Universidad de **Valladolid**

Escuela de Ingeniería Informática

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática
Mención en Ingeniería de Software

Migración de aplicación Android con Google Play Games a API REST

Autor:
Juan García Diéguez

Universidad de **Valladolid**

Escuela de Ingeniería Informática

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática
Mención en Ingeniería de Software

Migración de aplicación Android con Google Play Games a API REST

Autor:
Juan García Diéguez

Tutores:
Cristian Tejedor García
Mario Corrales Astorgano

A mi familia. Gracias a vosotros pude comenzar esta etapa y cerrarla de la mejor manera posible.

Agradecimientos

Varias personas han ayudado a que este proyecto saliera adelante. En primer lugar, tengo que agradecer el esfuerzo y dedicación continua a mis tutores, Cristian Tejedor García y Mario Corrales Astorgano. Gracias a sus continuas ayudas y revisiones, tanto del código de la aplicación como del documento el proyecto ha podido seguir un camino adecuado hasta su finalización. También es necesario dar las gracias a mi familia y amigos, por el apoyo y ánimos de todos estos meses de trabajo.

Resumen

Debido al cierre de servicios multijugador de la API de Google Play Games en marzo de 2020, muchas aplicaciones que usaban este servicio quedaron inutilizables. Una de estas aplicaciones era Clash of Pronunciations (CoP), una aplicación Android multijugador para la práctica de la pronunciación en diferentes idiomas. Su principal idea es favorecer este aprendizaje de una manera divertida y entretenida, incluyendo desafíos en partidas de juego multijugador y escalando puestos en un ranking. Durante el curso 2019-2020 se implementó de cero una API en otro Trabajo Fin de Grado para reemplazar las funcionalidades de Google Play Games integradas en CoP. A lo largo de este proyecto se adapta el código de CoP para utilizar las llamadas de la nueva API y se actualiza el código de la aplicación Android a la última versión disponible para garantizar su correcto funcionamiento en los dispositivos actuales siguiendo la normativa de Android.

Abstract

Many web and mobile applications were useless after the shutdown of the Google Play Games API multiplayer services since May 2020. Clash of Pronunciations (CoP), a multiplayer Android application for pronunciation practice in different languages, was one of the applications affected by the shutdown. The main goal of CoP is to favor pronunciation training in a fun and entertaining way, including multiplayer games and a ranking. During the 2019-2020 course, a new API was carried out from scratch in another bachelor's thesis to substitute the Google Play Games' functionalities integrated into CoP. Throughout this project, CoP's source code is adapted to use this new API. Furthermore, the code is upgraded to the last available Android version in order to ensure compatibility in current mobile devices following Android guidelines.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

1.1. Contexto

En el año 2016, Cristian Tejedor García, tutor de este TFG, y alumno del Máster en Ingeniería Informática en ese momento, junto con sus tutores David Escudero Mancebo y César González Ferreras decidieron realizar un TFM [1] que fomentara el entrenamiento de la pronunciación extranjera mediante el uso de una nueva aplicación fundamentada en la gamificación (del término ingés *gamification*) de la pronunciación y discriminación de palabras en diferentes idiomas. Con esta idea en mente se creó TipTopTalk! [1].

Durante el curso 2016-2017, se decidió dar continuidad al proyecto añadiendo un componente muy importante para mejorar la gamificación del juego. En concreto, a mediados de 2017 se añadió la modalidad multijugador a la aplicación, lo que permitía crear partidas en las que se competía con otros usuarios reales con similares puntuaciones, lo que daba a la aplicación de aprendizaje de idiomas un punto más de diversión y entretenimiento. Para poder crear esa nueva modalidad de juego se optó por el uso de la API de Google Play Games [2], que permitía crear partidas, emparejamientos, mantenía la base de datos de usuarios y creaba rankings en función de las puntuaciones obtenidas. Además de estas labores, es importante destacar el cambio de nombre que sufrió la aplicación, pasando de TipTopTalk! al actual Clash of Pronunciations (CoP) [3]. A continuación, los tutores llevaron a cabo un trabajo de actualización y adaptación del código de CoP que permitió utilizar la aplicación en uno de los experimentos de investigación de la tesis de Cristian Tejedor [4], en el que participaron usuarios reales y se reportaron mejoras de pronunciación extranjera y de motivación de los jugadores [5]. El problema surgió casi tres años después (2020) cuando Google decidió dejar de mantener la API y la aplicación perdió la funcionalidad online que tanto había mejorado la gamificación.

Como consecuencia de lo anterior, en el curso 2019-2020 se llevó a cabo otro TFG [6] en el que se implementó una API en Node.js con 14 llamadas o *endpoints* que permitiría volver a dar continuidad a la aplicación CoP sin depender de Google Play Games. Solo quedaba un paso más a realizar para que CoP volviera a estar disponible para su uso y era el de migrar todas las antiguas llamadas y código que usaba la API de Google Play Games a la nueva API creada y actualizar todo el código obsoleto que se

había quedado atrasado [7]. En ese punto es donde nos encontramos y en lo que se basa este TFG. En la Figura 1.1 podemos ver de forma resumida la evolución del código fuente desde su inicio hasta la actualidad, para poner en contexto al lector, tal y como se ha explicado anteriormente.

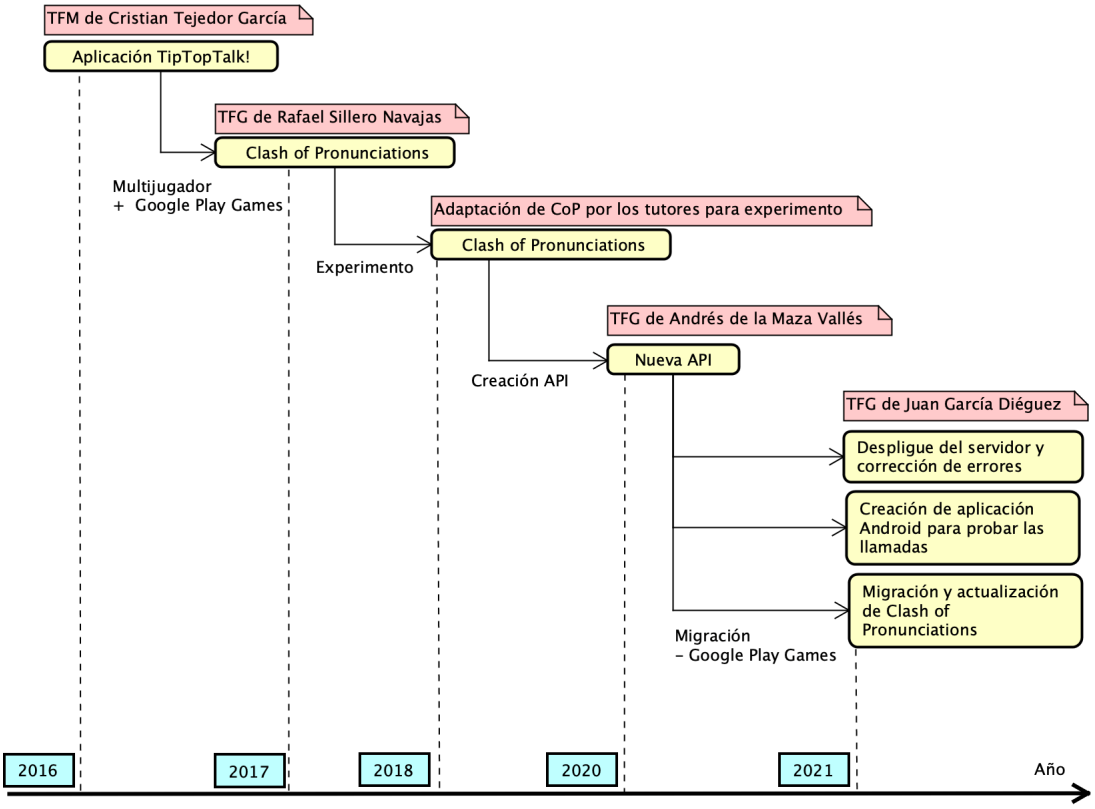


Figura 1.1: Trabajos y actualizaciones de CoP a lo largo del tiempo

1.2. Motivación

Los continuos cambios y avances en la tecnología, y concretamente en Android, hacen que sea totalmente necesario el mantenimiento y puesta al día de las aplicaciones ya creadas. En concreto, Google obliga a los desarrolladores a actualizar el código de compilación de las aplicaciones a la última versión. Esto conlleva cambios en la compatibilidad hacia atrás del código, obteniendo partes desfasadas (deprecated) y otras totalmente nuevas que deben adaptarse tanto a las versiones anteriores como a las nuevas [8]. Actualmente (principios de 2021) la versión de compilación de Android es la 11 (API LEVEL 30). La primera fue lanzada en el año 2008, por lo que cada año cambia. A esto hay que sumarle el hecho de que CoP no es funcional a fecha de empezar este TFG, inicio de 2021, debido a la eliminación de la API de Google Play Games, como ya hemos comentado en el apartado 1.1.

La motivación es, por tanto, recuperar la aplicación CoP y hacerla funcional de nuevo mediante la migración de todas las llamadas [7] creadas en el Trabajo Fin de Grado de Andrés de la Maza Valles [6] junto a actualizar y realizar un mantenimiento de todos los problemas que puedan aparecer en la aplicación como consecuencia de su actualización a la última versión de Android.

1.3. Estado de la cuestión

1.3.1. TipTopTalk!

Como se ha comentado en la sección 1.1, esta fue la aplicación original y de la que parte CoP. Fue desarrollada por Cristian Tejedor García [1] a finales del curso 2015-2016 y cuyo fin principal era el de aportar diversión a la práctica de la pronunciación en diferentes idiomas, haciendo del proceso de aprendizaje un proceso entretenido y nada tedioso.

Es una aplicación móvil para dispositivos Android que integra la posibilidad de mejorar la pronunciación en varios idiomas entre los que encontramos el español, inglés y chino. Presenta diferentes juegos de pronunciación, discriminación y exposición que otorgan puntos y logros y los usuarios ven representada esa puntuación en un ranking, favoreciendo la competitividad de los mismos y, por tanto, el entrenamiento en el idioma escogido. En todo momento se guarda información acerca de la actividad de los usuarios con la aplicación mediante ficheros de log. En las siguientes imágenes podemos ver la interfaz de usuario que presentaba TipTopTalk! (ver Figura 1.2).

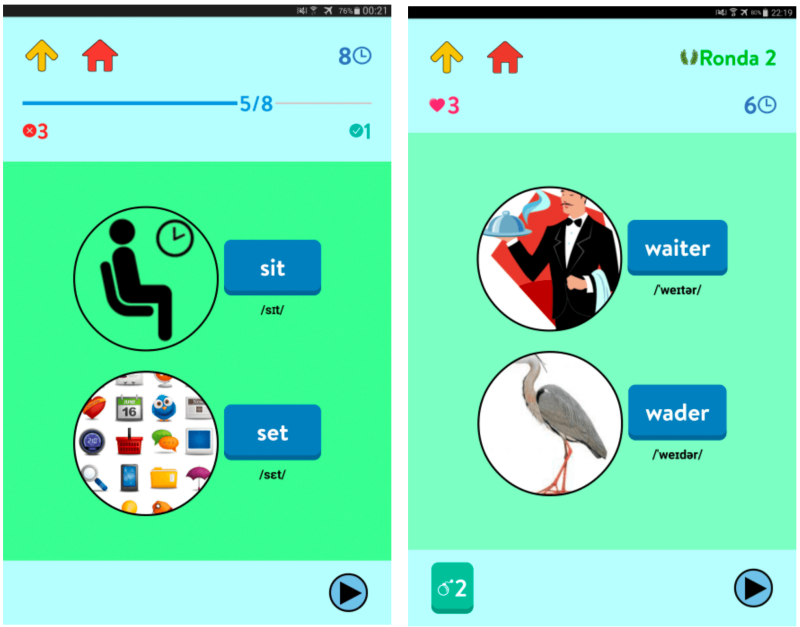


Figura 1.2: Vistas de la interfaz de TipTopTalk!

Como puede verse en la Figura 1.2, la interfaz es casi idéntica a la que encontraremos en nuestra aplicación (CoP) ya que TipTopTalk! fue la semilla del proyecto de CoP y de la que partió nuestra aplicación unos años después.

1.3.2. Clash of Pronunciations

CoP arrancó con el desarrollo del Trabajo Fin de Grado de Rafael Sillero Navajas [9]. La parte principal de su código fuente está basado en la antigua TipTopTalk! y durante ese proyecto se añade parte de la funcionalidad que tiene ahora. Junto con los desarrollos llevados a cabo durante ese proyecto

inicialmente, fueron necesarias nuevas iteraciones para poder utilizar la aplicación en un experimento de investigación [5, 4].

Como resultado de ese trabajo, CoP se convirtió en una aplicación para la práctica de la pronunciación en diferentes idiomas y con capacidad de juego multijugador, en el que se podían crear partidas para enfrentarse a otros jugadores a los que se les otorgaban puntuaciones en función de los aciertos o fallos de los diferentes ejercicios. Las principales funcionalidades que ofrece CoP en la actualidad son:

- Registro: los usuarios pueden crear un perfil de usuario para poder jugar (ver Figura 1.3). Este perfil tiene un nick de usuario único, junto con el mail y la imagen de perfil. El perfil de usuario se asocia a diferentes métricas como son las diferentes puntuaciones de las partidas o el ranking de puntuaciones, que muestran la puntuación de cada usuario.
- Login: el usuario puede acceder mediante su nick y contraseña (ver Figura 1.3).
- Ranking: la aplicación presenta un ranking de usuarios en función de sus puntuaciones totales y ordenado de mayor a menor puntuación. El usuario que ha iniciado sesión es resaltado del resto (ver Figura 1.5).
- Partidas multijugador: se pueden crear partidas multijugador, escogiendo antes a los oponentes contra los que se desea jugar (ver Figura 1.4). Esta partida será compartida con estos oponentes en su lista de partidas pendientes para que puedan jugarla y completar todas las puntuaciones. Cuando la partida ha sido completada por todos, se añaden una serie de puntos extra en función del creador y de las posiciones en el ranking antes de jugar y se incrementa la puntuación total de cada usuario. Existe un límite de 30 partidas jugadas al día. Una vez superado dicho límite no se podrá volver a jugar partidas multijugador hasta el siguiente día.
- Comprobar partidas pendientes: muestra una lista de las partidas creadas por otros usuario y que no han sido jugadas aún (ver Figura 1.4).
- Historial de partidas: muestra una lista de las partidas ya jugadas por el usuario en cuestión. Si se pulsa en alguna de las partidas se muestra información extra.
- Logros: el usuario puede ganar logros a medida que supera ciertas metas en la aplicación. Estas metas pueden ser superar hitos de puntuación total o de partidas jugadas (ver Figura 1.5).
- Modificación del perfil: el perfil puede ser editado por el usuario (ver Figura 1.3). Se puede editar el mail, la imagen de perfil o la contraseña.
- Partidas de entrenamiento: junto con las partidas multijugador existe también otra modalidad de juego que consiste en la práctica individual. En este caso no hay límite de partidas diarias (ver Figura 1.4).

CoP ofrecía una serie de modalidades de juegos, tanto de práctica individual como el multijugador ya comentado. Dentro de la modalidad de práctica individual podemos encontrar los modos de juego exposición, discriminación y pronunciación y dentro del modo multijugador encontramos el modo infinito, que hereda el nombre de TipTopTalk! ya que en ese juego la ronda no acababa hasta perder todas las vidas, pero en CoP se limitó únicamente a 9 rondas por partida.

