioLogueado(usuario);
    Liga mLiga = mock(Liga.class);
    Participante mParticipante = mock(Participante.class);
    when(mParticipante.getUsuario()).thenReturn("test");
    List<Participante> mParticipantes = new ArrayList<>();
    mParticipantes.add(mParticipante);
    when(mLiga.getParticipantes()).thenReturn(mParticipantes);
    LigaControlador.ligaSeleccionada = mLiga;
    Participante p = LigaControlador.obtenerParticipanteLogueado();
    assertEquals(p.getUsuario(), actual: "test");
}
```

Figura 32. Ejemplo JUnit y Mockito

6.2.2. Unittest

Unittest es un módulo de Python inspirado en JUnit que también sirve para crear pruebas unitarias [21]. En este caso, se ha utilizado para crear tests para probar las funcionalidades de los métodos presentes en el servidor pero que no están relacionados con la API REST, sino con las funcionalidades de asignación de puntos y cambios de valor. Las pruebas también se realizar usando `Asserts` de igual forma que ocurre en Java con JUnit.

Pese a que no existen demasiados métodos independientes que probar en la parte servidora, ya que una gran parte de ella está formada por métodos de intercambio de información entre la base de datos y la aplicación, y que ya se han probado previamente, la inclusión de las pruebas unitarias ha sido de gran utilidad para verificar no sólo el buen funcionamiento de los métodos, sino también para comprobar el comportamiento de los algoritmos (ver Figura 33). Es decir, se han usado estos tests para comprobar que la asignación de puntos y de valores es correcta, y además, para analizar a través de los resultados si las modificaciones de valor tienen o no sentido lógico y si cumplen con los objetivos que se perseguían en un principio.

```
def test_modificacion_valor_desplome(self):
    self.assertEqual(scheduler.modificacion_valor(67.8, 22, 28, 24.0, 127), 22.5)

def test_modificacion_valor_ascenso(self):
    self.assertEqual(scheduler.modificacion_valor(39.1, 76, 61, 16.3, 127), 17.0)

def test_modificacion_valor_baratos(self):
    self.assertEqual(scheduler.modificacion_valor(20.5, 12, 21, 7.1, 127), 7.0)

def test_modificacion_valor_caros(self):
    self.assertEqual(scheduler.modificacion_valor(75.7, 73, 92, 30.3, 127), 30.0)

def test_get_puntos_pilotos(self):
    resultado_carrera = dict()
    resultado_clasificacion = dict()
    status_carrera = dict()

    for i in range(20):
        resultado_carrera[i] = 'hamilton'
        resultado_clasificacion[i] = 'hamilton'
        status_carrera[i] = 'Finished'

    resultado_carrera[17] = 'russell'
    resultado_carrera[14] = 'ocon'
    resultado_carrera[9] = 'alonso'
    resultado_clasificacion[6] = 'russell'
    resultado_clasificacion[10] = 'ocon'
    resultado_clasificacion[12] = 'alonso'

    puntuaciones_pilotos = scheduler.get_puntos_pilotos(resultado_carrera,
    resultado_clasificacion, status_carrera, 'alonso')
    self.assertEqual(puntuaciones_pilotos['alonso'], 48)
```

Figura 33. Ejemplos Unittest

6.3. Pruebas de usabilidad

Las pruebas de usabilidad son un método de evaluación de la experiencia de los usuarios en la aplicación desarrollada. Son útiles para realizar un estudio de la interfaz de usuario y de la interacción entre el usuario final y la aplicación. También es de utilidad para detectar y corregir posibles problemas relacionados con la capa Vista del patrón MVC.

En este caso, se ha seleccionado un grupo de personas externas al desarrollo del sistema que serán los encargados de probar la aplicación y realizar un cuestionario personal sobre su experiencia. Esta selección de usuarios se ha realizado de forma premeditada y teniendo en cuenta el perfil de cada uno de ellos, de tal forma que se

han escogido personas familiarizadas con este tipo de juegos y aplicaciones, y otras cuyos conocimientos sobre la temática eran desconocidos hasta el momento de la prueba.

La evaluación de usabilidad se ha realizado siguiendo el estándar de pruebas SUS, conocido así por sus siglas en inglés (*System Usability Scale*) [22]. Es una herramienta sencilla y útil tanto para los usuarios como para los desarrolladores y con ella se puede realizar una evaluación completa del uso de la aplicación. El cuestionario que deben realizar los usuarios tras probar la aplicación se compone de diez preguntas y cinco posibles valores de respuesta, siendo el contenido del mismo el siguiente:

Criterios de respuesta:

(1) Nada de acuerdo

(2) En desacuerdo

(3) Indiferente

(4) De acuerdo

(5) Muy de acuerdo

Preguntas:

1. Creo que me gustaría usar esta aplicación con frecuencia.
2. Creo que la aplicación es innecesariamente compleja.
3. Pienso que la aplicación es fácil de usar.
4. Creo que necesitaría la ayuda de una persona para poder utilizar esta aplicación.
5. Creo que las diversas funciones en esta aplicación estaban bien integradas.
6. Creo que hay demasiada inconsistencia en la aplicación.
7. Pienso que la mayoría de la gente aprendería a usar esta aplicación en poco tiempo.
8. La aplicación me pareció muy engorrosa de usar.

9. Me sentí muy seguro usando la aplicación.
10. Necesitaba aprender muchas cosas antes de poder comenzar con esta aplicación.

En el tercer apéndice de este documento (ver Apéndice C) se recogen los resultados de los cuestionarios realizados a cuatro de los usuarios seleccionados para la prueba de evaluación, mostrando sus respuestas a cada una de las preguntas planteadas.

Los resultados de las pruebas de usabilidad se realizan conforme al siguiente estándar:

1. Sumar las respuestas de los enunciados impares y posteriormente restarle 5.
2. Sumar las respuestas de los enunciados pares y restárselo a 25.
3. Sumar ambos resultados previos y multiplicar por 2,5.

Tras el proceso de pruebas, el resultado obtenido en este caso arroja un valor de 69 sobre 100. Teniendo en cuenta que la media se sitúa aproximadamente en 68 puntos y en base a lo que indica el estándar, se puede concluir que el resultado que se ha logrado es satisfactorio.

Además de los cuestionarios de usabilidad, también se incluye una sección de problemas y sugerencias que han planteado estos usuarios durante la realización de la prueba. Se han tomado nota de sus comentarios y posteriormente se han traducido a términos de desarrollo e implementación, para buscar una solución en caso de problema, o bien para realizar un análisis de viabilidad en caso de que se trate de una nueva propuesta o recomendación. Esta sección se encuentra detrás de los cuestionarios del apéndice nombrado en el párrafo anterior.