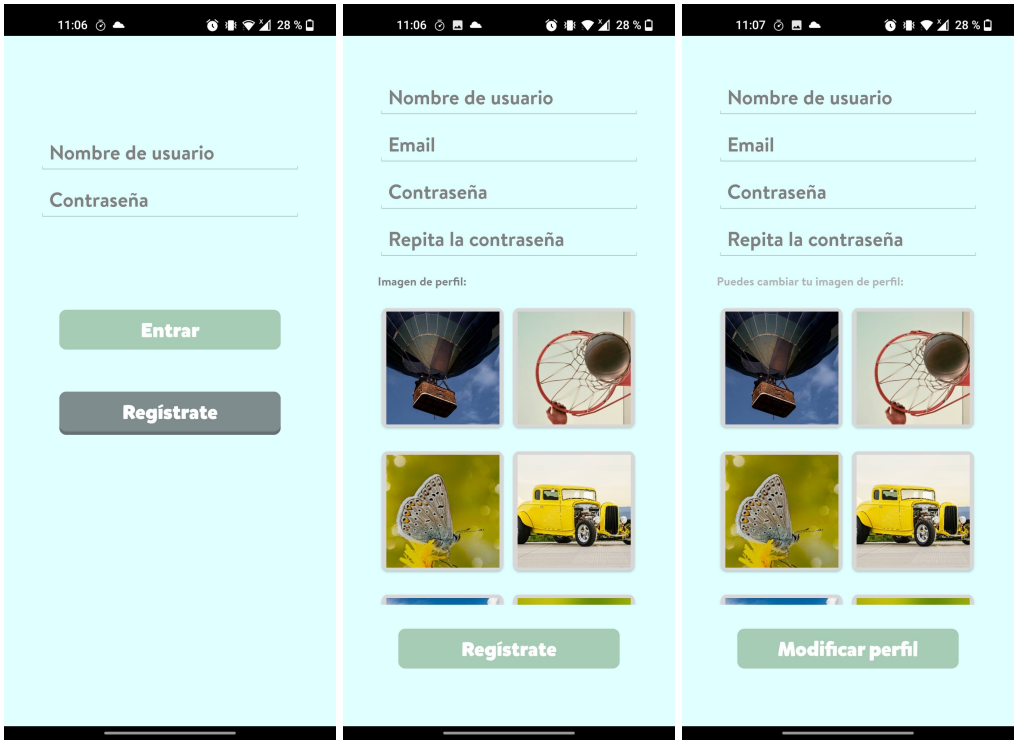


Figura 1.3: Interfaz de CoP para el login, registro y modificación del perfil

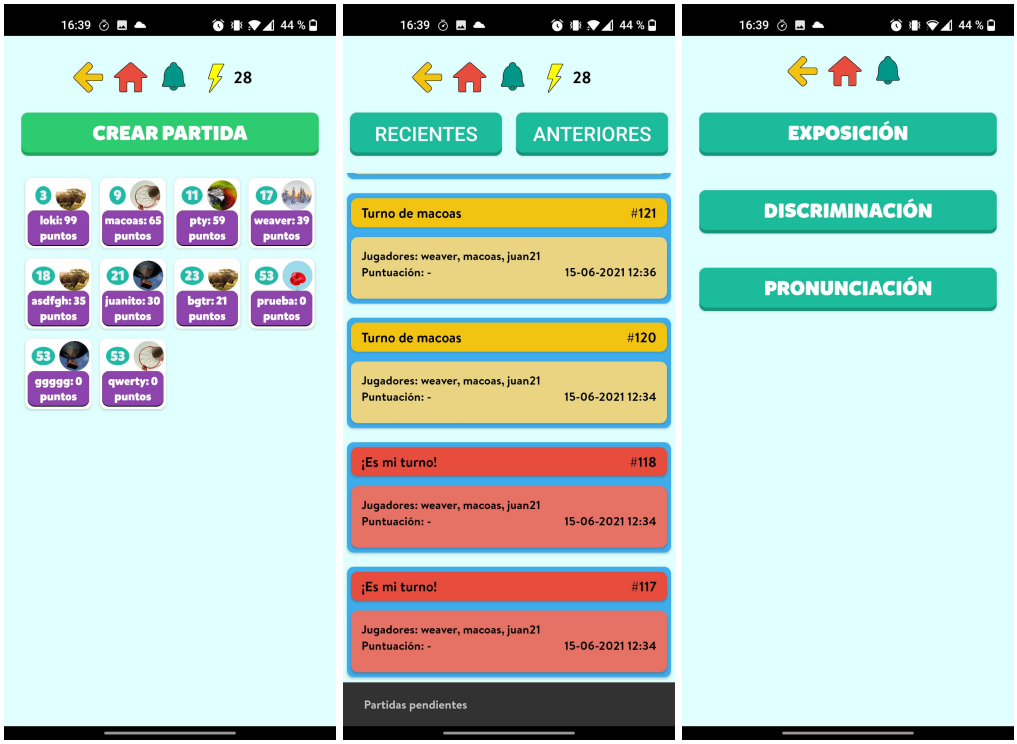


Figura 1.4: Interfaz de CoP de la pantalla de crear partida multijugador, partidas pendientes y entrenamiento

Figura 1.5: Interfaz de CoP del menú principal, lista de logros y ranking

Modo Exposición

El dispositivo reproduce un par de palabras de manera alterna múltiples veces (ver Figura 1.6). Cuando la reproducción finaliza el usuario debe pulsar en cada una de las palabras y grabar un audio pronunciando cada una de ellas. Al finalizar, el audio será reproducido justo después del audio original. De esta manera el usuario podrá escucharlo y comparar el audio real con el suyo.

Modo Discriminación

Se reproduce un audio de una palabra mientras se muestran en pantalla dos posibles respuestas. Solo una de ellas es la palabra pronunciada y el usuario debe pulsar sobre la que cree haber escuchado (ver Figura 1.6).

Modo Pronunciación

Se presentan dos palabras en pantalla y el usuario debe tratar de pronunciar cada una de ellas grabándose un audio para cada una (ver Figura 1.6). El audio se valida mediante ASR (Automatic Speech Recognition) y se genera una lista de palabras con un índice de confianza para cada una de ellas. El tamaño de esta lista depende del nivel de dificultad seleccionado. Si la palabra a pronunciar (o su homófona) se encuentra entre las de la lista la respuesta se da como válida y si no no.

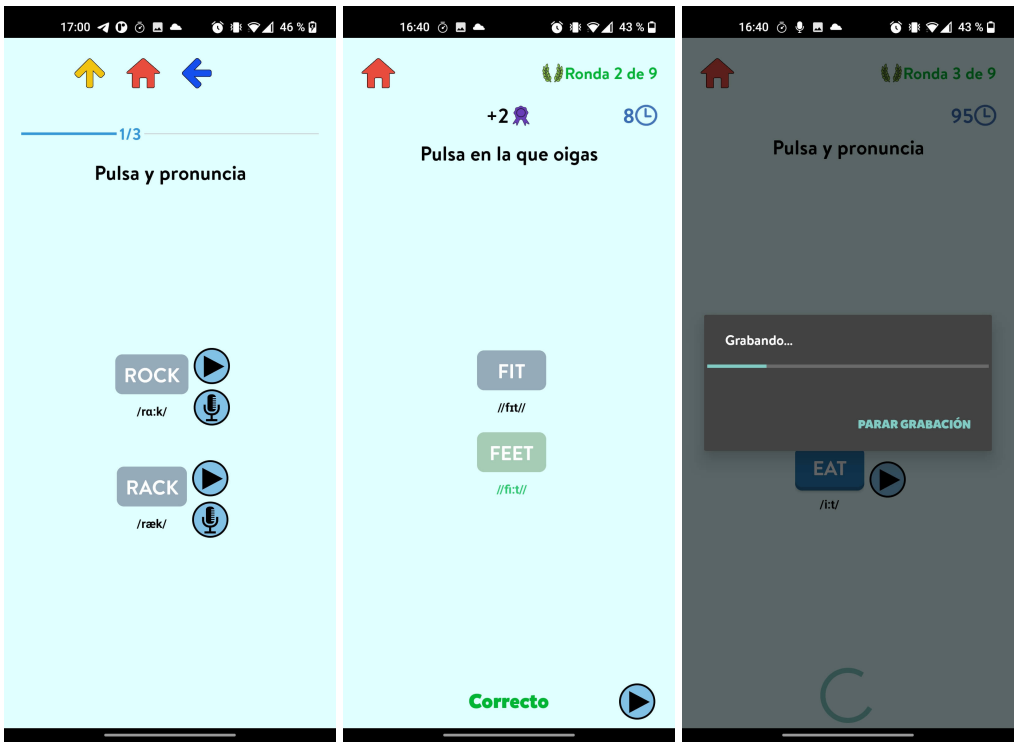


Figura 1.6: Interfaz de CoP de los entrenamientos Exposición, Discriminación y Pronunciación

Modo Multijugador

El modo multijugador mezcla los modos de Discriminación y Pronunciación en uno solo. En él se usa una misma lista de palabras para todos los participantes de la partida y se ejecutan una serie de rondas (9 actualmente) en las que se sigue el siguiente patrón un total de 3 veces: 2 rondas de Discriminación y 1 ronda de Pronunciación (ver Figura 1.7). Al finalizar la partida se otorgan una serie de puntos base en función de las respuestas acertadas, fallos y ayudas obtenidas. Cuando todos los usuarios juegan se añaden una serie de puntos extra que dependen de las posiciones en el ranking y las puntuaciones obtenidas y se suman todas las puntuaciones a los usuarios, actualizando su ranking.

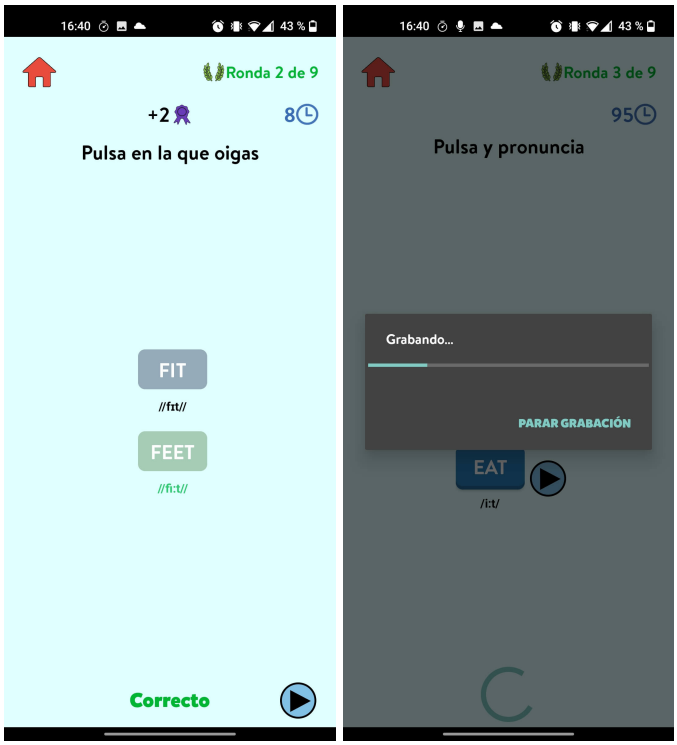


Figura 1.7: Interfaz de CoP de los dos modos de juego de las partidas multijugador. A la izquierda una ronda de Discriminación y a la derecha Pronunciación

1.3.3. Google Play Games

Todo lo anteriormente comentado de CoP se apoyaba en la funcionalidad que ofrecía la API de Google Play Games para servicios multijugador[2], desde el inicio de sesión y mantenimiento de usuarios, pasando por las partidas multijugador, rankings, logros y puntuaciones, estadísticas personales y un sinfín de funcionalidades extra como la grabación de vídeos de partidas para publicarlas en YouTube.

El problema surgió aproximadamente tres años después de la creación del Trabajo Fin de Grado de Rafael Sillero Navajas [9]. Google comunicó el cierre de sus APIs multijugador de Google Play Games a partir del día 31 de marzo de 2020 [10]. Como consecuencia de esto la aplicación quedó inservible y necesitó una migración completa de toda la funcionalidad multijugador así como de la gestión de sus usuarios y estadísticas. A continuación, se muestran las principales funcionalidades que ofrecía la API

multijugador de Google Play Games:

1. **Perfil de usuario**. Cada jugador tiene un perfil asociado que almacena diferentes estadísticas como experiencia global, aplicaciones jugadas o tiempo de juego entre otros. Estas pueden ser usadas para diferentes acciones de marketing.

2. **Logros**. Representan metas que pueden ser alcanzadas por los jugadores al completar diferentes acciones en el juego. En Clash of Pronunciations, un ejemplo claro de esto es obtener los logros a medida que se sobrepasan ciertos valores de puntuación.

3. **Ranking**. Listado de los jugadores de la aplicación ordenados por su puntación total u otra serie de métricas que puedan ser usadas en otros juegos.

4. **Partidas guardadas**. Registra en la nube la información de las partidas para evitar pérdidas debido a desinstalación de la aplicación en el dispositivo o de cambio del mismo.

5. **Partidas multijugador en tiempo real**. Conecta a varios jugadores en tiempo real para transmitir a los demás las acciones que realiza uno de los usuarios en su dispositivo. Esto es necesario en juegos que requieren una realimentación en tiempo real de lo que cada jugador está realizando. Se puede ver en juegos como ”Brawl Stars” [11].

6. **Partidas multijugador por turnos**. Alternativa a las partidas en tiempo real. En este modo solo un usuario puede modificar, de manera simultánea, el estado de la partida.

1.3.4. Android y API REST

REST es el acrónimo de Representational State Transfer (transferencia de representación de estado en español). Su característica principal es que no tiene estado, lo cual significa que el servicio pierde todos los datos entre dos llamadas diferentes. El estado es necesario que lo mantenga el cliente. Si se requiere que el servicio mantenga cierta información será necesario identificarse en cada llamada mediante un token o algún otro identificador de usuario.

API es el acrónimo de Application Programming Interface (en español interfaz de programación de aplicaciones) y cuya principal característica es que permite que sus productos se comuniquen con otros sin que estos conozcan su implementación. Su funcionamiento es similar al de un contrato en el que el emisor envía una solicitud con una estructura conocida y el receptor podrá realizar la operación pertinente de forma adecuada y devolver una respuesta con también un formato preestablecido.

A continuación, en la Figura 1.8 vemos una imagen que ilustra qué es una API REST y como varios clientes diferentes pueden comunicarse con ella para obtener sus servicios.

En la actualidad, Android no permite llamadas a través de internet sin antes pedir y obtener los permisos de internet. Es por esta razón indispensable añadir ciertos permisos a nuestra aplicación y que estos sean otorgados por parte de usuario.

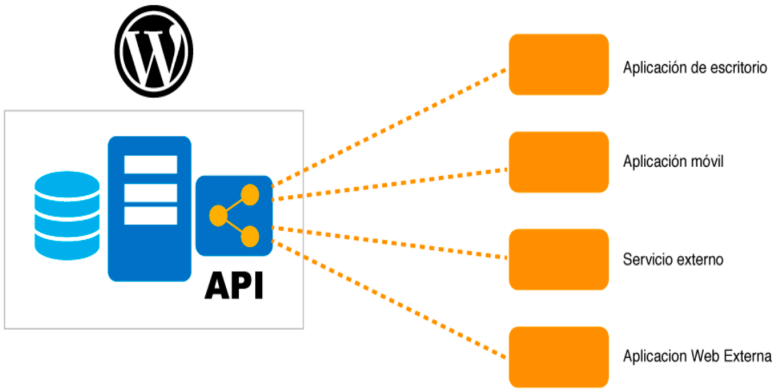


Figura 1.8: Esquema de una aplicación API REST. Fuente [12]

En cuanto a las API's, aunque esto se puede extrapolar a cualquier otra petición a través de internet, es necesario que se realicen en hilos secundarios de la aplicación, ya que ejecutarlos en el hilo principal, hilo que maneja la interfaz de usuario, resultaría en problemas de bloqueos de la interfaz.

1.3.5. Trabajo Fin de Grado de Andrés de la Maza Valles

Es el proyecto que finalmente implementó 14 nuevas llamadas API en el lado del servidor [6], que son las que será necesario migrar en el lado del cliente durante este proyecto para que la aplicación vuelva a ser funcional completamente. Las llamadas creadas son login, y registro de usuarios, así como modificación de sus datos, refrescar token, que crea un nuevo token para poder identificar al usuario en las llamadas, obtener los datos del usuario, obtener el ranking de los jugadores, listar las partidas pendientes, ver información de las partidas, obtener oponentes para una partida, crear partida, iniciar y finalizar la partida, añadir logros y ver lista de logros.

Cabe destacar que para la identificación de usuarios se utilizó JWT. Json Web Token (JWT) [13] es un estándar abierto que define una forma para transmitir información segura entre dos partes mediante un objeto JSON. Esta información es segura y confiable ya que está firmada digitalmente y esto se puede hacer de dos formas posibles que son la firma utilizando un secreto (esta es la firma que se utiliza en este proyecto) o utilizando un par de claves públicas o privadas. En este proyecto se utiliza como manera de mantener la sesión de un usuario en el sistema pasando el número de identificación de este entre las llamadas del cliente y el servidor.

1.3.6. Alternativas a Google Play Games

Como alternativa al cierre de Google Play Games, y solo en lo referente a la gestión de usuarios y datos, disponemos de Firebase. Firebase es un conjunto de herramientas que proporcionan una serie de servicios como la gestión de usuarios y gestión de datos como base de datos NoSQL, autenticación de usuarios y almacenamiento de ficheros, junto con una enorme cantidad de herramientas más. Los datos se almacenan en formato JSON y se sincronizan en tiempo real con cada cliente conecta-

do. Cuando se compila una aplicación multiplataforma todos sus clientes comparten una instancia de Firebase y reciben actualizaciones automáticamente con los datos más recientes [14]. Por esta razón, puede ser usado por Android, web o iOS.

En vez de utilizar Firebase, se optó por crear una API propia para poder configurar el servidor según las necesidades específicas de este aplicación. En palabras de Andrés de la Maza Valles en su TFG [6] ”configurarlo sin depender de programas externos y que además se adecue a los requisitos del proyecto. Esto aporta ventajas como una libre configuración de los requisitos y la gestión propia de los datos”. Además de esto, también aporta la posibilidad de cambios en el futuro de la funcionalidad, así como el evitar el pago por los servicios.

1.3.7. Duolingo

La aplicación de aprendizaje de idiomas más descargada de la Play Store, con más de 100 millones de descargas [15]. Utiliza la gamificación como elemento principal para favorecer el aprendizaje de una manera atractiva y divertida para los usuarios. Para ello permite avanzar en el juego al completar unidades restando vidas con las respuestas incorrectas, otorgando puntos con las correctas y se subiendo de nivel como en un juego. Además, es una aplicación gratuita que contiene la posibilidad de pagar por lecciones o eliminación de anuncios mediante el uso de una cuenta premium. En la Figura 1.9 podemos observar la interfaz de la aplicación.

A diferencia de Clash of Pronunciations, Duolingo se basa más en el aprendizaje de un idioma de manera general y no exclusivamente en la pronunciación de las palabras como sí hace CoP.