6.4. Mantenimiento de la aplicación

Una vez se tiene la aplicación desarrollada, probada y en funcionamiento, es necesario realizar un mantenimiento a través del tiempo, con la finalidad de que todo

siga funcionando correctamente. En realidad el mantenimiento para este sistema es una labor sencilla y que no requiere de grandes cambios, simplemente es una tarea de revisión de que todo siga estable.

En primer lugar, como varios de los componentes están alojados en servidores en la nube, como es el caso del propio servidor de la aplicación o de la base de datos, es necesario mantener un seguimiento sobre estos servicios para constatar que no han sufrido una caída de conexión inesperada por parte de las herramientas proveedoras. Por otro lado, de igual forma, es necesario revisar que la API externa Ergast siga operativa y que los *endpoints* y los datos ofrecidos no han variado con el tiempo. Esto es altamente improbable que ocurra, pero pese a ello, al tratarse de un componente externo al sistema es recomendable tener precaución y estar al tanto.

En cuanto a la base de datos, la información que se contiene en ella es válida para la presente temporada de la competición, siendo posible que tengan que añadirse o eliminarse pilotos, escuderías o eventos en función de los cambios que se produzcan en la realidad. En cualquier caso, estas modificaciones se efectuarían una vez al año, entre el final de una temporada y el inicio de la siguiente, y se dispondrían de unos dos o tres meses para realizarlo, en función del calendario de la Fórmula 1.

En cuanto a la aplicación en sí y el juego, a priori no se precisan modificaciones o actualizaciones rutinarias. En caso de que en un futuro se incorporen nuevas funcionalidades o se plantease la modificación de alguna de las desarrolladas, en ese caso sí que se debería de realizar una actualización del servidor y posterior despliegue en Heroku, y una actualización de la aplicación Android desarrollada para introducir los cambios, aunque en cualquier caso esto es una suposición de cara al futuro.

7

Conclusiones y líneas futuras

Este es el capítulo final, y en él se escriben las conclusiones sobre el trabajo realizado, tratando los resultados que se han obtenido al finalizarlo, los conocimientos que se han adquirido durante el proceso, las dificultades encontradas y posibles formas de extender el proyecto en un futuro.

7.1. Resultados obtenidos

Fruto de la realización de este Trabajo de Fin de Grado, se ha conseguido desarrollar una aplicación móvil basada en una arquitectura cliente-servidor que cumple con las características deseadas en el momento de plantear el proyecto y definir los objetivos.

Se han diseñado e implementado funcionalidades acorde al tipo de juego de género *fantasy* que se pretendía construir, respetando las ideas propuestas en el momento en el que surgió la oportunidad de realizar este proyecto. Además, todo el proceso de construcción, desde el diseño y modelado hasta las pruebas, ha estado

centrado en conseguir que todos los requisitos planteados en la fase de análisis se cumpliesen, tanto aquellos funcionales como no funcionales. Todo esto siguiendo la metodología propuesta en un principio y realizando las iteraciones que se han considerado oportunas para tener un sistema final lo más completo posible.

En cuanto a la aplicación en sí y su jugabilidad, el resultado se asemeja a lo que se esperaba conseguir en un principio. Las interfaces de usuario son atractivas para los jugadores, las funcionalidades cumplen con la finalidad del juego y se ha comprobado que la aplicación cumple unos estándares de usabilidad. Por otro lado, se ha comprobado que la aplicación es estable y que funciona exactamente como se pretende, por lo que se usar más allá de este proyecto. En definitiva, los resultados de este proyecto se consideran satisfactorios en todos los aspectos.

7.2. Conocimientos adquiridos

Durante el trascurso de este proyecto, se han afianzado conocimiento previos y se han adquirido otros nuevos fruto del trabajo con nuevas tecnologías. En primer lugar, si bien es cierto que sí se tenía experiencia con el lenguaje Java, no ocurría lo mismo con Python, cuyos conocimientos hasta el momento eran más básicos. De igual forma, la aplicación desarrollada en Android supera con creces cualquiera de los pequeños proyectos que se habían realizado previamente usando esta tecnología, lo cual ha sido un reto que se ha solventado con grandes resultados.

Por otro lado, el diseño de una arquitectura tan completa, teniendo la aplicación móvil por un lado, el servidor con su respectivo alojamiento por otro, la base de datos en una plataforma *cloud*, el uso de una API externa… también ha sido un reto, debido a que hasta el momento se había trabajado con estas tecnologías y plataformas por separado, pero el hecho de tener que combinarlas, hacer que funcionen entre sí y lograr

que el sistema creado y toda la arquitectura que lo sustenta sea estable ha resultado muy productivo para el aprendizaje.

Además, otro aspecto a tener en cuenta ha sido el desarrollo de los algoritmos de puntuación y modificaciones de valor, ya que para lograr su desarrollo ha sido necesario realizar un poco de ingeniería inversa, partiendo de las propuestas e ideas personales que se tenían al inicio del proyecto, y también analizando el funcionamiento de otros juegos similares con el fin de extraer conclusiones, ventajas e inconvenientes de éstos.

7.3. Dificultades encontradas

El uso algunas tecnologías nuevas o con las que no se había trabajado previamente ha supuesto un reto, tal y como se ha comentado en el apartado previo. Sin embargo, esto no se ha entendido como una dificultad sino como una oportunidad para aprender y afianzar nuevos conceptos. No obstante, en ocasiones, el hecho de tener que dedicar tiempo a entender el funcionamiento de la herramienta o tecnología ha ralentizado el proceso de desarrollo, ya que había que destinar esfuerzos a buscar información, ejemplos y documentación sobre frameworks o bibliotecas con los que no se había trabajado nunca, como puede ser el caso de Volley para Android. En cualquier caso, el tiempo de aprendizaje ya se había contemplado en el momento de distribuir las tareas y las etapas de este proyecto, y se había reservado un gran número de horas con esta finalidad.

Un suceso que sí supuso una dificultad, de hecho conllevó a un problema serio durante la realización de este proyecto, fue la pérdida total de la información que se encontraba almacenada en la base de datos. Esto ocurrió durante las primeras semanas de desarrollo, cuando ya se tenía preparada la arquitectura del sistema, se habían inicializado los entornos de programación y se había introducido información en la base

de datos. De hecho, en el momento en el que se produjo el fallo de seguridad, ya se había desarrollado gran parte de la API REST propia, por lo que esta dejó de estar operativa y se hizo imposible continuar con su implementación ya que el desarrollo se vio completamente paralizado por la brecha de seguridad.