Figura 1.9: Vistas de la interfaz de Duolingo

1.3.8. Entorno y herramientas tecnológicas

En este apartado se exponen las tecnologías utilizadas para desarrollar este proyecto así como las principales características de cada una de ellas. De manera resumida para este TFG se ha utilizado un entorno para desarrollo Android bajo el lenguaje de programación Java (Android Studio) para la parte cliente. Además, se han hecho llamadas a la API RESTful (ver sección 1.3.4) desarrollada en el TFG de Andrés de la Maza Valles, implementada en Node y Mysql en los que se han debido hacer pequeñas modificaciones en el código cuando se encontraron *bugs* o se pretendía aumentar la funcionalidad. Para probar estos cambios se ha usado Postman. Por último, para el despliegue del servidor y el almacenamiento de datos se han utilizado máquinas virtuales a las que se ha accedido mediante Visual Studio Code y su extensión Remote-SSH para la visualización y acceso de ficheros.

Android 11

Según la propia Google ”Android es una pila de software de código abierto basado en Linux creada para una variedad amplia de dispositivos y factores de forma” [16]. Fue diseñado para dispositivos móviles con pantalla táctil. Proporciona su propio entorno sobre el que construir aplicaciones. En la Figura 1.10 podemos ver la arquitectura de Android. En ella podemos apreciar como en la base de la figura encontramos el kernel de Linux que otorga funcionalidades como la generación de subprocesos y la administración de memoria de bajo nivel. Por encima del kernel encontramos la capa de abstracción de hardware (HAL). Esta brinda interfaces estándares que exponen las capacidades de hardware del dispositivo al marco de trabajo de la API de Java de nivel más alto. Si seguimos subiendo en la figura encontramos el Android Runtime y las bibliotecas C/C++ nativas. Finalmente encontramos el marco de trabajo de la API de Java, que es todo el conjunto de funciones de Android que están disponibles y por encima de este encontramos las aplicaciones del sistema, que son aquellas que vienen instaladas de fábrica. En este mismo nivel se sitúan las aplicaciones que podemos descargar de la Play Store como CoP.

Inicialmente fue desarrollado por Android Inc., pero en 2005 fue adquirido por Google. El código fuente principal de Android se conoce como Android Open Source Project (AOSP). Android es el sistema operativo móvil más utilizado del mundo, con una cuota de mercado superior al 86 % el año 2020 [17].

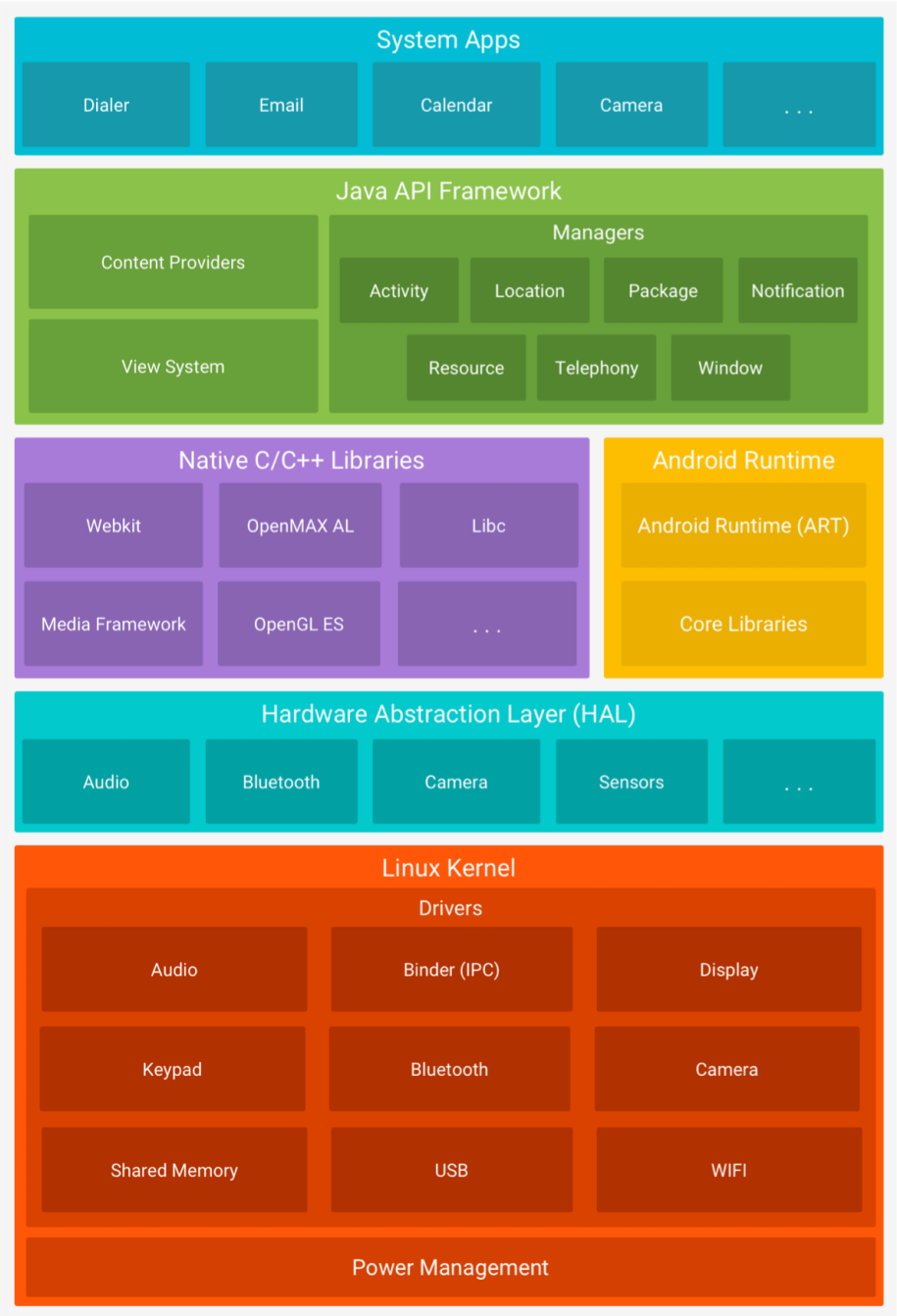


Figura 1.10: Arquitectura de Android. Fuente: Android developers [16]

Java 8

Java (ver Figura 1.11) es un lenguaje de programación cuya principal característica es que es orientado a objetos. Es el lenguaje usado para el desarrollo de las aplicaciones Android desde el comienzo del sistema operativo, aunque en la actualidad y desde hace varios años, Android ya no tiene preferencia por este lenguaje, si no por Kotlin [18], un lenguaje más moderno y sobre el que se construyen ya las principales aplicaciones de Android en todo el mundo [19]. A pesar de esto, Java sigue estando presente en gran cantidad de aplicaciones Android ya que en la mayor cantidad de casos no es viable la migración del código de un lenguaje a otro. Aunque se pueden usar ambos lenguajes de manera conjunta en un mismo proyecto [20], en CoP se decidió mantener Java para no complicar más los trabajos de la migración y porque además, Java es el lenguaje que más hemos usado a lo largo del Grado de Ingeniería Informática, por lo que evitamos la curva inicial de aprendizaje tanto para este proyecto como para futuras revisiones.

Figura 1.11: Logo de Java

Android Studio

Android Studio (ver Figura 1.12) [21] es el IDE oficial de Android que se creó exclusivamente a fin de acelerar el desarrollo y compilación de las aplicaciones Android. Por tanto, Android Studio es el entorno de desarrollo integrado (IDE) oficial para el desarrollo de aplicaciones para Android y está basado en IntelliJ IDEA [22]. Además de tener un potente editor de código y las herramientas para desarrolladores de IntelliJ, Android Studio ofrece otras funcionalidades añadidas como un sistema de compilación flexible basado en Gradle, un emulador de dispositivos Android, herramientas para identificar problemas de rendimiento, usabilidad y compatibilidad con versiones antiguas, entre otras muchas.

Figura 1.12: Logo de Android Studio

Google TTS y STT

Google Text-to-speech (TTS) [23] es la herramienta utilizada para la conversión de texto a voz en la aplicación. El servicio utiliza un lenguaje muy natural gracias a la inteligencia artificial de Google. Además, este servicio está disponible en más de 40 idiomas. En CoP esta herramienta es utilizada en las modalidades multijugador, discriminación y pronunciación para que el dispositivo lea las palabras en voz alta cuando pulsamos un botón de ayuda o al iniciar cada ronda de una partida.

Google Speech-to-Text (STT o ASR en inglés, **Automatic Speech recognition**)[24] es el caso contrario a TTS. Es una API de reconocimiento de voz que recibe un *json* con un formato dado y un audio. Mediante la inteligencia artificial de Google se interpreta este audio y se genera una respuesta que contiene una lista de palabras y la probabilidad de que cada una de ellas haya sido la palabra pronunciada. Además, en la versión de pago Google permite almacenar los audios y gracias a ello, podemos guardarlos en nuestro propio servidor de ECA-SIMM para su análisis posterior.

Node

Node.js (ver Figura 1.13) es un entorno de ejecución para JavaScript orientado a eventos asíncronos y está diseñado para crear aplicaciones de red escalables [25]. Está construido con el motor de JavaScript V8 de Chrome. En él casi ninguna función realiza I/O directamente, por lo que el proceso nunca se bloquea y por esa razón es muy propicio desarrollar sistemas escalables. HTTP es un elemento destacado en Node.js, diseñado teniendo en cuenta la transmisión de operaciones con streaming y baja latencia. Esto hace que Node.js sea muy adecuado para la base de una bilbioteca o un framework web.

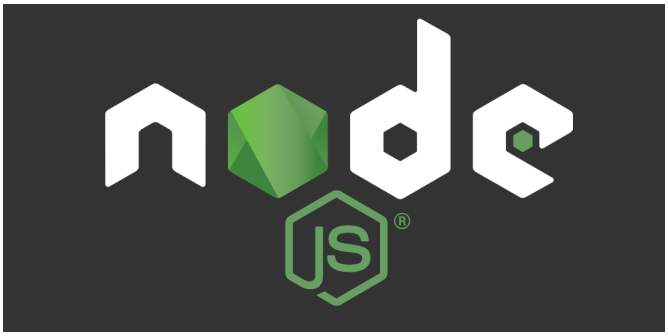


Figura 1.13: Logo de Node.js

MySQL

Como base de datos utilizamos MySQL (ver Figura 1.14) [26]. MySQL es un base de datos de tipo relacional. Una base de datos relacional almacena datos en tablas separadas. Las estructuras de la base de datos están organizadas en archivos físicos optimizados para aumentar su velocidad. El modelo lógico, con objetos como bases de datos, tablas, vistas, filas y columnas, ofrece un entorno de programación flexible. A diferencia de las bases de datos no relacionales, estas se centran en la velocidad y el rendimiento de las consultas, lo que favorece que el servidor responda más rápido las peticiones de las aplicaciones cliente.

Figura 1.14: Logo de MySQL

Postman

Postman [27] (ver Figura 1.15) es una herramienta muy útil para el desarrollo de API's, ya que permite ejecutar las llamadas a un servidor editando la petición HTTP al completo y recuperando los datos devuelve el servidor. Postman actúa como cliente creando diferentes llamadas API que se pueden almacenar para volver a usar y mostrando el resultado de estas consultas. Durante este proyecto se ha usado para realizar comprobaciones rápidas del servidor y almacenar consultas que han generado error para repetirlas hasta encontrar dicho problema y solucionarlo.

Figura 1.15: Logo de Postman

Visual Studio Code

Como herramienta para poder editar el código de la API hemos usado Visual Studio Code [28] (ver Figura 1.16). Además, gracias al uso del plugin Remote-SSH podemos crear una sesión SSH para editar los archivos del servidor alojados en la máquina virtual.

Figura 1.16: Logo de Visual Studio Code

1.4. Objetivos

El objetivo principal del proyecto es actualizar su código fuente para que vuelva a estar activo. Para ello se deberá migrar todas las antiguas llamadas a la API de Google Play Games a la nueva API REST creada en el Trabajo Fin de Grado de Andrés de la Maza Valles [6] volviendo a dar vida a la aplicación, que a día de hoy no es funcional. Además, habrá que actualizar el código fuente de Android para que sea compatible con la última versión hasta la fecha (junio de 2021). De manera más particular, los objetivos serán los siguientes:

- Migrar todas las llamadas a la nueva API REST.
- Actualizar la versión de Android desde la API 25 a la API 30.
- Corregir todos los errores o problemas que surjan durante la migración y la actualización de la versión.

1.5. Metodología

Dadas las características del proyecto, del hecho de ser un solo desarrollador y partiendo de la base de que los requisitos están fijados desde el inicio y no van a recibir apenas variaciones se ha optado por una metodología iterativa e incremental. Esta permite un cierto grado de modificación de los requisitos iniciales pero no se basa en la idea de que estos son totalmente cambiables como es en las metodologías ágiles. Las metodologías ágiles se realizan en entornos de grupos de trabajo con varios trabajadores y por esta razón no parece relevante para un solo trabajador.

Gracias a la metodología iterativa, en cada iteración se podrá presentar al cliente cierta funcionalidad implementada y no esperar hasta el final para presentar todo como es lo propio de las metodologías en cascada. Se puede comprobar, por tanto, que el proyecto sigue el camino indicado y realizar pequeñas correcciones de lo presentado en cada reunión con los clientes antes de continuar con las nuevas

funcionalidades. Además, el hecho de entregar el proyecto por partes permite eliminar el posible *gold-plating* (añadir funcionalidad que no ha sido requerida) y entregar solo lo planteado en los requisitos.

En la siguiente figura (ver Figura 1.17) podemos ver los pasos que requiere una metodología iterativa. Además, en el Capítulo 3 se definirán las diferentes iteraciones que se van a llevar a cabo en el proyecto.

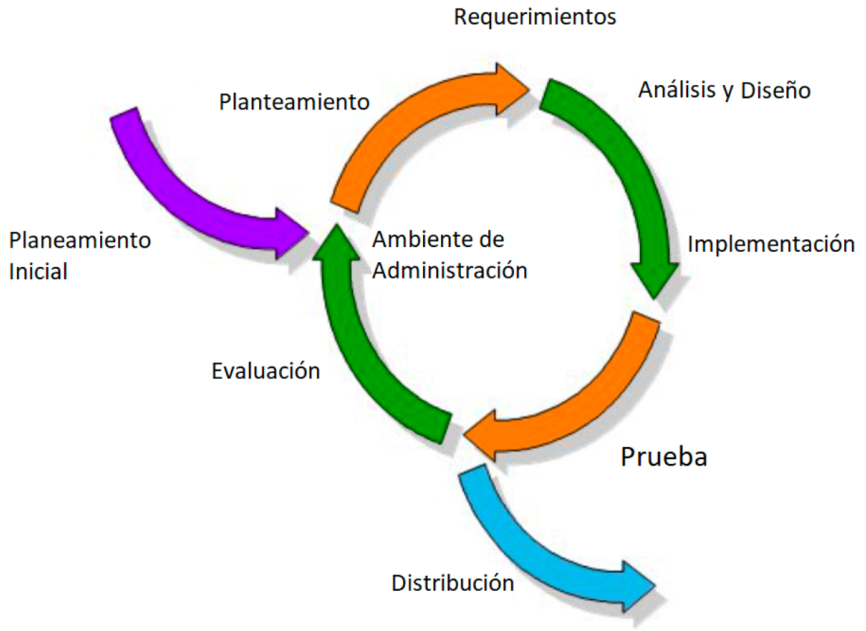


Figura 1.17: Ciclo de la metodología iterativa [29]

1.6. Estructura de la memoria

La presente memoria está dividida en los siguientes capítulos:

- **Introducción**: se presenta el proyecto de una forma general, tratando su contexto, motivación, estado actual de la aplicación, objetivos y metodología a seguir.
- **Planificación**: análisis, descripción y estimación del trabajo que se realizará, en cuántas iteraciones y bajo qué costes y riesgos.
- **Descripción de la iteraciones**: describe el trabajo real realizado en cada iteración y hasta finalizar el proyecto.
- **Estado de la cuestión**: describe todo el análisis y diseño de la aplicación, agrupando todos los diagramas y figuras realizados.
- **Pruebas**: agrupa el conjunto de pruebas realizadas en el proyecto.
- **Conclusiones**: conocimientos y conclusiones obtenidas del proyecto realizado así como una lista de trabajos futuros que podrían realizarse para mejorar las funcionalidades que ofrece la aplicación.

- **Apéndices**: distintos documentos adicionales como los acrónimos, un manual de despliegue completo sobre el despliegue del servidor y su base de datos, el despliegue CoP en el cliente, el manual de uso de CoP, el detalle de la funcionalidad implementada, los datos recabados en el test de usabilidad y el contenido entregado en el TFG.
- **Bibliografía**: conjunto de referencias utilizadas a la hora de desarrollar el proyecto.

Capítulo 2

Planificación

2.1. Planificación inicial

A lo largo de este capítulo se tratará de planificar y organizar el tiempo de trabajo necesario para completar las diferentes iteraciones del proyecto, alcanzando todos los requisitos al finalizar el proyecto.

2.1.1. Distribución temporal

El trabajo se va a llevar a cabo a lo largo de todo el segundo cuatrimestre del curso 2020-2021, empezando la primera iteración el día 25 de enero de 2021 y con fecha de finalización del 11 de junio de 2021 (dejando un margen de 17 días hasta el límite de la solicitud de la defensa del mismo). Empezar el Trabajo Fin de Grado (TFG) el día 25 de enero supone tener 138 días hasta el día final marcado como objetivo. El trabajo del desarrollador será diario de lunes a viernes con una carga aproximadamente de 3h, con una segunda iteración diferente con 5h diarias de trabajo, lo que supone un total de 330h de trabajo acumulado. Como la normativa actual de TFG establece 300h de trabajo a realizar para todo el proyecto completo, disponemos de 30h extra para cualquier retraso que pueda suceder. Además, contamos con 20 fines de semana (40 días en total) en lo que podría realizarse alguna hora extra si esta fuera necesaria para evitar posibles retrasos en la fecha de finalización del proyecto.

Es necesario destacar que se llevarán a cabo reuniones con los tutores aproximadamente cada 2 semanas, dependiendo de la necesidad en cada momento del proyecto, en las que se exponen los avances hechos en la última iteración, o desde la última reunión realizada, en caso de seguir en la misma iteración, se plantean los nuevos objetivos a realizar y se aclaran las dudas que hayan podido surgir.

El tiempo de trabajo estará dividido en iteraciones. A continuación se muestra la propuesta inicial:

- **1ª Iteración (2 semanas, 30h)**
  Se realizará una primera reunión con el cliente, en este caso, los tutores del proyecto. En dicha reunión se presentará el proyecto y se expondrá, de forma muy general, las labores que deberá

llevar a cabo el alumno a lo largo de todo el trabajo, se responderán las dudas iniciales del alumno y se definirán, de manera concreta, las primeras labores a realizar en esta primera iteración. A lo largo de esta iteración, y tras la revisión de toda la tecnología necesaria que se va a utilizar en el presente proyecto, se llevará a cabo el despliegue de la API en la máquina virtual prestada por la universidad según lo indicado en el Apéndice D. Además, se documentará la planificación del proyecto.

- **2ª Iteración (3 semanas, 75h)**
  A lo largo de esta iteración, el desarrollador no tendrá que realizar otras tareas como las prácticas de empresa (aún sin comenzar) o trabajar para otras asignaturas. Debido a esta situación del desarrollador, en esta iteración el número de horas de trabajo diarias ascenderá a 5h, por lo que el trabajo realizado cada semanas será mucho mayor que el trabajo del resto de semanas del proyecto, suponiendo un total de 75h al terminar la iteración. Durante esta iteración se creará una aplicación de prueba en la que se programarán todas las llamadas a la API ya desplegada. Para esto será necesario, en primer lugar, crear un mock-up para la aplicación y a continuación, crear la aplicación. Un mock-up no es más que una representación de una vista de manera simplificada que puede ser mostrada antes de su implementación. Para realizar estos mock-ups se ha utilizado Balsamiq [30]. También se dará acceso a través de phpMyAdmin a la base de datos para poder editarla de manera gráfica.
- **3ª Iteración (4 semanas, 60h)**
  Se llevará a cabo la descarga del código de la aplicación en su estado actual, se deberá realizar una comprobación inicial de la tecnología para encontrar posibles problemas de incompatibilidades, ya que esta ha estado sin actualizarse durante varios años. Se realizarán correcciones de posibles errores hasta que la aplicación se pueda compilar sin ellos y se pueda comenzar con la migración de las llamadas creadas en la aplicación de prueba a la aplicación real.
- **4ª Iteración (8 semanas, 120h)**
  Se efectuará la migración de todas las llamadas a la aplicación real, comprobando cada una de ellas. Al termino de la iteración, se deberá finalizar la completa migración de las llamadas desde la aplicación de prueba a la aplicación final.
- **5ª Iteración (3 semanas, 45h)**
  Se realizarán reuniones semanales con los tutores. Se deberán exponer todos los posibles inconvenientes o problemas encontrados a través de todo tipo de pruebas y ponerlos solución a lo largo de toda la iteración. Al finalizar la iteración, el proyecto debe estar finalizado en su totalidad.

En la Figura 2.1 se puede ver la distribución gráfica de las semanas para cada iteración. Se puede apreciar como la iteración que más semanas va a llevar de todo el proyecto es la 4ª iteración, ya que es la iteración en la que se migrarán todas las llamadas desde la aplicación de prueba a la aplicación real.

En lo que respecta al reparto de horas, y sabiendo que la segunda iteración va a suponer un trabajo de 5h al día, que son 2h diarias más que el resto de iteraciones, se puede ver en la Figura 2.2 como queda el reparto final.

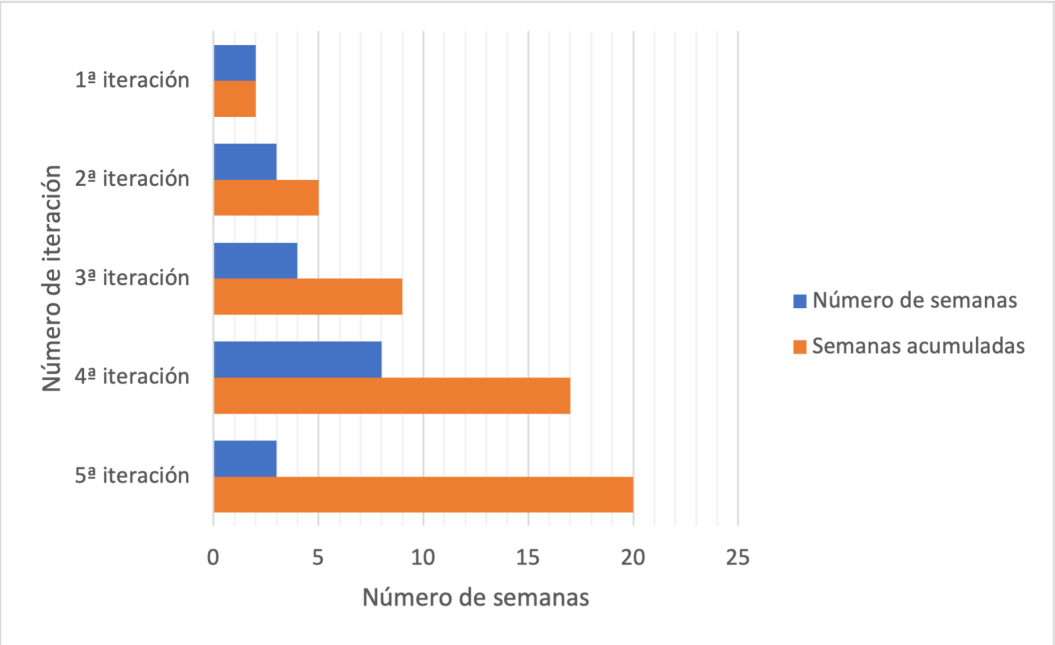


Figura 2.1: Distribución del número de semanas de desarrollo en cada iteración.

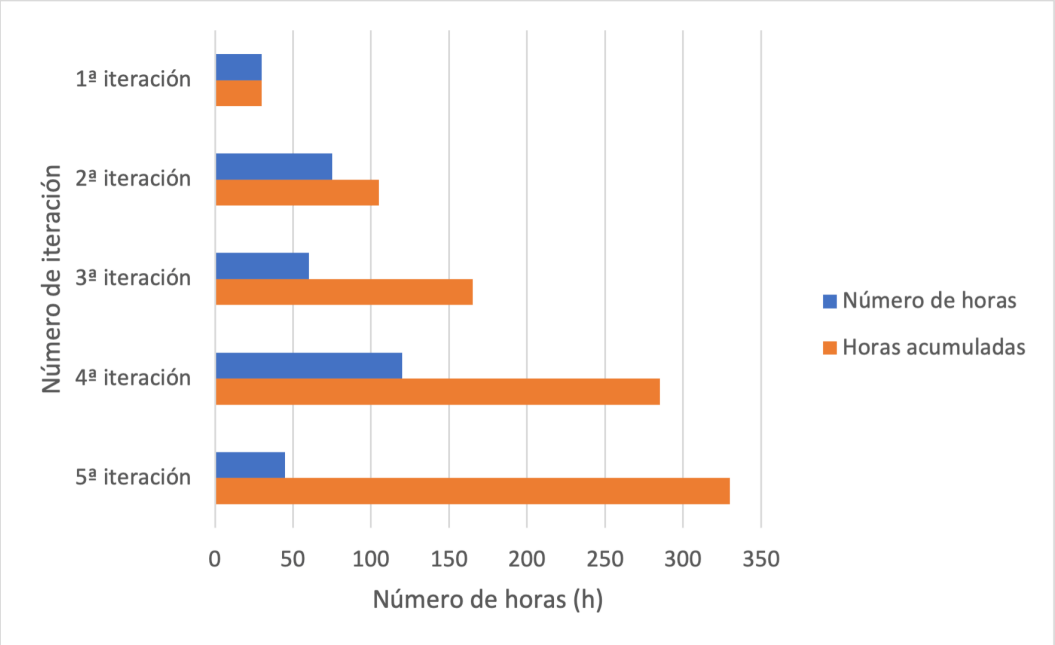


Figura 2.2: Distribución del número de horas de desarrollo en cada iteración

2.1.2. Análisis de riesgos

En la presente sección se van a detallar riesgos que se creen probables y que su aparición podría provocar retrasos en el desarrollo del proyecto. Todos estos riesgos aparecen junto a una pequeña definición, a la estimación de su probabilidad y al impacto que podría suponer sufrir cualquiera de ellos medido en días de retraso. Todo esto se puede ver en la Tabla 2.1. A tener en cuenta que cada día indicado en la última columna corresponden 4h/hombre de trabajo.

Como consecuencia de la existencia de los riesgos anteriores, también existe una serie de acciones para reducir la probabilidad del riesgo u otras que se llevan a cabo una vez que el riesgo se ha producido. En la tabla 2.2 se muestra en detalle las posibles medidas de reducción de la exposición al riesgo así como las medidas de contingencia en caso de que el riesgo se haya producido.

A continuación, en la Tabla 2.2, se detalla el plan de acción para cada uno de los posibles riesgos que ocurrieran durante el desarrollo del proyecto. Cabe destacar que la exposición al riesgo es el tiempo promedio que se perdería debido a cada uno de los riesgos; y por tanto se obtiene multiplicando las columnas de Probabilidad de que el riesgo suceda y el retraso estimado.

ID	Riesgo	Probabilidad (tanto por uno)	Descripción	Retraso estimado (días)
1	Suspender la asignatura Sistemas Empotrados en la convocatoria ordinaria	0.01	La asignatura se ha suspendido y es necesario dedicarla tiempo extra para la convocatoria extraordinaria.	21
2	Suspender la asignatura Sistemas Empotrados en la convocatoria extraordinaria	0.001	La asignatura queda suspensa este cuatrimestre y será necesario realizarla a lo largo de primer cuatrimestre del siguiente curso	120
3	Enfermedad	0.05	El desarrollador cae enfermo.	5
4	Contagio por COVID-19	0.05	El desarrollador es contagiado de COVID-19 y sufre una semana de fiebre y malestar.	7
5	Falta de experiencia en el desarrollo Android	0.5	Se emplea más tiempo del debido, sobre todo al inicio del desarrollo, en realizar las tareas asignadas.	10
6	Problemas en el despliegue de la aplicación de la API	0.4	Se produce un retraso en la primera iteración como consecuencia de los problemas asociados al despliegue de la API en el servidor	3
7	Planificación incorrecta	0.6	La falta de experiencia en la tarea de planificar junto con la tendencia a recortar los plazos más de lo debido provoca retrasos en la realización de tareas respecto a lo planeado	20
8	Pérdida de datos a lo largo del desarrollo	0.4	Durante la realización del proyecto se produce algún problema en los entornos de desarrollo o de documentación lo que provoca la pérdida de información	1
9	El ordenador de trabajo sufre algún problema	0.02	El ordenador con el que se desarrolla sufre un problema que impide seguir trabajando con él.	2

Tabla 2.1: Riesgos durante el desarrollo del proyecto. ID: Identificador del riesgo.

ID	Riesgo	Exposición (tanto por uno)	Plan de reducción del riesgo	Plan de mitigación del riesgo
1	Suspender la asignatura Sistemas Empotrados en la convocatoria ordinaria	0.21	Se llevará al día la asignatura para evitar suspenderla	Se pospondrá la entrega del proyecto para la convocatoria extraordinaria
2	Suspender la asignatura Sistemas Empotrados en la convocatoria extraordinaria	0.12	Se dedicará la mayor cantidad de horas posibles desde la convocatoria ordinaria a la extraordinaria para aprobar la asignatura	Se pospondrá la entrega del proyecto a mitad del primer cuatrimestre del siguiente curso
3	Enfermedad	0.25	-	Se usarán horas de los fines de semana para recuperar el retraso
4	Contagio por Covid-19	0.35	Cumplir las medidas de prevención anunciadas por los organismos públicos de sanidad	Se usarán horas de los fines de semana para recuperar el retraso
5	Falta de experiencia en el desarrollo Android	5	Haber cursado la asignatura de Sistemas Móviles	Usar la documentación oficial de Android ante cualquier duda y usar horas de los fines de semana en caso de retrasos
6	Problemas en el despliegue de la aplicación de la API	1.2	Seguir los manuales de despliegue del TFG anterior y leer documentación sobre aplicaciones Node.js junto con Nginx y PM2	Se usarán horas de los fines de semana para recuperar el retraso
7	Planificación incorrecta	12	Ser conservador a la hora de crear los plazos de entrega	Se usarán horas de los fines de semana y de los días posteriores a la fecha de finalización con el fin de mantener la entrega final del proyecto antes del final de la primera convocatoria
8	Pérdida de datos a lo largo del desarrollo	0.4	Usar autoguardado junto con el uso de la nube para almacenar copias del documento, así como uso de git y gitlab para mantener copias del código tanto en local como en remoto	Usar copias del último día de trabajo
9	El ordenador de trabajo sufre algún problema	0.04	Mantener el ordenador actualizado y guardarlo siempre en lugar adecuado.	Pedir un ordenador prestado y recuperar el trabajo del día anterior almacenado en la nube

Tabla 2.2: Plan de acción de los riesgos del proyecto. ID: Identificador del riesgo.

2.2. Entorno tecnológico

2.2.1. Herramientas utilizadas

A continuación, se describen las herramientas utilizadas a lo largo del desarrollo del proyecto junto con las funciones que han aportado cada una de ellas.

- **Android SDK API 30 [31]**: Suite proporcionada por Google para el desarrollo de aplicaciones para la plataforma móvil Android.
- **Android Studio 4.1.1 [32]**: Entorno de desarrollo Android para lenguaje Java o Kotlin.
- **Astah** [33]: Creación de diagramas UML.
- **Balsamiq Wireframes** [30]: Creación de mock-ups para las vistas de la aplicación de pruebas.
- **Excel 365 [34]**: Herramienta software con la que se han creado tablas y figuras para este proyecto.
- **Git 2.25.1 [35]**: Control de versiones del código del proyecto.
- **Gitlab (Versión Web) [36]**: Repositorio remoto para almacenar todo el código fuente y datos del proyecto.
- **Google Drive (Versión Web) [37]**: Almacenamiento de ficheros en la nube relacionados con la memoria.
- **Java 8 [38]**: Lenguaje utilizado para el desarrollo en Android.
- **MySQL 8.0.23-0ubuntu0.20.04.1 [26]**: Gestor de bases de datos relacionales y usado en el servidor para soportar nuestra base de datos.
- **Nginx 1.18.0 (Ubuntu) [39]**: Usado como proxy inverso que permite recoger las peticiones http procedentes del puerto 80 (en nuestra máquina virtual este puerto es el 65232) y mandárselas al servidor Node que tiene la API y que escucha en un puerto local 3001 (no accesible desde fuera).
- **Node 10.19.0 [40]**: Entorno de ejecución en tiempo real de JavaScript usado para construir la API.
- **npm 6.14.4 [41]**: Es el Node Package Manager y ayuda en la gestión de las dependencias y módulos de la API con Node así como en su ejecución.
- **Overleaf (versión web)[42]**: Sitio web usado para escribir documentos en LaTeX.
- **phpMyAdmin [43]**: Herramienta gratuita de software utilizada para poder acceder a la base de datos de manera gráfica a través del navegador.
- **PM2 4.5.1 [44]**: Usado para mantener el servidor corriendo continuamente. Si el servidor falla y se cierra PM2 volverá a iniciarle.

- **Postman 8.0.2 [27]**: Entorno para el desarrollo de APIs que permite realizar pruebas contra el servidor y almacenar todas las pruebas para volver a usarlas.
- **Safari 14.0.3 [45]**: Navegador usado para acceder a overleaf y escribir el documento así como buscar información.
- **Telegram 7.4 [46]**: Comunicación con los tutores del proyecto.
- **Visual Studio Code versión: 1.53.0 [47]**: Edición de ficheros de texto.

2.2.2. Entorno de desarrollo

Respecto al entorno utilizado, podemos ver en las siguientes Tablas 2.3, 2.4 y 2.5 los datos hardware y software asociados a cada dispositivo utilizado para poder llevar a cabo el proyecto.