El problema se originó debido a que el proyecto, con el código que se había programado hasta la fecha, se encontraba subido en un repositorio de GitHub, práctica que se ha realizado durante todo el proyecto con el fin de tener un control de versiones y de poder sincronizarlo la aplicación del servidor con Heroku. Lo que no se tuvo en cuenta es que en uno de los archivos que formaban el servidor se encontraba la dirección URL completa de la base de datos en MongoDB, a través de la cual se realizó un ataque de consultas con el objetivo de eliminar todos los documentos del clúster de MongoDB Atlas y pedir una recompensa económica para recuperarlos. Evidentemente esto último no se produjo, sino que se volvió a introducir la información en la base de datos y se modificó la URL usando un sistema de claves de Heroku para que no se repitiese el problema. Como es lógico, todo esto supuso un retraso en el desarrollo del proyecto.

Más tarde, se ha descubierto que este fallo de seguridad no fue tanto un problema propio sino un ataque generalizado a las plataformas GitHub y Heroku. En el mes de Abril de 2022, la plataforma Heroku sufrió un ataque que se saldó con el acceso no autorizado a los repositorios de GitHub de los usuarios que tenían ambas plataformas conectadas para desplegar sus servidores, como era el caso de este proyecto. Si bien es cierto que la URL se encontraba en el código careciendo de seguridad, habría sido prácticamente imposible que nadie accediese a ella de no haberse producido esta brecha en las plataformas de terceros, ya que la otra opción posible era que alguien tuviese acceso directo al proyecto en el repositorio de GitHub.

Como consecuencia de todo lo sucedido, la plataforma Heroku ha restringido la opción de sincronización de repositorios con GitHub, lo que ha provocado que se tenga

que cambiar el método de despliegue para este servidor, el cual se realiza a través de Heroku CLI hasta nuevo aviso [23].

7.4. Líneas futuras

Existen diversas funcionalidades que podrían complementar y mejorar la aplicación que se ha desarrollado en este proyecto, por ello se van a proponer algunas de ellas para posibles cambios en un futuro.

Una propuesta de mejora sería la incorporación de otra API externa que proporcione la información de los eventos, escuderías y pilotos del juego, sin necesidad de que estos datos se encuentren en la base de datos de la aplicación. Aunque las modificaciones entre temporadas son mínimas y el cambio en la base de datos sería ligero, esta actualización reduciría aún más el mantenimiento que hay que realizar en la aplicación.

Otra idea sería la incorporación de Inteligencia Artificial (IA) en el algoritmo de modificaciones de valor, para hacer de este un mecanismo más complejo y aún más justo respecto a los resultados y puntuaciones obtenidas. El principal lenguaje de programación que usa la IA es Python, por lo que teniendo en cuenta que también es el lenguaje escogido para el servidor y que el componente donde se produce la lógica de puntos y valores está también escrito en este lenguaje, no resultaría un impedimento el poder utilizar bibliotecas o herramientas para incluir esta modificación.

Por último, desde un punto de vista más visual, podrían añadirse gráficas relacionadas con las puntuaciones obtenidas o con los valores de los componentes del juego. De hecho, esta ha sido una opción valorada durante el desarrollo de la aplicación y que incluso se puso en marcha su implementación, pero finalmente fue descartada con el objetivo de destinar tiempo de desarrollo a realizar pruebas y revisiones de otros elementos ya implementados y que constaban de una mayor prioridad. Al fin y al cabo

esto sería una modificación de cara a la visualización de los datos, pero que no afectaría ni en las funcionalidades que se han desarrollado ni en el trascurso normal del juego, aunque sí podrían otorgar aún más vistosidad a la interfaz de la aplicación.

Referencias

[1] M. Berrocal, «Aumenta el interés por la Fórmula 1 entre los más jóvenes» *SoyMotor,* 23 Agosto 2021. [En línea]. Disponible: https://soymotor.com/noticias/aumenta-el-interes-por-la-formula-1-entre-los-mas-jovenes-989741.

[2] «Formula 1 announces audience figures for 2020» *Fórmula 1,* 8 Febrero 2021. [En línea]. Disponible: https://corp.formula1.com/formula-1-announces-audience-figures-for-2020/.

[3] F1 Fantasy Web [En línea]. Disponible: https://fantasy.formula1.com/.

[4] Atlassian [En línea]. Disponible: https://www.atlassian.com/es/agile/scrum.

[5] D. Calvo, «Metodología SCRUM» [En línea]. Disponible: https://www.diegocalvo.es/metodologia-scrum-metodologia-agil/.

[6] Python [En línea]. Disponible: https://www.python.org/.

[7] Flask [En línea]. Disponible: https://palletsprojects.com/p/flask/.

[8] MongoDB [En línea]. Disponible: https://www.mongodb.com/es/what-is-mongodb.

[9] Heroku [En línea]. Disponible: https://www.heroku.com/platform.

[10] Java [En línea]. Disponible: https://www.java.com/es/download/help/whatis_java.html.

[11] R. Adeva, «Qué es Android: todo sobre el sistema operativo de Google» 11 Marzo 2022. [En línea]. Disponible: https://www.adslzone.net/reportajes/software/que-es-android/.

[12] Volley Android [En línea]. Disponible: https://developer.android.com/training/volley?hl=es-419.

[13] Visual Studio Code [En línea]. Disponible: https://code.visualstudio.com/docs.

[14] Android Studio [En línea]. Disponible: https://developer.android.com/studio/features.

[15] Y. Fernández, «Qué es Github y qué es lo que le ofrece a los desarrolladores» 30 Octubre 2019. [En línea]. Disponible: https://www.xataka.com/basics/que-github-que-que-le-ofrece-a-desarrolladores.

[16] Postman [En línea]. Disponible: https://www.postman.com/product/tools/.

[17] Ergast Developer API [En línea]. Disponible: http://ergast.com/mrd/.

[18] I. García, «Qué es Modelo-Vista-Controlador» 30 Noviembre 2021. [En línea]. Disponible: https://carontestudio.com/blog/que-es-modelo-vista-controlador/.

[19] JUnit [En línea]. Disponible: https://junit.org/junit5/docs/current/user-guide/.

[20] Mockito [En línea]. Disponible: https://site.mockito.org/.

[21] Unittest Python [En línea]. Disponible: https://docs.python.org/3/library/unittest.html.

[22] C. Busquets, «Medir la usabilidad con el Sistema de Escalas de Usabilidad (SUS)». [En línea]. Disponible: https://www.uifrommars.com/como-medir-usabilidad-que-es-sus/.

[23] F. Salido, «Detalles sobre el ataque a Heroku» 10 Mayo 2022. [En línea]. Disponible: https://unaaldia.hispasec.com/2022/05/detalles-sobre-el-ataque-a-heroku.html.

Apéndice A

Manual de usuario

Esta sección recoge el manual de usuario que contiene las instrucciones para el manejo de la aplicación desarrollada. El manual está dividido según las funcionalidades principales de la aplicación, y en él se incluyen capturas del juego para seguir las instrucciones paso a paso.

A.1. Inicio de sesión y registro

Lo primero que aparece cuando se ejecuta la aplicación en el dispositivo móvil es la siguiente pantalla de carga que se observa en la Figura 34. Tras unos segundos, se accede al juego y se muestra la pantalla de inicio de sesión (ver Figura 35).

En esta pantalla existen dos opciones, por un lado, el usuario puede iniciar sesión introduciendo su nombre de usuario y su contraseña, y en caso de que la verificación sea correcta accederá al sistema; por otro lado, en caso de que el usuario sea nuevo y aún no se haya registrado, podrá hacerlo a través del botón Crear cuenta que le redirigirá a la pantalla de la Figura 36. En dicha pantalla, el usuario introduce sus datos, que pasarán un proceso de verificación, y en caso de validez se creará su cuenta, redirigiéndole a la pantalla anterior de Inicio de sesión.

En ambas secciones, tanto en la de inicio como la de registro, si se produce algún error con los datos introducidos o no es posible ejecutar la acción, se mostrarán mensajes informativos por pantalla, tal y como se puede observar en las dos capturas que se muestran a continuación (ver figuras 37 y 38).

Figura 34. Manual de usuario. Carga

Figura 35. Manual de usuario. Inicio

Figura 36. Manual de usuario. Registro

Figura 37. Manual de usuario. Error registro

Figura 38. Manual de usuario. Error inicio

A.2. Menú principal y perfil del usuario

Una vez que el usuario accede al sistema, se muestra la pantalla del menú principal (Figura 39). En ella existen diversos botones con diferentes funcionalidades que se abordarán a continuación. Antes de eso, en la esquina superior izquierda aparece un botón con la silueta de una persona, donde se podrá modificar el perfil del usuario. El funcionamiento de esta tarea es similar a la que se realiza durante la creación de una nueva cuenta, como se puede observar en la Figura 40. Por otro lado, en la esquina superior derecha del menú existe un botón para salir del juego, que cerrará la sesión activa y reconducirá al usuario a la pantalla de inicio del principio.

Figura 39. Manual de usuario. Menú

Figura 40. Manual de usuario. Perfil

En el menú principal se encuentran cuatro secciones (Pilotos, Equipos, Ligas y Eventos). Cada uno de estos botones, representados con imágenes, dirige al usuario a cada una de las secciones que se van a ir explicando a continuación.

A.3. Eventos

Cuando se accede a la pantalla de Eventos desde el menú principal, se muestra un listado con todos los Grandes Premios que se celebran durante la temporada (ver Figura 41). En dicho listado, aparece destacado en otro color el evento que se está celebrando actualmente, o en caso de no ser fin de semana de competición, el próximo evento que se celebrará. Cuando se pulsa en uno de ellos, se accede a una pantalla con información personalizada sobre ese evento, tal y como se puede ver en la imagen de la Figura 42. Además de los datos sobre el Gran Premio, también aparece visible un botón de Puntuaciones. Si se pulsa dicho botón, la aplicación muestra los puntos repartidos a cada componente (pilotos y escuderías) en ese evento, siempre y cuando ya haya sido disputado.

Figura 41. Manual de usuario. Eventos

Figura 42. Manual de usuario. Evento

Figura 43. Manual de usuario. Puntuaciones evento

A.4. Pilotos y escuderías

Estas dos secciones, Pilotos y Escuderías, son prácticamente idénticas entre sí, y a su vez son parecidas a Eventos ya que el funcionamiento de todas es similar, aunque la información y los detalles que se muestran varían en función de cada sección.

Cuando se accede a una de estas dos secciones, Piloto o Escudería, se muestra un listado que contiene los pilotos y escuderías reales que componen el campeonato de Fórmula 1, como se puede observar en la Figura 44 (para el caso de escuderías es similar). Si se pulsa sobre uno de los elementos, se abre una nueva ventana con información sobre el mismo, así como algunos datos de puntuación relacionados con el juego. A su vez, si se avanza pulsando el botón de Puntuaciones, se listan los registros se puntuación de ese componente en cada uno de los Grandes Premios celebrados, viendo así como se han conseguido sus puntos hasta el momento (Ver Figura 46).

Figura 44. Manual de usuario. Pilotos

Figura 45. Manual de usuario. Piloto

Figura 46. Manual de usuario. Puntuaciones piloto

A.5. Ligas

Cuando el usuario pulsa sobre el botón de ligas del menú principal, la aplicación le redirige a la pantalla principal de esta sección (Figura 47). Esta vista está compuesta por las ligas en las que participa el usuario y un botón para poder añadir una nueva liga. Si pulsa en él, el usuario verá otra pantalla donde dispone de dos mecanismos para añadir una nueva liga (Figura 48).

En la parte superior, el usuario puede crear una nueva liga introduciendo los datos de la misma y pulsando en el botón de crear. De esta forma creará una nueva competición desde cero y podrá invitar a rivales usando el código que se crea también junto a la liga. Por otra parte, en la zona inferior aparece la opción de unirse a una liga ya existente, que como se puede intuir se realiza introduciendo el código de acceso de la liga correspondiente a la que el usuario desea unirse.

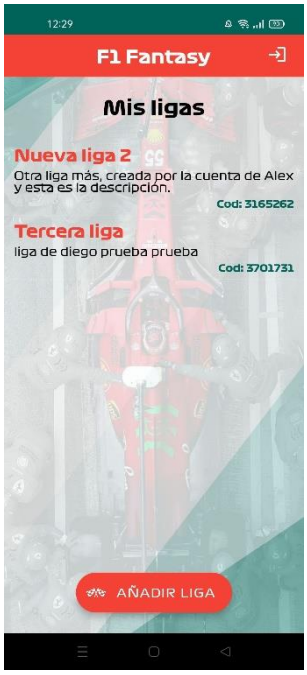

Figura 47. Manual de usuario. Ligas

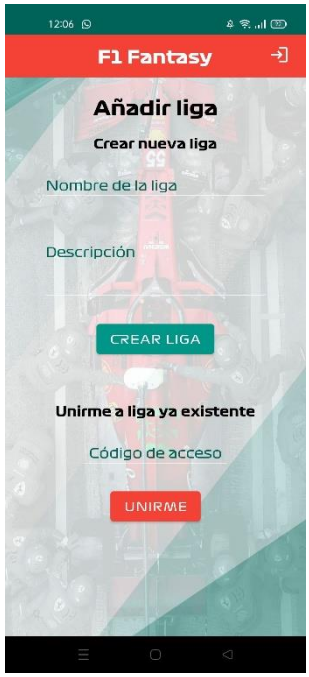

Figura 48. Manual de usuario. Añadir liga

De vuelta a la pantalla de ligas, si el usuario pulsa sobre una de las ligas en las que participa se abrirá otra sección que contendrá la clasificación de dicha liga, con los usuarios que participan en ella y sus respectivos puntos. Además, si se pulsa sobre uno de los participantes, el sistema muestra el historial de puntos de ese usuario en dicha liga dividido por rondas celebradas, como se puede ver en las siguientes imágenes (ver figuras 49 y 50).