Ordenador	
MacBook Pro 15 2019	
Hardware	
Procesador	Intel Core i7-9850H
Memoria	16 GB 2400 MHz DDR4
Disco duro	512 GB
Tarjeta gráfica	Radeon Pro 555X
Software	
Sistema operativo	macOS 11.2
Arquitectura del sistema	64 bits

Tabla 2.3: Ordenador utilizado para llevar a cabo el desarrollo

Móvil	
OnePlus 8	
Hardware	
Procesador	Qualcomm Snapdragon 865
Memoria	8GB
Disco duro	128 GB
Tarjeta gráfica	Adreno 650
Software	
Sistema operativo	Android 11
Arquitectura del sistema	64 bits

Tabla 2.4: Móvil utilizado en el desarrollo

Máquina virtual	
Hardware	
Procesador	4 cores
Memoria	4GB
Disco duro	20 GB
Software	
Sistema operativo	Ubuntu 20.04.1 LTS
Arquitectura del sistema	64 bits

Tabla 2.5: Máquina virtual usada para desplegar el servidor con la API durante el desarrollo

2.3. Estimación de costes

En el presente apartado se detalla la estimación inicial de costes del proyecto, tanto humanos como técnicos o de servicios. Todos los importes del hardware usado se contabilizan con IVA, así como el salario del desarrollador es bruto.

Salario del trabajador

En el caso de este proyecto, el trabajador, Juan García Diéguez, no ha estado contratado por la Universidad de Valladolid por lo que el salario se considerará 0, pero para poder realizar una estimación realista de los costes que este proyecto supondrían, se va llevar a cabo un segundo presupuesto incluyendo el coste medio del salario de un desarrollador junior. Teniendo en cuenta un salario medio de 20.000€ brutos anuales [48], o lo que es lo mismo, un salario de 10,26€/h.

Espacio de trabajo

El proyecto se ha llevado a cabo en el lugar de residencia del desarrollador. A pesar de esto, en la universidad están disponibles diferentes laboratorios o lugares de trabajo en los que podría realizarse, por lo que, al desconocer el coste real del espacio, se va a recurrir a estimarlo comparándolo con las opciones que hay en Valladolid sobre el co-working y el alquiler de un puesto similar.

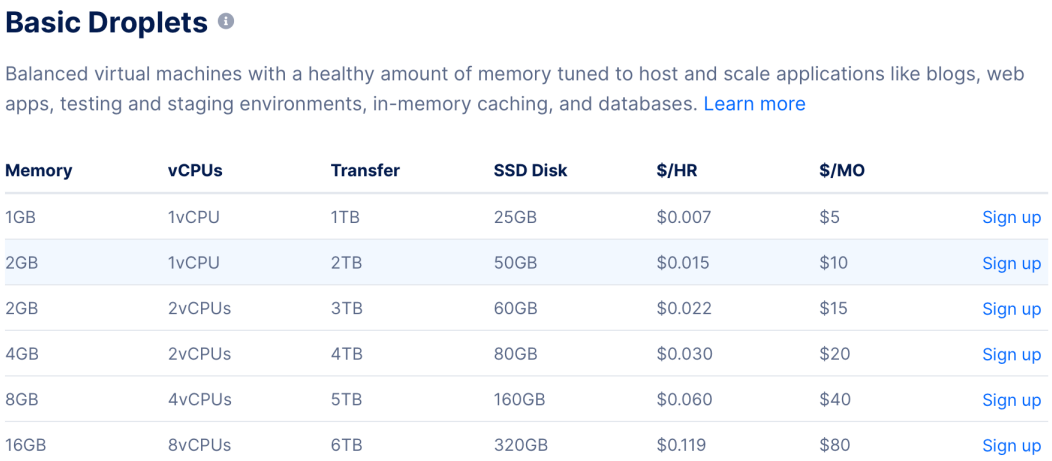
Material físico

El material utilizado para la elaboración del proyecto tiene asociado el coste cuando este fue comprado divido por el número de años que se estima tendrá su vida útil. Entre dichas herramientas encontramos obviamente el ordenador de trabajo junto con el móvil usado para probar la aplicación. El ordenador supone un coste para el proyecto de 162,5€ teniendo en cuenta que su precio fue de 2600€ y que se prevé un tiempo de vida de 8 años pero el trabajo se estima terminado en aproximadamente 5-6 meses. Respecto al móvil, este costó 530€ y prevé una vida de 4 años, por lo que es coste para este trabajo es de 66,25€ teniendo en cuenta de nuevo, que el trabajo completo durará aproximadamente 6 meses.

Servidor Cloud

La máquina virtual puesta a disposición del alumno para llevar a cabo el despliegue del servidor

del proyecto supone un coste que es muy difícil contabilizar, por lo que se optará por contabilizar el coste asociado al alquiler de un servidor cloud en el que soportarla. El proveedor elegido escogido es DigitalOcean y dentro de todos sus planes se ha escogido el plan Basic Droplets [49], que son máquinas virtuales con capacidad de mantener el servidor igual que lo hace la máquina virtual prestada por la universidad. En Figura 2.3 se pueden ver las diferentes configuraciones ofrecidas, siendo escogida la tercera, por ser la que tiene especificaciones más similares a la máquina virtual usada. El coste es de 15$ al mes, lo que suponen 90$ durante todo el proyecto, que transformados a euros con el cambio actual, 1€ = 1,21$ a 5-2-2021, son 74,18€.



Basic Droplets

Balanced virtual machines with a healthy amount of memory tuned to host and scale applications like blogs, web apps, testing and staging environments, in-memory caching, and databases. Learn more

Memory	vCPUs	Transfer	SSD Disk	$/HR	$/MO	
1GB	1vCPU	1TB	25GB	$0.007	$5	Sign up
2GB	1vCPU	2TB	50GB	$0.015	$10	Sign up
2GB	2vCPUs	3TB	60GB	$0.022	$15	Sign up
4GB	2vCPUs	4TB	80GB	$0.030	$20	Sign up
8GB	4vCPUs	5TB	160GB	$0.060	$40	Sign up
16GB	8vCPUs	6TB	320GB	$0.119	$80	Sign up

Figura 2.3: Diferentes configuraciones de máquinas virtuales ofrecidas por DigitalOcean. Fuente: DigitalOcean [49]

Servicios

Los servicios básicos utilizados para poder desarrollar el proyecto son luz e internet. Internet supone 50€/mes para la tarifa estándar de O2 [50], lo que serían 300€ durante toda la duración del proyecto. En cuanto a la luz, según tarifaluzhora.es [51], un uso de ordenador de 6h al día y 180h/mes supone 1,8kWh. Como nuestro uso será aproximadamente de 60h mensuales a lo largo de 6 meses estimamos un gasto de 3kWh, que con un precio de 0.1270€/kWh por parte de endesa [51] supone unos coste final para todo el proyecto de 0,381€.

Cómputo total

Como se puede apreciar de la tabla 2.6, el precio total de la realización del proyecto asciende a 1023,31€, lo cual es muy inferior en los visto en la tabla 2.7, que sería el coste real de realizar este proyecto bajo la contratación de un desarrollador junior.

Material	Precio Unitario	Unidades	Subtotal
Espacio de trabajo	200€/mes	3 meses	600€
Ordenador	162,5€	1	162,5€
OnePlus 8	66,25€	1	66,25€
Servidor cloud	12,36€/mes	6	74,18€
Internet	50€/mes	6	300€
Luz gastada por el ordenador	0.1270€/kWh	3	0,381€
Total	**-**	**-**	**1203,31€**

Tabla 2.6: Costes de la realización de proyecto, sin incluir el salario de un desarrollador junior.

Material	Precio Unitario	Unidades	Subtotal
Salario de desarrollador	10,26€/hora	330 horas	3.385,8€
Espacio de trabajo	200€/mes	3 meses	600€
Ordenador	162,5€	1	162,5€
OnePlus 8	66,25€	1	66,25€
Servidor cloud	12,36€/mes	6	74,18€
Internet	50€/mes	6	300€
Luz gastada por el ordenador	0.1270€/kWh	3	0,381€
Total	**-**	**-**	**4589,11€**

Tabla 2.7: Costes de la realización de proyecto, incluyendo el salario de un desarrollador junior.

2.4. Planificación final

Esta última sección de la planificación se realiza una vez terminado el proyecto al completo. En ella vamos a comparar los datos de la planificación inicial con los datos finales obtenidos, tanto en horas de trabajo, como en reuniones o seguimiento de las iteraciones.

Lo primero que debemos comentar es que la estimación de horas de trabajo quedó corta, ya que inicialmente se estimaron 330h y al finalizar el proyecto, las horas totales han sido 396h. Un parte importante de este exceso de horas viene dado por los problemas encontrados en la API desarrollada en el TFG de Andrés de la Maza [6]. Este riesgo no se tuvo en cuenta a la hora de estimar los riesgos y por tanto, tampoco tenía asociado un plan de mitigación. El hecho de que no se considerara como riesgo se debe a que el TFG de Andrés había sido probado. Por suerte, en la planificación inicial sí se tuvo en cuenta el riesgo de una mala planificación (ver en la Tabla 2.1 el riesgo ID-7), que tenía como plan de mitigación usar horas extra tanto de los fines de semana (ver Tabla 2.2 riesgo ID-7), como de los días posteriores al 11 de junio. Además de los retrasos provocados por este problema, ha habido otra serie de discrepancias entre lo establecido inicialmente y lo realizado al final. El cambio más importante es la unión de la iteración 3 y 4 (ver sección 2.1.1), ya que durante el desarrollo se llegó a la conclusión de que sería más sencillo realizar ambos trabajos a la vez. De esta manera la aplicación fue evolucionando a medida que se añadían llamadas y para cada nueva llamada, se corregían y actualizaban los problemas que surgieran en ese momento.

Finalmente, el número de iteraciones llevadas a cabo fue de 4. La primera y segunda iteración planificadas fueron iguales a las llevadas a cabo y en tiempo prácticamente iguales a los establecido. En la primera de ellas se realizó el despliegue de la API en una máquina virtual y en la segunda se creó la aplicación de pruebas. Como ya hemos dicho, la iteración 3 y 4 se fusionaron en una sola en la que se completó la migración y actualización de la aplicación al completo. Durante esta iteración también se corrigieron los problemas encontrados en el servidor. Aunque el tiempo trabajado ha sido de 190h y el estimado era de 180h (60h + 120h) con los problemas encontrados en el servidor no se pudo avanzar tanto como se deseaba el documento del proyecto. Esto fue lo que generó los problemas de tiempo ya comentados. Por último, la quinta iteración estimada inicialmente fue nombrada como la cuarta iteración. En su transcurso, cada semana se llevaron a cabo reuniones con los tutores y se presentaron diferentes avances. En la primera semana de ellas se avanzó el documento únicamente mientras los tutores corregían la aplicación. En la siguiente (semana 2 de la cuarta iteración) se corrigieron todos los problemas encontrados en la aplicación por parte de los tutores. En la tercera semana se acabó de escribir el documento y se corrigieron a la par los problemas que los tutores encontraron. Por último, durante la última semana se corrigieron los últimos errores o cambios requeridos por los tutores en una segunda revisión tanto en la aplicación como en el documento.

Durante todas estas iteraciones se han llevado a cabo un total de 13 reuniones con los tutores. Siempre se llevó a cabo una reunión al inicio de cada iteración, además, en la iteración 2 se efectuaron 2 reuniones con una separación de 3 semanas. La tercera iteración fue la que más reuniones tuvo, ya que en ellas se fueron presentando los diferentes avances y problemas encontrados en la aplicación. En esta tercera iteración el número de reuniones total fue de 6. Por último, a lo largo de la cuarta y última iteración las reuniones se sucedieron semanalmente dando un total de 4 reuniones más. A la par que las reuniones, se ha usado Telegram de manera más ostensible para plantear dudas o dar correcciones por parte de los tutores.

Estudiando el desarrollo del proyecto se puede deducir que las horas de trabajo totales superaron a las horas estimadas aunque no en todas la iteraciones, como ya se ha dicho. En las siguientes figuras (ver Figura 2.4 y Figura 2.5) podemos ver la distribución de total de horas en las iteraciones y la distribución de horas acumuladas. Como se ve las semanas de desarrollo tuvieron que alargarse en 2 y las horas superaron por más de un 20 % las horas estimadas inicialmente. En total las horas trabajas fueron 396h, esto es 66h más de las 330h planteadas inicialmente.

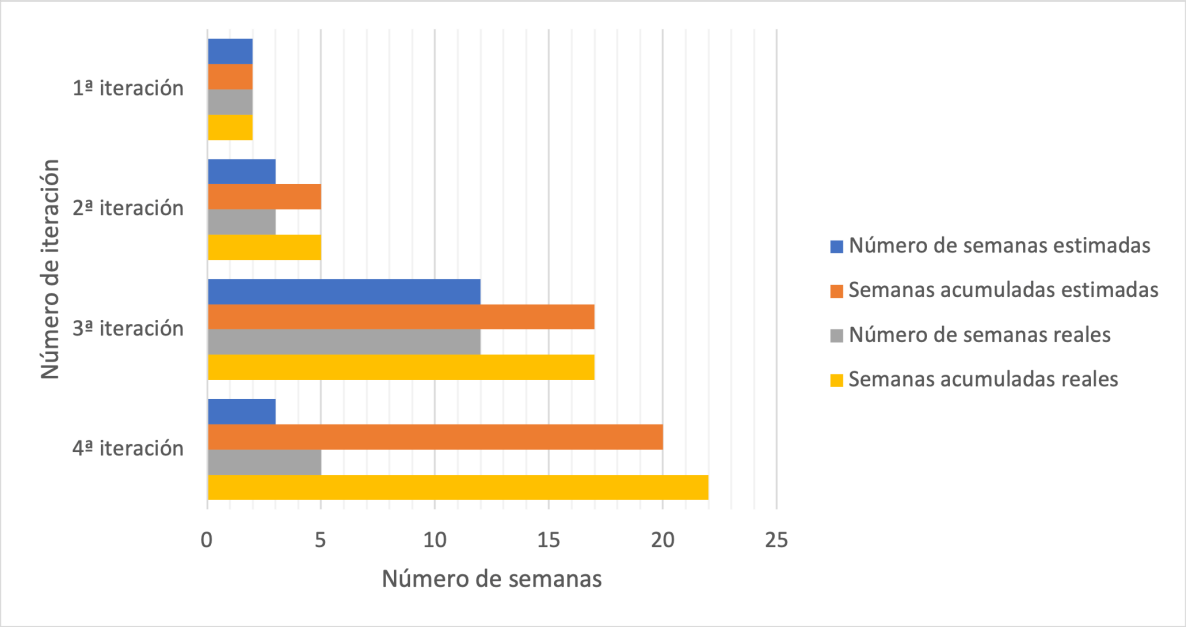


Figura 2.4: Comparación del número de semanas de desarrollo en cada iteración estimadas y reales

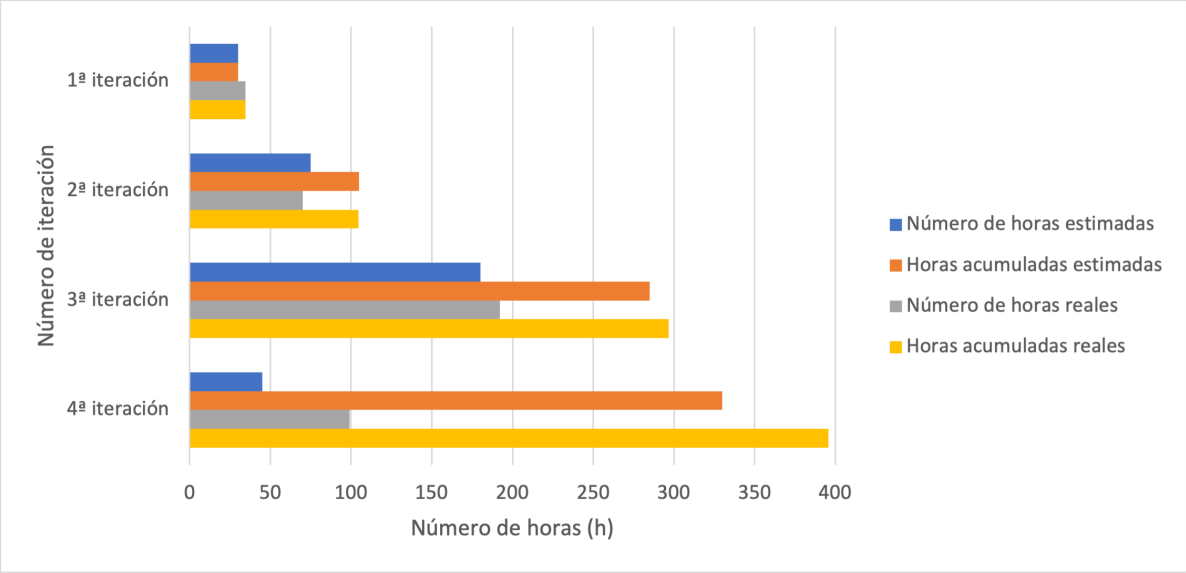


Figura 2.5: Comparación del número de horas de desarrollo en cada iteración estimadas y reales

Capítulo 3

Descripción de las iteraciones

En este capítulo se expone todo el trabajo realizado durante las diferentes iteraciones a lo largo del desarrollo del proyecto. Entre otras cosas, se van a comentar los requisitos del proyecto, las actualizaciones que ha sufrido la aplicación, el diseño de todas las nuevas pantallas necesarias en la aplicación y los diferentes problemas encontrados y solucionados a lo largo del desarrollo, principalmente en el servidor.