Figura 49. Manual de usuario. Clasificación

Figura 50. Puntos participante

Estando en la pantalla de la liga, el usuario puede acceder a su equipo si pulsa sobre el botón inferior. Al hacerlo, el sistema muestra los componentes de su equipo, formado por cinco pilotos y una escudería. Como se puede observar en la Figura 51, es posible que el equipo no esté completo y falte algún miembro aún por fichar. En dicha imagen también se puede ver que los componentes del equipo aparecen bloqueados y no se pueden vender, como sí que ocurre en la imagen de la Figura 52. Eso ocurre durante la celebración de un Gran Premio, momento en el equipo permanece bloqueado para que no se puedan realizar modificaciones hasta que se complete el evento y se

asignen los puntos. Como se puede observar, en la esquina superior derecha aparece en todo momento el valor actual del equipo, cifra que no puede superar los 100M de limite salarial que existe en el juego.

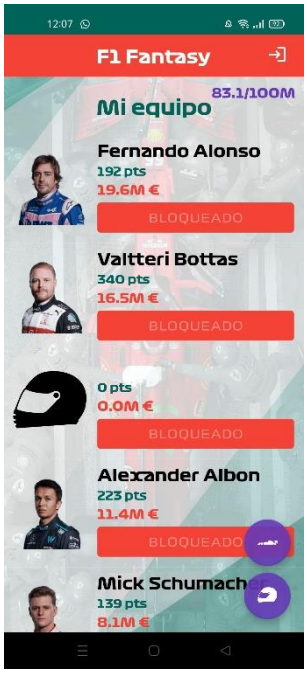

Figura 51. Manual de usuario. Equipo bloqueado

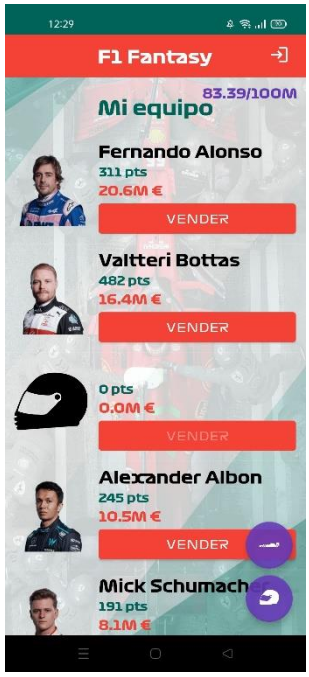

Figura 52. Manual de usuario. Equipo desbloqueado

Cuando se pulsa en un hueco vacío del equipo, o alternativamente en los botones de la esquina inferior derecha, se despliega un listado con todos los pilotos y escuderías. Este listado es parecido al que se ha comentado anteriormente en las secciones previas, pero con la diferencia de que desde él puede adquirirse nuevos miembros para el equipo (ver Figura 53). Para ello, el usuario debe pulsar en el botón de fichar y si se reúnen las condiciones necesarias, tanto de espacio en la plantilla como se margen económico, se añadirá el nuevo miembro al equipo. En caso contrario se muestra al usuario un mensaje informativo sobre el por qué no es posible como se puede ver en la Figura 54. Además, la opción de fichaje aparecerá restringida para aquellos pilotos o escuderías que ya forman parte de la plantilla actual del participante. En caso de pulsar sobre

algún componente, se muestra la información sobre el mismo, como es el caso de la Figura 55 si se pulsa una de las escuderías.

Figura 53. Manual de usuario. Fichaje

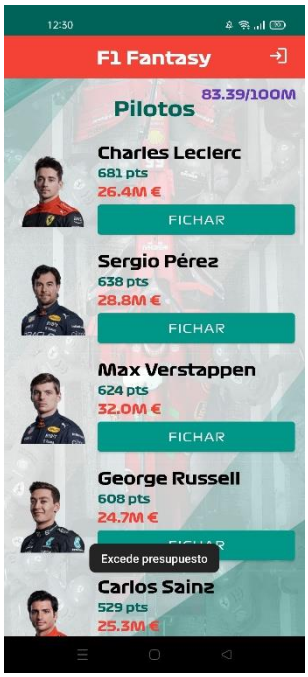

Figura 54. Manual de usuario. Límite presupuesto

Figura 55. Manual de usuario. Escudería

Apéndice B
Manual de despliegue

A continuación se describen la instrucciones para poner en marcha el sistema, tanto para desplegar el servidor en la plataforma Heroku, como para generar la aplicación cliente desde Android Studio.

B.1. Despliegue del servidor

En primer lugar, es necesario acceder a la plataforma GitHub mediante las credenciales otorgadas, ya que ahí se encuentra el repositorio con el código que forma la parte del servidor de la aplicación.

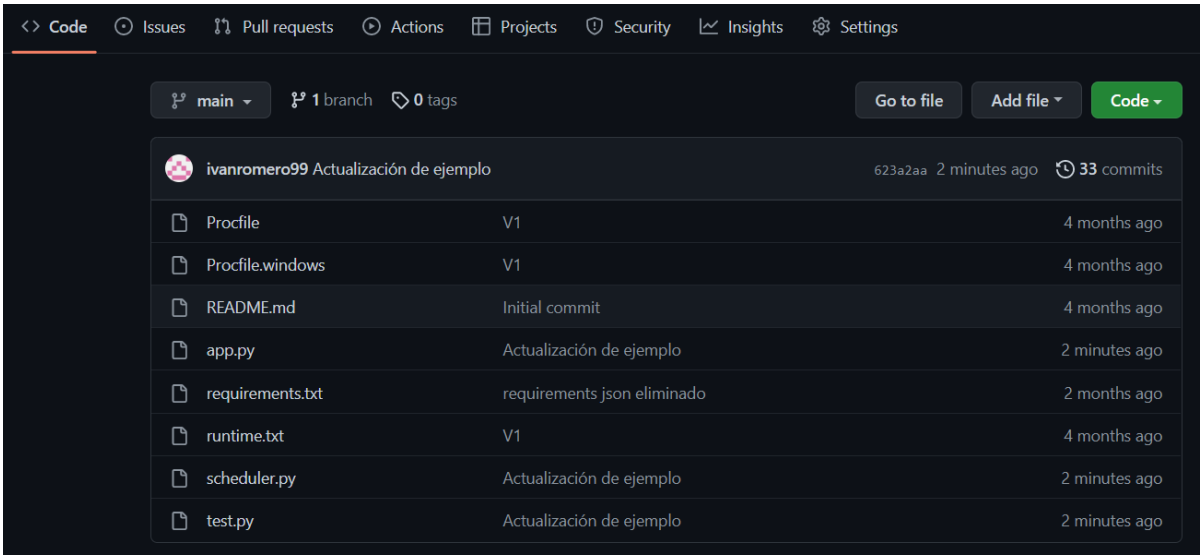

Figura 35. Manual de despliegue. Repositorio GitHub

Para realizar actualizaciones en el desarrollo del servidor, es necesario actualizar los ficheros que se hallan en el repositorio antes de realizar un despliegue en Heroku. Esto puede realizarse por línea de comandos usando Git (ver ejemplo en la Figura 57) o directamente subiendo los archivos al repositorio desde GitHub.

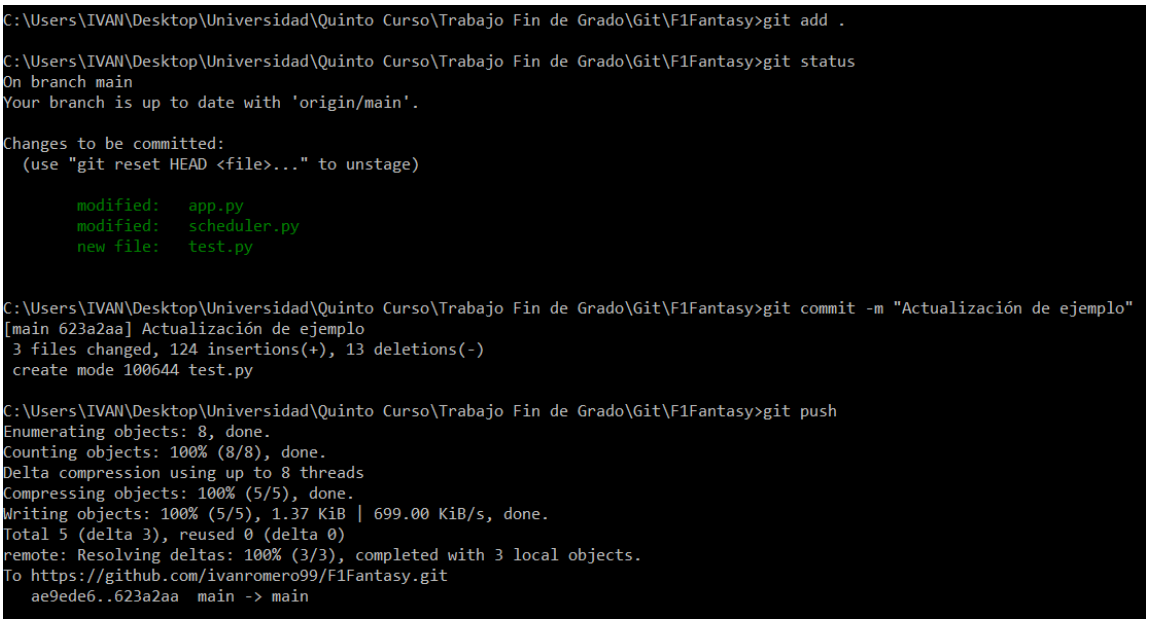

Figura 57. Manual de despliegue. Ejemplo actualización

Una vez que el código está preparado para ser desplegado, hay que acceder a Heroku a través de las credenciales y realizar un despliegue usando alguna de las opciones posibles. En este caso, para aprovechar el repositorio que alberga todo el código, se elige la opción de sincronización de Heroku con GitHub (ver Figura 58). En este caso, el repositorio ya se encuentra interconectado con la aplicación de Heroku, y así seguirá salvo que se decida desconectarlo en algún momento.

Para realizar el despliegue, es necesario bajar hasta encontrar la opción de "Deploy Branch", y tras pulsar en dicho botón simplemente hay que esperar a que la plataforma acceda al código y lo despliegue, emitiendo la información correspondiente por la consola. También existe la posibilidad de agilizar este proceso, dando acceso a Heroku para que despliegue automáticamente cualquier actualización de código que se

produzca en el repositorio de GitHub, si necesidad que tenga que realizarlo un usuario manualmente. Ambas opciones pueden verse en la Figura 59.