3.1. Iteración 1

Para poder comenzar el proyecto de manera adecuada, el alumno antes de empezar revisó la tecnología a usar, siendo esta principalmente Android 11 (API 30) programado con Java y llamadas a la API. Gracias a cursar las asignaturas Desarrollo Basado en Componentes y Servicios y Sistemas Móviles estos dos aspectos han quedado bien preparados. En el comienzo oficial del proyecto, el día 25 de enero de 2021, se llevó a cabo la reunión de inicio del trabajo. En ella se expusieron los requisitos que los clientes, en este caso los tutores, querían llevar a cabo, cuáles eran los primeros pasos que dar y cómo se iría desarrollando la tarea a lo largo tiempo.

A continuación, se expondrán los trabajos realizados durante la iteración, así como las tecnologías utilizadas.

3.1.1. Trabajo realizado

En primer lugar, se diseñó y escribió la planificación del presente proyecto (ver Sección 2.1.1 del Capítulo 2). Con toda la planificación desarrollada, se comenzó con el despliegue de la API creada en el TFG de Andrés de la Maza Valles [6].

Durante la mayor parte del tiempo de la iteración se estuvo llevando a cabo el despliegue del servidor en la máquina virtual cedida por la universidad. Para ello, fue necesaria el uso de una serie de herramientas software cuya instalación está descrita en el Apéndice D (Manual desarrollado durante

esta iteración). La API está gestionada por PM2 y Nginx. PM2 se encarga de mantener el proceso de la aplicación siempre activo y en caso de caída o fallo lo reinicia. Nginx en cambio, se usa para reenviar al servidor las llamadas http que reciba la máquina virtual, permitiendo la conexión de la aplicación con internet y no solo estar disponible con localhost. En cuanto a la base de datos usada ha sido MySQL, según lo indicado en el TFG de Andrés de la Maza Valles [6].

Tras la instalación de todas las herramientas y ya con el servidor en funcionamiento, se pasó a la creación de un script para automatizar los insert en la base de datos. Estos insert contienen los datos mínimos necesarios para poder arrancar la aplicación de manera adecuada y se dividieron en 2 archivos. El primero de ellos contiene datos de las 3 tablas de la base de datos que son obligatorias al inicio. Estas son las tablas de Languages, MultiLanguages y Achievements. En el segundo script se añadieron usuarios falsos para poder realizar las pruebas durante todo el desarrollo. Este script no es necesario cuando la aplicación pase a producción y se lance en la tienda de aplicaciones. Terminada la instalación del servidor se pasó a probar las llamadas desde Postman mediante la realización de una consulta para cada una de ellas y obteniendo resultados *json* en cada una.

3.2. Iteración 2

Al comienzo de esta iteración se llevó a cabo la segunda reunión con los tutores. En ella se confirmaron los objetivos marcados en la planificación y se marcó el inicio de la creación de la aplicación para probar las llamadas a la API. Esta aplicación permitirá comprobar cómo realizar llamadas a una API desde una aplicación de Android aislando los posibles problemas que contendrá CoP al actualizarla de versión por primera vez. Además de esto, esta aplicación permitirá realizar pruebas sin apenas interfaz y editando directamente la consulta (como si se ejecutara desde Postman) para poder probar diferentes tipos de consultas que generen error y bajo un entorno Android. En esta aplicación se podrán añadir futuras llamadas añadidas a la API y que se desean probar en una aplicación Android antes de ser añadidas a la aplicación real. Por último, esto permitirá alcanzar un cierto grado de experiencia con el framework de Android al desarrollador sin encontrar los problemas que CoP presentará en su primera compilación tras el cambio de versión. Para la creación de la aplicación de prueba de las llamadas ha sido necesario crear antes un pequeño mock-up de las 2 vistas que tendrá. Junto con la aplicación, se ha llevado a cabo la instalación de phpMyAdmin en la máquina virtual (ver en Apéndice D), con el fin de poder acceder a la base de datos de manera visual y desde cualquier navegador.

A lo largo de esta sección, además de lo ya comentado, se mostrará un diagrama de flujo usado para mostrar el orden de las llamadas a la API así como los problemas encontrados que han tenido que ser corregidos y las pruebas realizadas.

3.2.1. Requisitos de la aplicación de prueba de las llamadas

En primer lugar, fue necesario la creación de una tabla de requisitos funcionales de la aplicación de pruebas que permita validar si esta se ha creado cumpliendo todos los objetivos descritos. Los requisitos funcionales describen los servicios que la aplicación debe implementar para completarse. Aunque no se va a llevar a cabo un análisis más en profundidad de esta aplicación los requisitos son totalmente necesarios. A continuación, se muestra la Tabla 3.1 en la que podemos ver todos los requisitos funcionales de la aplicación de pruebas.

ID	Nombre	Descripción	Prioridad
RF01	Crear llamada login	El sistema debe realizar una llamada login con los datos de la vista y mostrar la respuesta del servidor	Extrema
RF02	Crear llamada refreshToken	El sistema debe realizar una llamada refreshToken con el refreshToken almacenado y mostrar la respuesta del servidor	Extrema
RF03	Crear llamada userData	El sistema debe realizar una llamada userData con el token almacenado y mostrar la respuesta del servidor	Extrema
RF04	Crear llamada ranking	El sistema debe realizar una llamada ranking con el token almacenado y mostrar la respuesta del servidor	Extrema
RF05	Crear llamada register	El sistema debe realizar una llamada register con los datos de la vista y mostrar la respuesta del servidor	Alta
RF06	Crear llamada listChallenges	El sistema debe realizar una llamada listChallenges con el token almacenado y mostrar la respuesta del servidor	Alta
RF07	Crear llamada opponents	El sistema debe realizar una llamada opponents con el token almacenado y mostrar la respuesta del servidor	Alta
RF08	Crear llamada createChallenge	El sistema debe realizar una llamada createChallenge con el token almacenado y las palabras y oponentes almacenados y mostrar la respuesta del servidor	Alta
RF09	Crear llamada startMatch	El sistema debe realizar una llamada startMatch con el token almacenado y mostrar la respuesta del servidor	Alta
RF10	Crear llamada finishMatch	El sistema debe realizar una llamada finishMatch con el token almacenado y mostrar la respuesta del servidor	Alta
RF11	Crear llamada infoChallenge	El sistema debe realizar una llamada infoChallenge con el token almacenado y el id de un challenge y mostrar la respuesta del servidor	Media
RF12	Crear llamada addAchievement	El sistema debe realizar una llamada addAchievement con el token almacenad y el id de un logro y mostrar la respuesta del servidor	Baja
RF13	Crear llamada showAchievements	El sistema debe realizar una llamada showAchievements con el token almacenado y mostrar la respuesta del servidor	Baja
RF14	Crear llamada modify	El sistema debe realizar una llamada modify con el token almacenado y los datos recuperados de la vista y mostrar la respuesta del servidor	Baja
RF15	Crear prueba de partida completa	El sistema debe ejecutar una serie de llamadas en cadena para completar una partida completa y mostrar el resultado de esta	Baja

Tabla 3.1: Requisitos funcionales de la aplicación de pruebas

No se detallan requisitos no funcionales o de información porque esta aplicación se crea como herramienta para comprobar las llamadas y poder realizar pruebas en caso de fallos desde un entorno Android con la última versión posible (Android 11).

3.2.2. Mock-up de la aplicación de prueba de las llamadas

En segundo lugar, se ha creado un mock-up con las 2 vistas que va a tener la aplicación de pruebas. La primera de ellas es una lista con las 14 llamadas que se van a reescribir en la aplicación CoP. Pinchando en cualquiera de ellas la aplicación realiza la llamada concreta al servidor con unos datos en formato json. A continuación, se muestra la segunda pantalla en la que se podrá ver si el resultado ha sido el esperado o en cambio, hemos recibido un error. Podemos ver los bocetos del mock-up en la Figura 3.1.

Figura 3.1: Mock-up de las vistas de la aplicación de prueba

3.2.3. Detalles de la aplicación de prueba de llamadas

Al crear la aplicación de prueba aparece el primer problema, que es escoger o no el uso de android.support. Las antiguas aplicaciones con la versión del SDK 28 o inferior, escogida para compilar, vienen por defecto con esta biblioteca. Como las versiones escogidas para nuestra aplicación son la versión 30 del SDK para compilación, la versión 26 como mínima versión del SDK y la versión 30 como target del SDK se ha pensado en usar la nueva biblioteca llamada androidx [52].

Android.support ha quedado obsoleta pero debido a que es la que se usó en la aplicación CoP cuando esta fue creada hace unos años habrá que migrar las clases, funciones y métodos usados en dicha aplicación a lo largo de la tercera y cuarta iteración. Para poder estar seguro de que el cambio es posible se ha realizado una serie de búsquedas en la página de desarrollo de android [52] y en alguna

página más [53]. Finalmente se ha decidido utilizar la nueva biblioteca androidx, ya que incorpora todas las clases, métodos y campos y solo cambia los nombres de estos. Además, desde el entorno de desarrollo Android Studio se puede realizar un refactor para migrar de la aplicación CoP a la nueva biblioteca androidx.

Una vez creada la estructura de la aplicación ha sido necesario crear los permisos para poder acceder a internet desde ella. Para ello se han seguido los pasos oficiales de la web de Android Developer [54].

Otros problemas surgidos del uso de la versión 30 del SDK, es que la clase AsyncTask, utilizada para realizar toda la parte de llamadas a la API en la aplicación CoP, ha quedado obsoleta [55], por lo que hay que buscar un nuevo método para poder realizar estas llamadas de manera paralela al hilo principal, que es el encargado de la interfaz. La solución al problema se encuentra en el uso de la clase Thread [56].

Se ha creado una clase que será la encargada de realizar las llamadas a la API. A la hora de actualizar las diferentes vistas de los fragments, y sabiendo que solo es posible que esto lo haga el hilo principal, ha sido necesario encontrar la manera de devolver la información de vuelta. Para ello callInNewThread, método encargado de crear un Thread y hacer la llamada, debe recibir las vistas a cambiar y ejecutar el método post de estas vistas. Post pone en la cola del hilo principal la ejecución del cambio en las vistas [57], que permitirá mostrar el mensaje de ”error” o de ”ok” en la vista del resultado 3.1 en función del contenido del *json* recibido.

En cuanto a la interfaz final de la aplicación, podemos verla en las siguientes Figuras (ver Figura 3.2 y Figura 3.3.

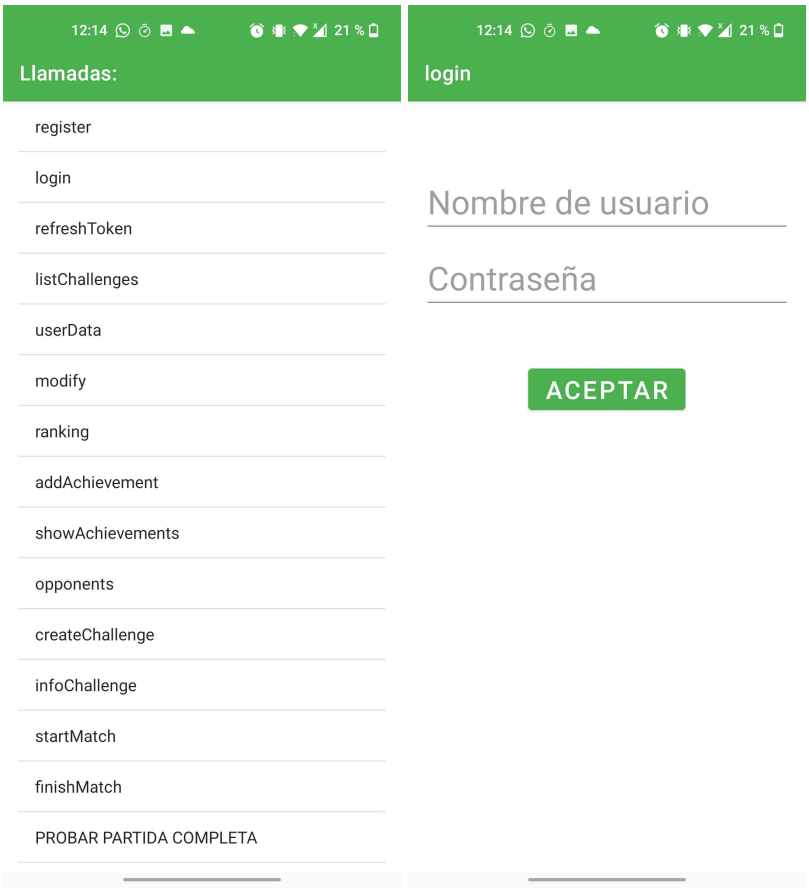


Figura 3.2: Vistas de todas las llamadas y del login de la aplicación de pruebas

Figura 3.3: Vistas de la pantalla de resultado de las llamadas de login (izquierda) y opponents(derecha)

3.2.4. Flujo de las llamadas

Para poder ejecutar las llamadas API en la aplicación de pruebas y posteriormente en CoP, y tener un resultado correcto es necesario seguir un orden adecuado. Se ha realizado un diagrama de flujo de las llamadas que será usado tanto para la aplicación de pruebas como para la aplicación final de CoP y que podemos ver en la Figura 3.4.

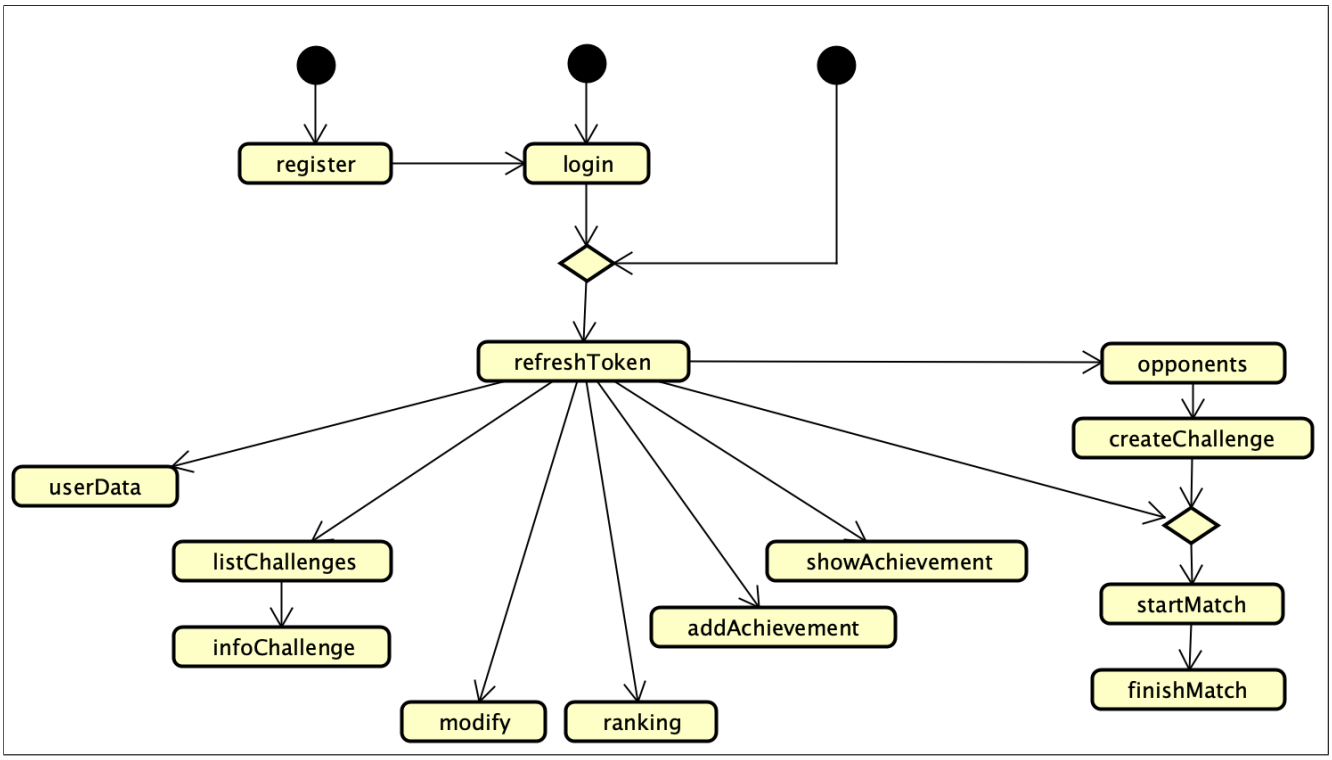


Figura 3.4: Flujo de las llamadas a la API

Como puede verse, lo primero que se debe hacer para realizar alguna prueba es o bien registrar un nuevo usuario, o realizar login con algún usuario ya creado o directamente usar refreshToken en caso de que ya se haya iniciado la sesión anteriormente. A continuación, se pueden lanzar varias llamadas distintas sin importar el orden, como son userData, modify, ranking, addAchievement, showAchievements y opponents. En el caso de querer crear una partida nueva, es necesario antes buscar oponentes con opponents, y luego crear la partida con createChallenge. Una vez creada se puede tanto ver la información con infoChallege, como jugar con startMatch y finalizar con finishMatch.

Además de poder ejecutar todas las llamadas de manera aislada, se ha creado también una opción para jugar una partida completa, de manera que se pueda ver el resultado final de la misma y comprobar si el resultado ha sido el esperado o no. Para ello la prueba pide un login inicial del usuario que creará la partida, y una vez iniciada la sesión y, por tanto, generado el token, la aplicación busca oponentes, crea un partida y juega y finaliza el turno del usuario creador. Lo siguiente que realiza es el bucle en el que recorre la lista de oponentes obtenidos y cada uno de ellos lleva a cabo su login, su juego y su finalización del turno. Al terminar el último jugador, se muestra el resultado obtenido, que debería ser ”Partida finalizada, actualización de la puntuación total” Volviendo a la lista de llamadas y pinchando el ranking se podrán ver como las puntuaciones de los jugadores de la partida han cambiado. En caso de

aparecer algún error en el proceso, este se muestra y la prueba termina.