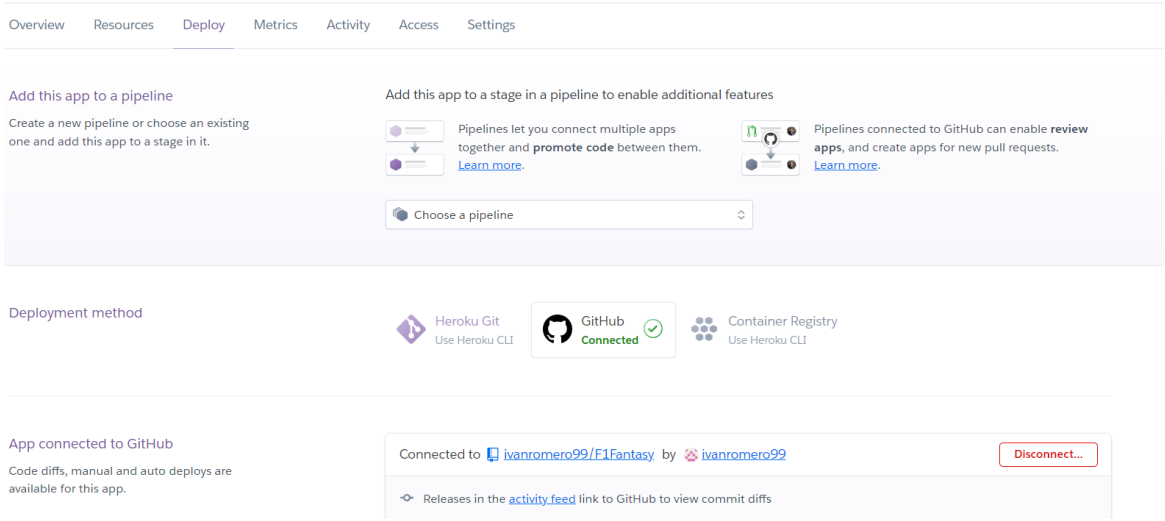

Figura 58. Manual de despliegue. Conexión Heroku-GitHub

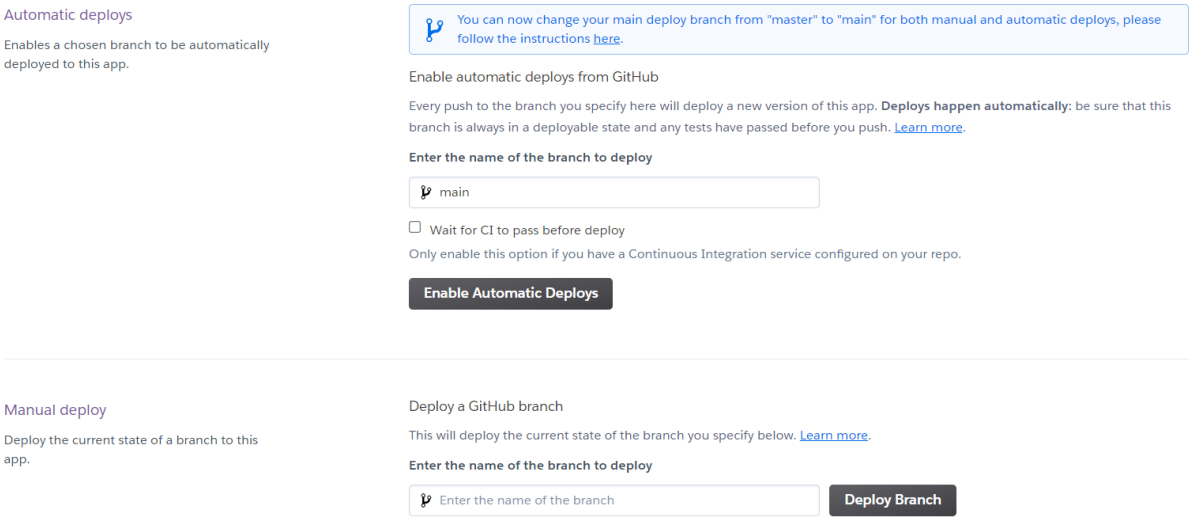

Figura 59. Manual de despliegue. Despliegue servidor

Con estos sencillos pasos el servidor quedará desplegado en la siguiente dirección URL: https://fantasyf1-ivan.herokuapp.com/, donde también se alojan los servicios REST ofrecidos a la aplicación móvil. Cabe mencionar, aunque no afecta directamente al despliegue del servidor, que también es recomendable acceder a MongoDB Atlas a

través de las credenciales, y comprobar que el clúster donde reside la base de datos está operativo y funcionando correctamente.

Figura 60. Manual de despliegue. Clúster MongoDB

B.2. Generación aplicación móvil

En cuanto a la aplicación móvil desarrollada en Android, ya se ha generado una APK que contiene la aplicación desarrollada. En cualquier caso, es posible generarla de nuevo desde Android Studio, para ello, en primer lugar es necesario acceder al proyecto de desarrollo. Una vez dentro, hay que desplegar la opción de "Build" en la barra de herramientas superior, y hacer click sobre "Build APK… " dentro de "Build Bundle", como puede observarse a continuación (Figura 61).

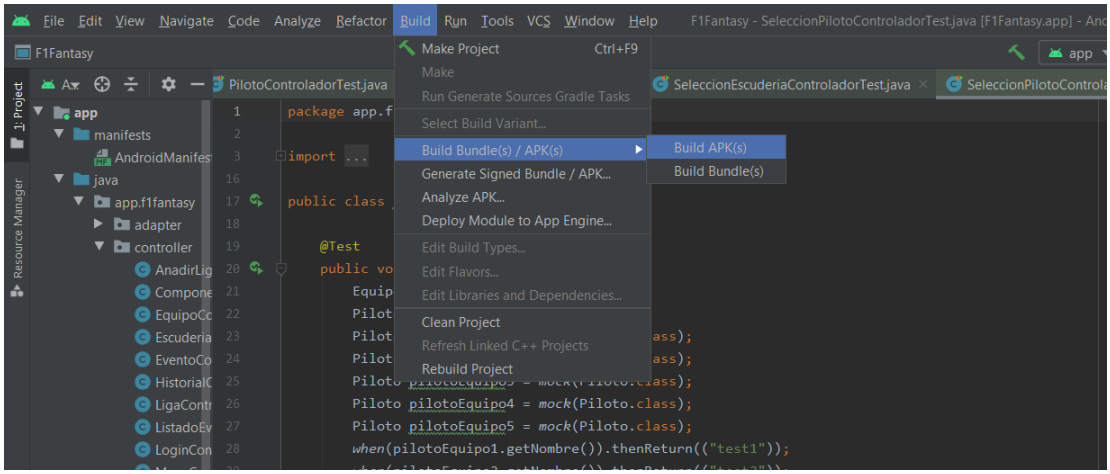

Figura 61. Manual de despliegue. Generar APK

También existe la posibilidad de desplegar la aplicación en Google Play. Esta no ha sido una opción contemplada hasta el momento, ya que por el momento la aplicación ha sido desarrollada con fines académicos.

Apéndice C

Usabilidad del sistema

En este apéndice se incluyen los cuestionarios realizados con usuarios durante las pruebas de usabilidad. Además, se incluyen también los comentarios recogidos durante la realización de la prueba y su relación con elementos del desarrollo de la aplicación.

C.1. Cuestionarios de usabilidad

Los dos primeros cuestionarios que se muestran se corresponden con usuarios con cierta experiencia y conocimientos tanto de la temática, Fórmula 1, como de este tipo de juegos. El tercer cuestionario pertenece a un usuario que desconoce el funcionamiento del juego pero que sí es seguidor de la competición deportiva. Por último, el cuestionario final lo ha completado un usuario que no está familiarizado con ninguno de estos aspectos.

C.1.1. Cuestionario 1

Responda a las preguntas en base al siguiente criterio:

(1) Nada de acuerdo

(2) En desacuerdo

(3) Indiferente

(4) De acuerdo

(5) Muy de acuerdo

	1	2	3	4	5
1. Creo que me gustaría usar esta aplicación con frecuencia.					X
2. Creo que la aplicación es innecesariamente compleja.		X			
3. Pienso que la aplicación es fácil de usar.				X	
4. Creo que necesitaría la ayuda de una persona para poder utilizar esta aplicación.		X			
5. Creo que las diversas funciones en esta aplicación estaban bien integradas.				X	
6. Creo que hay demasiada inconsistencia en la aplicación.	X				
7. Pienso que la mayoría de la gente aprendería a usar esta aplicación en poco tiempo.		X			
8. La aplicación me pareció muy engorrosa de usar.		X			
9. Me sentí muy seguro usando la aplicación.				X	
10. Necesitaba aprender muchas cosas antes de poder comenzar con esta aplicación.		X			

Tabla 23. Cuestionario SUS. Usuario 1

Respuestas enunciados impares: (5 + 4 + 4 + 2 + 4) =>19 – 5 = 14

Respuestas enunciados pares: (2 + 2 + 1 + 2 + 2) => 25 – 9 = 16

Cálculo del SUS: (14 + 16) * 2,5 = 75

C.1.2. Cuestionario 2

Responda a las preguntas en base al siguiente criterio:

(1) Nada de acuerdo

(2) En desacuerdo

(3) Indiferente

(4) De acuerdo

(5) Muy de acuerdo

	1	2	3	4	5
1. Creo que me gustaría usar esta aplicación con frecuencia.				X	
2. Creo que la aplicación es innecesariamente compleja.	X				
3. Pienso que la aplicación es fácil de usar.				X	
4. Creo que necesitaría la ayuda de una persona para poder utilizar esta aplicación.	X				
5. Creo que las diversas funciones en esta aplicación estaban bien integradas.			X		
6. Creo que hay demasiada inconsistencia en la aplicación.	X				
7. Pienso que la mayoría de la gente aprendería a usar esta aplicación en poco tiempo.			X		
8. La aplicación me pareció muy engorrosa de usar.		X			
9. Me sentí muy seguro usando la aplicación.			X		
10. Necesitaba aprender muchas cosas antes de poder comenzar con esta aplicación.	X				

Tabla 24. Cuestionario SUS. Usuario 2

Respuestas enunciados impares: (4 + 4 + 3 + 3 + 3) => 17 – 5 = 12

Respuestas enunciados pares: (1 + 1 + 1 + 2 + 1) => 25 – 6 = 19

Cálculo del SUS: (12 + 19) * 2,5 = 77,5

C.1.3. Cuestionario 3

Responda a las preguntas en base al siguiente criterio:

(1) Nada de acuerdo

(2) En desacuerdo

(3) Indiferente

(4) De acuerdo

(5) Muy de acuerdo

	1	2	3	4	5
1. Creo que me gustaría usar esta aplicación con frecuencia.				X	
2. Creo que la aplicación es innecesariamente compleja.		X			
3. Pienso que la aplicación es fácil de usar.			X		
4. Creo que necesitaría la ayuda de una persona para poder utilizar esta aplicación.				X	
5. Creo que las diversas funciones en esta aplicación estaban bien integradas.					X
6. Creo que hay demasiada inconsistencia en la aplicación.	X				
7. Pienso que la mayoría de la gente aprendería a usar esta aplicación en poco tiempo.		X			
8. La aplicación me pareció muy engorrosa de usar.			X		
9. Me sentí muy seguro usando la aplicación.			X		
10. Necesitaba aprender muchas cosas antes de poder comenzar con esta aplicación.		X			

Tabla 25. Cuestionario SUS. Usuario 3

Respuestas enunciados impares: (3 + 3 + 5 + 2 + 4) => 17 – 5 = 12

Respuestas enunciados pares: (2 + 4 + 1 + 3 + 2) => 25 – 12 = 13

Cálculo del SUS: (12 + 13) * 2,5 = 62,5

C.1.4. Cuestionario 4

Responda a las preguntas en base al siguiente criterio:

(1) Nada de acuerdo

(2) En desacuerdo

(3) Indiferente

(4) De acuerdo

(5) Muy de acuerdo

	1	2	3	4	5
1. Creo que me gustaría usar esta aplicación con frecuencia.		X			
2. Creo que la aplicación es innecesariamente compleja.		X			
3. Pienso que la aplicación es fácil de usar.		X			
4. Creo que necesitaría la ayuda de una persona para poder utilizar esta aplicación.				X	
5. Creo que las diversas funciones en esta aplicación estaban bien integradas.				X	
6. Creo que hay demasiada inconsistencia en la aplicación.	X				
7. Pienso que la mayoría de la gente aprendería a usar esta aplicación en poco tiempo.				X	
8. La aplicación me pareció muy engorrosa de usar.		X			
9. Me sentí muy seguro usando la aplicación.				X	
10. Necesitaba aprender muchas cosas antes de poder comenzar con esta aplicación.			X		

Tabla 26. Cuestionario SUS. Usuario 4

Respuestas enunciados impares: (2 + 2 + 4 + 4 + 4) =>16 – 5 = 11

Respuestas enunciados pares: (2 + 4 + 1 + 2 + 3) => 25 – 12 = 13

Cálculo del SUS: (11 + 13) * 2,5 = 60

C.2. Sugerencias de usuarios

A continuación se muestra una recopilación de diferentes comentarios y sugerencias sobre errores y problemas que se han encontrado los usuarios durante las pruebas de evaluación de la aplicación, los cuales se han utilizado para mejorar el sistema final. Algunos de estos fallos ya habían sido contemplados durante el desarrollo y simplemente estaban a la espera de una corrección en las próximas iteraciones, ya que los usuarios han probado diferentes versiones de la aplicación durante el desarrollo. No obstante, aunque ya estuviesen detectados, estos comentarios se anotaron para dejar constancia del proceso de actualización de errores a partir de las pruebas.

- Problemas con el color de los cuadros de texto usando el modo oscuro del móvil.
- Problemas con la resolución de la imagen de carga, se redimensiona y se estrecha más de lo esperado.
- Se permite crear una cuenta con un nombre de usuario ya utilizado previamente y a su vez de muestra fallo. Se ejecutan ambas posibilidades simultáneamente.
- Bug con la información del apartado “Sabías Qué”. Salen varios datos que se superponen durante uno o dos segundos hasta que aparece el dato definitivo.
- Escasez de información al fallar el registro. Considera necesario incluir alguna indicación concreta en el campo erróneo.
- Selección de un hueco libre en el equipo poco intuitiva. Cree que es más útil pulsar en el hueco vacío para que se abra el listado, en lugar de pulsar sobre el botón inferior derecho.
- Problema con el tamaño de los nombres de pilotos. Corregir tamaño de fuente en aquellos nombres de larga extensión.
- Poco práctico que se mantenga visible el botón “Vender” en los huecos del equipo que están vacíos, aunque dicho botón esté desactivado.

UNIVERSIDAD DE MÁLAGA | uma.es

E.T.S de Ingeniería Informática
Bulevar Louis Pasteur, 35
Campus de Teatinos
29071 Málaga

E.T.S. DE INGENIERÍA INFORMÁTICA