3.2.5. Problemas encontrados

A parte de los problemas comentados en la sección anterior, el mayor problema encontrado surgió al intentar realizar todas las llamadas a la API y jugar para simular una partida completa con varios jugadores. No todas las llamadas se habían probado hasta ahora y ha sido en esta iteración donde se han encontrado ciertos errores.

Tras llevar a cabo una serie de pruebas comprobamos que la API no escribía las palabras en la base de datos. Como la llamada createChallenge no funcionaba correctamente, provocaba fallos en las siguientes llamadas. Fue necesaria la edición de dos queries que realizan los insert a la base de datos. Editando los campos de fechas la inserción de las palabras a su tabla comenzó a funcionar y la llamada pasó a estar correcta.

Para poder comprobar que todo estaba bien se intentó crear un partida completa de nuevo pero volvió a aparecer un error. En esta caso el error solo surgía cuando el último jugador trataba de acabar su turno (la llamada de la API finishMatch) y, por tanto, acabar la partida. Realizando una serie de comprobaciones para poder corregir el error se encontró un fallo en el else encargado de finalizar la partida (cuando el último jugador finaliza su turno). Se estaba intentado acceder a una lista con un valor indefinido. Fijando ese valor a 0 el error se corrigió y la partida terminó actualizando los valores de las puntuaciones en el ranking.

El último error encontrado en el servidor apareció al intentar probar todo en la máquina virtual, ya que en local ya había funcionado. Al crear las tablas en la base de datos de acuerdo al manual de instalación (Apéndice D), algunas tablas fallaban al crearse y la creación de todas ellas se detenía. La corrección de este error fue escribir todas las referencias a tablas de la base de datos de todos los archivos en formato ”CamelCase”. Esto quiere decir que los nombres de las tablas tienen la primera letra en mayúscula y si el nombre está compuesto de varias, cada una de ellas empezará por mayúscula también.

En cuanto a errores en la aplicación Android, surgió un problema a la hora de mandar los datos al servidor. El error se daba en el formato de codificación de los caracteres y apareció al intentar usar palabras con ñ. Estas palabras no se escribían en la base de datos, porque el formato de envío no era el correcto y, por tanto, a la hora de recuperarlas para usarlas en una partida, esta fallaba. La corrección consistió en obtener los bytes del mensaje antes de enviarlo con la codificación UTF-8.

3.2.6. Pruebas realizadas

Para poder comprobar que los errores surgidos han sido subsanados y que la aplicación funciona de manera correcta, lo cual permitirá pasar a la migración de las llamadas en la aplicación real, se han realizado una serie de pruebas que podemos ver en el Apéndice E. Algunas de estas pruebas ya habían sido probadas en el TFG de Clash of Pronunciations de Andrés de la Maza Valles [6], pero

es necesario rehacerlas, ya que se habían hecho desde Postman y en una máquina local. En nuestro caso, ya estamos probando en un escenario prácticamente real, ya que tenemos la API desplegada en un servidor, junto con una serie de software que la hace más robusta como es PM2, que en caso de caídas, reinicia el servidor, o Nginx que se usa como proxy inverso y, además, la aplicación que realiza las llamadas es una aplicación de Android, que simula ser la aplicación CoP real.

3.3. Iteración 3

La tercera iteración, al igual que las otras, comenzó con una reunión con los tutores para establecer el orden de los trabajos, en función de la planificación ya establecida. En ella se presentó la aplicación de pruebas ya finalizada y se le dio el visto bueno. Además, se presentó la aplicación CoP mediante una explicación de su código y de los paquetes que a priori, se debían cambiar durante la migración de las llamadas. Además, se expusieron los requisitos del proyecto.

3.3.1. Trabajos previos

En primer lugar, después de la reunión con los tutores se llevaron a cabo trabajos de corrección de la iteración previa, tanto a nivel de documento, como a nivel de programación.

En lo que hace referencia al proyecto, se editó el script de datos con el que se insertaban en la base de datos los datos de prueba. Dicho script se dividió en dos para aislar los datos estrictamente necesarios en una base de datos en producción, de los datos de usuarios falsos usados durante el periodo en desarrollo de la aplicación. Junto con esto, se editó el manual de despliegue para dejar patente estos cambios. Por último, se añadió un archivo README.md en el git de la aplicación para probar las llamadas, en el que se explica el propósito de la misma.

Una vez hecho esto, se pasó a la descarga del proyecto de CoP en la máquina local para comenzar con la labores de programación, siendo necesario un estudio y realización de un informe que permita ordenar y medir la importancia de los cambios.

3.3.2. Requisitos funcionales

Los requisitos funcionales de la aplicación se corresponden con los requisitos funcionales definidos para la aplicación de pruebas en la Sección 3.2.1 a excepción del RF15, que no aplica a la aplicación real y que será sustituido por un nuevo requisito funcional que podemos ver en la Tabla 3.2.

ID	Nombre	Descripción	Prioridad
RF01	Crear llamada login	El sistema debe realizar una llamada login con los datos de la vista y mostrar la respuesta del servidor	Extrema
RF02	Crear llamada refreshToken	El sistema debe realizar una llamada refreshToken con el refreshToken almacenado y mostrar la respuesta del servidor	Extrema
RF03	Crear llamada userData	El sistema debe realizar una llamada userData con el token almacenado y mostrar la respuesta del servidor	Extrema
RF04	Crear llamada ranking	El sistema debe realizar una llamada ranking con el token almacenado y mostrar la respuesta del servidor	Extrema
RF05	Crear llamada register	El sistema debe realizar una llamada register con los datos de la vista y mostrar la respuesta del servidor	Alta
RF06	Crear llamada listChallenges	El sistema debe realizar una llamada listChallenges con el token almacenado y mostrar la respuesta del servidor	Alta
RF07	Crear llamada opponents	El sistema debe realizar una llamada opponents con el token almacenado y mostrar la respuesta del servidor	Alta
RF08	Crear llamada createChallenge	El sistema debe realizar una llamada createChallenge con el token almacenado y las palabras y oponentes almacenados y mostrar la respuesta del servidor	Alta
RF09	Crear llamada startMatch	El sistema debe realizar una llamada startMatch con el token almacenado y mostrar la respuesta del servidor	Alta
RF10	Crear llamada finishMatch	El sistema debe realizar una llamada finishMatch con el token almacenado y mostrar la respuesta del servidor	Alta
RF11	Crear llamada infoChallenge	El sistema debe realizar una llamada infoChallenge con el token almacenado y el id de un challenge y mostrar la respuesta del servidor	Media
RF12	Crear llamada addAchievement	El sistema debe realizar una llamada addAchievement con el token almacenado y el id de un logro y mostrar la respuesta del servidor	Baja
RF13	Crear llamada showAchievements	El sistema debe realizar una llamada showAchievements con el token almacenado y mostrar la respuesta del servidor	Baja
RF14	Crear llamada modify	El sistema debe realizar una llamada modify con el token almacenado y los datos recuperados de la vista y mostrar la respuesta del servidor	Baja
RF15	Corrección de logs	El sistema debe enviar logs de las partidas al servidor de ECA-SIMM (https://eca-simm.uva.es/)	Alta

Tabla 3.2: Requisitos funcionales de CoP

3.3.3. Requisitos no funcionales

Los requisitos no funcionales de la aplicación hacen referencia a las características técnicas que el sistema debe cumplir. En la Tabla 3.3 se muestran los requisitos no funcionales del sistema ordenados de nuevo por prioridad.

ID	Nombre	Descripción	Prioridad
RNF01	Sistema Android	El sistema debe poder ejecutarse en un sistema Android	Extrema
RNF02	Versión mínima de sistema	El sistema debe migrarse a una versión mínima de la API 26 de Android	Extrema
RNF03	Versión de compilación	El sistema debe compilarse con la versión más actual de Android. Siendo esta Android 11 (API 30)	Extrema
RNF04	Corregir código obsoleto (deprecated)	El sistema debe ser migrado al completo a la versión de Android escogida en el punto anterior, corrigiendo cualquier problema asociado	Alta
RNF05	Corrección de permisos en la app	El sistema debe permitir almacenar en el dispositivo audios en formato amr y logs en formato *json*	Alta
RNF06	Eliminar código de la antigua API de Google Play Games	El sistema debe quedar limpio de código antiguo referente a Google Play Games	Alta
RNF07	Multi-idioma	El sistema debe ser mostrado al menos en 2 idiomas (español e inglés)	Alta
RNF08	Manual	La aplicación contara con un manual de usuario bien estructurado	Medio
RNF09	Codificación	La aplicación deberá utilizar la codificación de caracteres UTF-8	Alta
RNF10	Responsivo	La aplicación devolverá un resultado en menos de 1'5 segundos o en su defecto mostrará pantallas de carga.	Alta

Tabla 3.3: Requisitos no funcionales de la aplicación CoP

3.3.4. Requisitos de información

Los requisitos de información detallan la información que se debe guardar en el sistema. En la Tabla 3.4 se muestran los requisitos de información que deben ser añadidos como consecuencia de la migración y eliminación de la API de Google Play Games.

ID	Nombre	Descripción	Prioridad
RI01	Datos de usuario	El sistema debe almacenar el nick, email, contraseña, imagen de perfil, puntuación, logros y partidas de cada usuario	Extrema
RI02	Datos de partidas	El sistema debe almacenar las rondas, oponentes, puntuaciones de cada oponente, palabras y orden de cada ronda y el turno de cada partida	Extrema
RI03	Grabaciones	El sistema debe almacenar las grabaciones	Extrema
RI04	Logs de las partidas	El sistema debe almacenar ficheros *json* con el resultado de cada ronda de las partidas, de cada reconocimiento correcto o erróneo, de la elección de la palabra en rondas de Discriminación y de cada puntuación de cada usuario junto con el ganador final	Alta

Tabla 3.4: Requisitos de información de la aplicación CoP

3.3.5. Funcionalidad implementada

Debido a que la aplicación ha sufrido un salto de versión desde la versión 25 del SDK (correspondiente a Android 7.1) hasta la última versión disponible a día 3/2/2021 (Android 11 o versión 30 del SDK), lo primero que ha sido necesario realizar ha sido un refactor para migrar todo el proyecto de la biblioteca android.support a la nueva biblioteca androidX. A continuación, se ha llevado a cabo un análisis del estado de proyecto completo, para encontrar todos los problemas asociados, principalmente, a bibliotecas obsoletas (entre otra, la ya mencionada anteriormente AsyncTask). El resultado de este análisis se puede observar en la siguiente Figura 3.5. En esta tabla podemos ver todos los cambios que se deben realizar solo referidos a mantenimientos de la aplicación y actualización de la misma y la ejecución de las llamadas desde alguna de esas clases. Además, se deberá modificar toda la lógica que se vea involucrada aunque por el momento se desconoce cuáles serán esas clases y se irán comprobando a medida que se avance en el desarrollo.

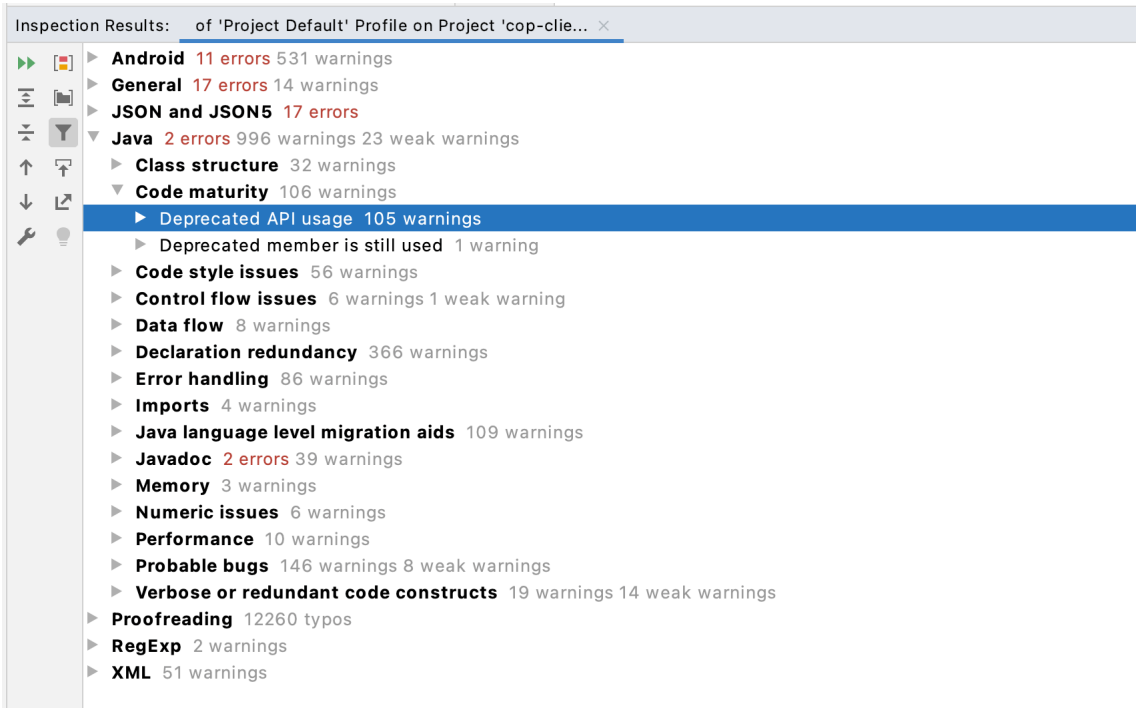


Figura 3.5: Resultado del análisis del proyecto mediante las herramientas de inspección de código de Android Studio

Centrándonos en los problemas más importantes, vemos que tenemos más de 100 errores asociados al uso de métodos, clases o bibliotecas obsoletas. Para poder diferenciar cuáles de los errores son más importantes y cuáles no tanto y organizar el trabajo a lo largo del tiempo, se ha realizado un informe. Este detalla el orden en el que deben ser corregidas las clases junto con otra serie de parámetros. Los parámetros usados para elaborar ese orden son el número de problemas encontrados, la llamada a la API a la que hacen referencia, si es que hacen referencia a alguna, y el nivel de importancia de dicha clase. El orden de ejecución de los trabajos de migración se puede ver en la Tabla 3.5.

Durante esta iteración se actualizaron todas las clases y ficheros relacionados como .xml, .json etc., de la Tabla 3.5. Además, se actualizó y migró el código de las clases que tienen relación con las llamadas API y se implementaron todas las llamadas necesarias. También se tuvieron que crear varias vistas desde cero ya que antes se usaban vistas que ofrecía la API de Google Play Games, como el login, registro, lista de partida, información de cada partida, modificación del perfil y logros. Junto con los cambios en CoP también fue necesario llevar a cabo cambios y correcciones en la API debido a que a lo largo de las iteraciones se fueron encontrando problemas y bugs en las diferentes llamadas.

Orden	Llamada API	Nombre de la clase	Warnings	Paquete	Importancia
1	-	RoundedImageView	1	ImageUtils	Extrema
2	-	SplashActivity	2	Activity	Extrema
3	Login	Login	1	Server	Extrema
4	RefreshToken	-	-	-	Extrema
5	UserData	UserDataRetriever	1	Server	Extrema
6	-	MenuActivity	18	Server	Extrema
7	Ranking	LeaderboardDataRetreiver	1	Server	Extrema
8	Ranking	LeaderboardHelper	4	ImageUtils	Extrema
9	Register	-	-	-	Alta
10	ListChallenges	UserDataRetriever	1	Server	Alta
11	Opponents	InviteesDataRetriever	1	Server	Alta
12	CreateChallenge StartMatch	CreateGameDataProvider	1	Server	Alta
13	-	MultiPlayerRoomActivity	5	Activity	Alta
14	-	MultiPlayerHandler	26	Game	Alta
15	-	Synthesis	1	Sound	Alta
16	-	PairsExposureActivity	10	Activity	Alta
17	-	SpeechAPIHandler	10	SpeechAPI	Alta
18	-	PostAudioTask	1	SpeechAPI	Alta
19	FinishMatch	EndedGameDataProvider	8	Server	Alta
20	FinishMatch	EndedGameDataRetriever	1	Server	Alta
21	InfoChallenge	-	-	-	Media
22	-	PermissionDialog	1	Permissions	Media
23	-	PermissionManager	1	Permissions	Media
24	-	SystemPermissionDialog	1	Permissions	Media
25	AddAchievement ShowAchieve-ments	AchievementHelper	6	Game	Baja
26	Modify	-	-	-	Baja
27	-	NotificationHandler	8	Notifications	Muy baja
28	-	NotificationsReceiver	1	Notifications	Muy baja
29	-	TasksDataProvider	1	Dashboard	Muy baja
30	-	TasksDataRetreiver	1	Dashboard	Muy baja
31	-	ListsDataRetriever	1	Dashboard	Muy baja
32	-	BaseGameUtils	3	Game	Muy baja

Tabla 3.5: Informe de las clases que deben ser editadas

Para poder ejecutar los trabajos de manera correcta durante toda la iteración, fue necesario la realización de varias reuniones a lo largo de esta. En ellas se presentaba el trabajo realizado y se exponía el trabajo futuro hasta la siguiente reunión. Además, antes de realizar cada reunión se realizaban pruebas de la funcionalidad implementada desde la propia aplicación, para verificar el correcto funcionamiento de la misma.

Para más información de los trabajos ejecutados se ha elaborado un apéndice en el que se detallan en profundidad todos los problemas encontrados durante la implementación de la funcionalidad (Apéndice B).

3.3.6. Pruebas realizadas

Este apartado es solo un resumen de las prueba ejecutadas durante el proyecto, para ver toda la información referente a las pruebas ir al apartado de Pruebas (capítulo 5) más adelante en este documento.

A medida que se iban añadiendo llamadas y corrigiendo clase obsoletas (deprecated) se fueron realizando pruebas tanto con la aplicación de pruebas creada al inicio como con la aplicación de CoP. La aplicación de pruebas permitía comprobar si las llamadas funcionaban bien desde un entorno Android y sin la necesidad de tratar con la interfaz de usuario de CoP, que hasta estar terminada podía generar algún tipo de problema. De esta manera se podían aislar errores en las llamadas y ejecutarlo de manera sencilla simplemente cambiando el contenido del *json* que se enviaba al servidor en cada llamada.

A su vez, a medida que avanzaba la implementación de las llamadas en CoP y su acople dentro del código antiguo, se iban probando las funcionalidades añadidas. Evidentemente los primeros casos probados correspondieron a las llamadas de login, userData y refreshToken y los últimos, ejecutar varias partidas completas y ver las puntuaciones y logros obtenidos en partidas multijugador entre el desarrollador del TFG y los tutores.

3.4. Iteración 4

3.4.1. Trabajos realizados

De nuevo la iteración comenzó con una reunión con los tutores en la que se expusieron las labores ejecutadas durante toda la iteración anterior, la presentación de la aplicación final y los siguientes pasos antes de concluir el trabajo. La aplicación se entregó a los tutores para que comprobaran las funcionalidades añadidas y crearan una lista de fallos o bugs a corregir durante esta iteración. Mientras los tutores llevaban a cabo esta corrección se pasó a avanzar la memoria del proyecto durante la primera semana de la iteración.

Al comenzar la segunda semana de la iteración se llevó a cabo una nueva reunión. Los tutores presentaron un informe con 40 cambios o fallos ordenados por prioridad y que debían ser solventados

a lo largo de la semana. Mientras se trabajaba de nuevo en la aplicación, los tutores corrigieron todo el documento que se había completado hasta ese momento generando un nuevo informe con cambios y correcciones que realizar.

Al inicio de la tercera semana se organizó una reunión para comprobar el avance del proyecto. Se entregó la nueva aplicación para una última revisión y se comentaron los cambios necesarios en el documento. A lo largo de esta semana la labores del alumno, por tanto, fueron acabar y corregir el documento mientras los tutores realizaban la última corrección de la aplicación.

Por último, en la última semana de trabajo se llevó a cabo una última reunión en la que se entregó el documento para realizarse una última corrección de 25 cambios o fallos encontrados. A lo largo de esta semana se corrigieron tanto dichos fallos en la aplicación como los del manuscrito y se realizaron pequeños ajustes en las vistas de la aplicación.

3.4.2. Productos finales

Al final del TFG se ha obtenido una aplicación (cop.apk) basada en Android 11 (API 30) lista para ser subida a la Play Store. Además de la aplicación, el código fuente de la aplicación de pruebas, de la app CoP y del servidor de CoP se encuentran en tres repositorios privados del grupo ECA-SIMM. En cuanto al servidor, podemos encontrarlo desplegado junto a la base de datos en una máquina virtual proporcionada por la Escuela de Ingeniería Informática de la Universidad de Valladolid. Por último, todo el TFG ha sido documentado en este manuscrito.

Capítulo 4

Estado final de la aplicación

A lo largo de este capítulo trataremos los temas relacionados con el análisis y diseño de la aplicación, exponiendo tanto diagramas como patrones o técnicas utilizadas.

4.1. Diagramas de análisis

Durante esta sección se trata el flujo de las llamadas en la aplicación, las historias de usuario, el modelo de dominio y los diagramas de actividad de las historias de usuario obtenidas.

4.1.1. Historias de usuario

A lo largo de esta sección se muestran todas las historias de usuario, de manera detallada, que tiene el proyecto. Todas las historias de usuario tienen como prerrequisitos tener una sesión iniciada en la aplicación, tener acceso a internet y haber aceptado los permisos que se piden al inicio a excepción del login y register, que no requieren haber iniciado sesión ni de haber aceptado permisos.

ID	HU01
Nombre	Login de un usuario
Prioridad	Alta
Riesgo	Bajo
Descripción	Como usuario quiero poder acceder a la aplicación a través de una pantalla de login
Validación	- Quiero poder introducir mi nick y contraseña para acceder a mi perfil de usuario - Quiero que mi nick sea único

Tabla 4.1: Historia de usuario HU01

ID	HU02
Nombre	Datos de usuario
Prioridad	Alta
Riesgo	Bajo
Descripción	Como usuario quiero poder ver mis datos del perfil, junto con mi puntuación y ranking
Validación	- Quiero poder ver mis datos en las diferentes vistas junto con mi puntuación o posición en el ranking - Quiero que mis datos no se almacenen en el dispositivo y solo se usen para mostrar la información

Tabla 4.2: Historia de usuario HU02

ID	HU03
Nombre	Registro de un nuevo usuario
Prioridad	Alta
Riesgo	Bajo
Descripción	Como usuario quiero poder registrarme en el sistema
Validación	- Quiero poder crear un nuevo usuario con mi nick, email, foto de perfil y contraseña - Quiero que mi nick sea único - Quiero que mi email sea único - Quiero que mi contraseña se pida 2 veces para evitar equivocaciones al escribirla - Quiero que mi imagen de perfil provenga de una lisa de imágenes por defecto para evitar poner una imagen propia - Quiero que todos los datos se almacenen de forma segura y no se utilicen con fines inadecuados

Tabla 4.3: Historia de usuario HU03

ID	HU04
Nombre	Ranking de usuarios
Prioridad	Alta
Riesgo	Medio
Descripción	Como usuario quiero poder ver el ranking de usuarios y mi posición en él
Validación	- Quiero poder acceder al ranking de usuarios - Quiero el ranking esté ordenado por puntuación - Quiero que el ranking muestre los nicks de los usuarios junto con su imagen de perfil - Quiero que mi posición aparezca de otro color para identificarme bien y que al acceder al ranking se haga scroll hasta ella

Tabla 4.4: Historia de usuario HU04

ID	HU05
Nombre	Lista de partidas
Prioridad	Alta
Riesgo	Alto
Descripción	Como usuario quiero poder ver la lista de partidas
Validación	- Quiero poder ver la lista de partidas en las que soy participante - Quiero que se diferencien las partidas acabadas, las que ya he jugado pero no están terminadas y las que no he jugado aún - Quiero que se muestre información de la partida para poder identificarla

Tabla 4.5: Historia de usuario HU05

ID	HU06
Nombre	Crear partida
Prioridad	Alta
Riesgo	Alto
Descripción	Como usuario quiero poder crear una nueva partida multijugador
Validación	- Quiero poder ver una lista de oponentes de un nivel similar al mío y a los que pueda retar - Quiero poder escoger varios de esos oponentes y crear la partida con ellos - Quiero poder jugar una partida completamente y al finalizar se me otorgue la puntuación correspondiente a las respuestas correctas - Quiero que si me salgo de una partida se me den 0 puntos como penalización - Quiero que al completar la partida se almacene la información y se me muestre la puntuación obtenida por mí y por el resto de participantes - Quiero poder obtener logros por la partida jugada

Tabla 4.6: Historia de usuario HU06

ID	HU07
Nombre	Ver información de partida
Prioridad	Media
Riesgo	Alto
Descripción	Como usuario quiero poder ver la información de una partida de manera detallada y, en caso de no haberla jugado, poder iniciar la partida
Validación	- Quiero poder acceder a más información sobre una partida concreta - Quiero poder iniciar la partida si no la he jugado aún - Quiero poder obtener logros por la partida jugada

Tabla 4.7: Historia de usuario HU07

ID	HU08
Nombre	Ver logros
Prioridad	Baja
Riesgo	Bajo
Descripción	Como usuario quiero ver mis logros
Validación	- Quiero poder acceder a los logros que ya he obtenido - Quiero poder ver los logros que me quedan por obtener

Tabla 4.8: Historia de usuario HU08

ID	HU09
Nombre	Modificar el perfil de usuario
Prioridad	Baja
Riesgo	Bajo
Descripción	Como usuario quiero poder modificar mis datos de perfil
Validación	- Quiero poder modificar mi email - Quiero poder modificar mi foto de perfil - Quiero poder modificar mi contraseña

Tabla 4.9: Historia de usuario HU09

4.1.2. Modelo de dominio

El modelo de dominio de la aplicación TipTopTalk! ha evolucionado y cambiado en gran parte (ver Figura 4.1), ya que con los trabajos realizados desde la creación de CoP, este ha variado bastante. El original fue creado para TipTopTalk! [1] y en aquel momento no se pensaba en añadir funcionalidad multijugador. Más tarde se añadió esta funcionalidad pero este no fue actualizado correctamente [9]. Por estas razones se ha optado por rehacer de nuevo dando lugar la siguiente figura. Las clases en amarillo corresponden con clases del modelo de dominio original y en azul son clases nuevas. Podemos ver como un usuario (Participant) puede lograr trofeos (Trophy) y se encuentra en un ranking (Ranking). Este usuario, además, juega partidas (Challenge), que pueden ser tanto de entrenamiento como multijugador. Cada partida estará compuesta de tareas (Task) (9 rondas actualmente) y estas contienen palabras (WordData) (2 actualmente). Podemos observar como cada tarea puede ser de un tipo diferente ya sea de exposición (Exposure), pronunciación (Pronunciation) o de discriminación (Discrimination). Por último, discriminación, pronunciación y exposición harán uso de TTS y pronunciación y exposición usarán ASR. En las siguientes tablas se describirán más en detalle las nuevas clases añadidas (ver Tabla 4.10, Tabla 4.11, Tabla 4.12 y Tabla 4.13).

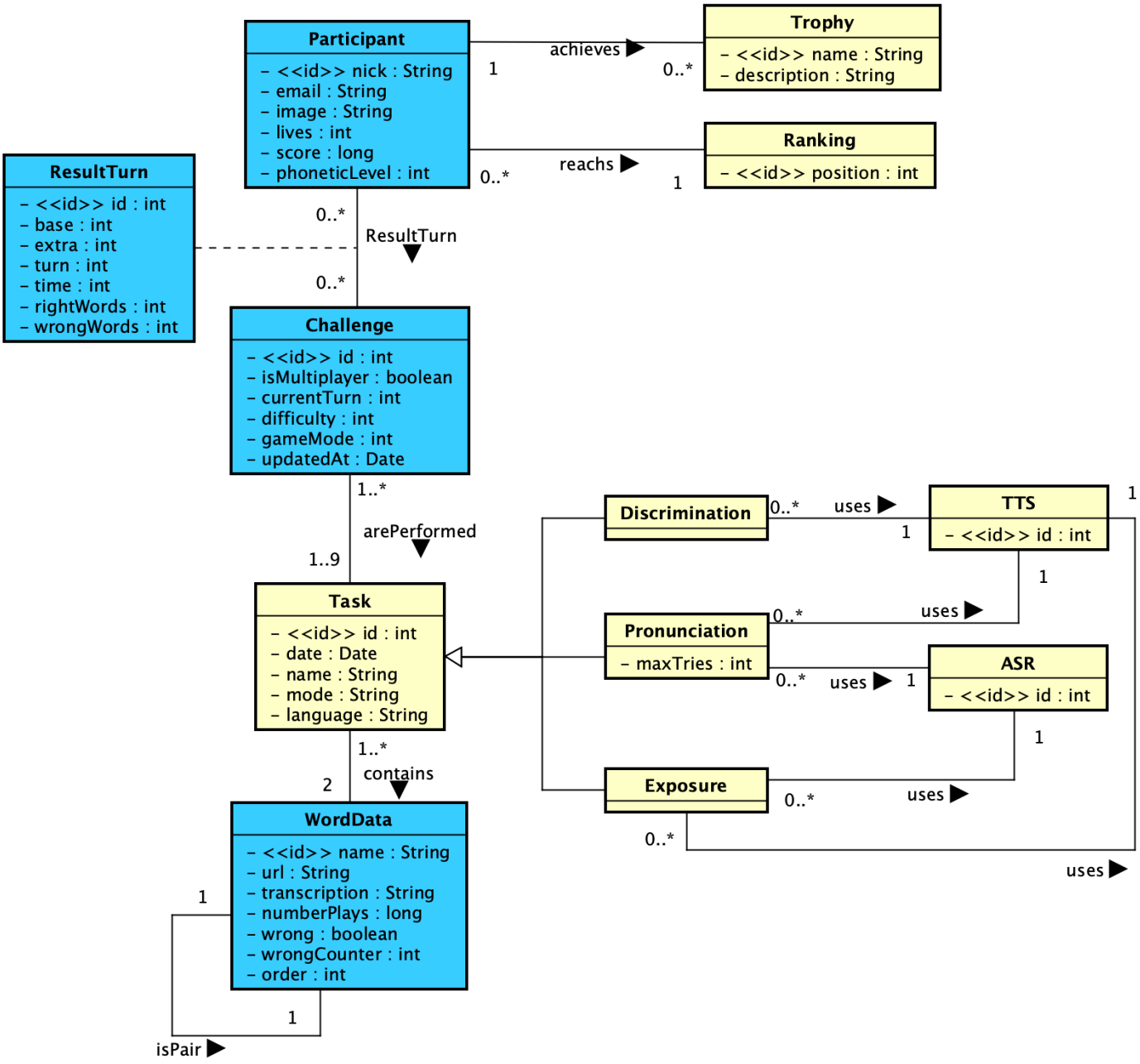


Figura 4.1: Modelo de dominio de CoP

Nombre	Participant
Descripción	Modela un participante de un partida. Contiene todos los datos necesarios que se reciben o envían en las llamadas de opponents, createChallenge o infoChallenge
Atributos	■ nick: contiene el nombre de usuario, que permite iniciar sesión y que es único para cada usuario ■ email: contiene el email del usuario y que es único entre todos los usuarios ■ image: contiene la url de la imagen de perfil de un usuario ■ score: contiene la suma de las puntaciones extra y base ■ phoneticLevel: contiene el valor de nivel fonético del usuario ■ ranking: contiene la posición en el ranking del usuario

Tabla 4.10: Entidad Participant

Nombre	Challenge
Descripción	Modela una partida. Contiene todos los datos necesarios que se reciben o envían en las llamadas de listChallenges, createChallenge, startMatch, finishMatch o infoChallenge
Atributos	■ id: contiene el identificador único de un Challenge ■ difficulty: contiene la dificultad de una partida. Actualmente todas las partidas son dificultad media ■ isMultiplayer: boolean que a true indica que la partida es multijugador. Si es false será un entrenamiento ■ currentTurn: contiene el turno del usuario ■ gameMode: contiene el tipo de juego ■ updatedAt: contiene la última fecha de actualización del Challenge

Tabla 4.11: Entidad Challenge

Nombre	WordData
Descripción	Modela una palabra. Contiene todos los datos necesarios que se reciben o envían en las llamadas de createChallenge o infoChallenge
Atributos	■ name: contiene el nombre de la palabra en CamelCase. Este nombre es su identificador ■ counterWord: contiene un byte que identifica el contador de la palabra. Al principio iniciado a COUNTER_WORD_NOT_SET para que no se muestren mensajes hasta que no se haya jugado al menos una vez con ella ■ url: contiene la url de la palabra y permite mostrar su imagen. Actualmente no se usa, pero se mantiene para posibles añadidos futuros ■ transcription: contiene la transcripción de la palabra para mostrar como debe ser pronunciada ■ numberPlays: contiene el número de intento que puede tener una palabra ■ wrong: boolean que indica si la palabra se ha dado por incorrecta definitivamente ■ wrongCounter: contiene el número de fallos en un turno que ha tenido cada palabra ■ order: contiene el orden que las palabras han tenido en la partida del creador y que permite mantenerlo en las partidas de los oponentes

Tabla 4.12: Entidad WordData

Nombre	ResultTurn
Descripción	Modela el resultado de un Parcitipant en un Challenge concreto. Además, permite identificar cada relación Participant Challenge ya que un Participant puede tener muchos Challenges y un Challenge puede tener muchos Participants
Atributos	■ id: contiene su identificador ■ base: contiene la puntuación base del usuario o -1 si aún no tiene puntuación ■ extra: contiene la puntuación extra del usuario o -1 si aún no tiene puntuación ■ turn: contiene el turno que se le ha asignado al usuario ■ time: contiene el tiempo empleado por el usuario en jugar un Challenge ■ rightWords: contiene el número de palabras correctas de un Participant en un Challenge ■ wrongWords: contiene el número de palabras incorrectas de un Participant en un Challenge

Tabla 4.13: ResultTurn

4.1.3. Diagramas de actividad

En la presente sección se exponen todos los diagramas de actividad de las historias de usuario definidas anteriormente. Se detallan tanto las interacciones del usuario real con la aplicación como las acciones que realiza el frontend de la aplicación y las llamadas ejecutadas a la API del servidor.

Login de un usuario (HU01)

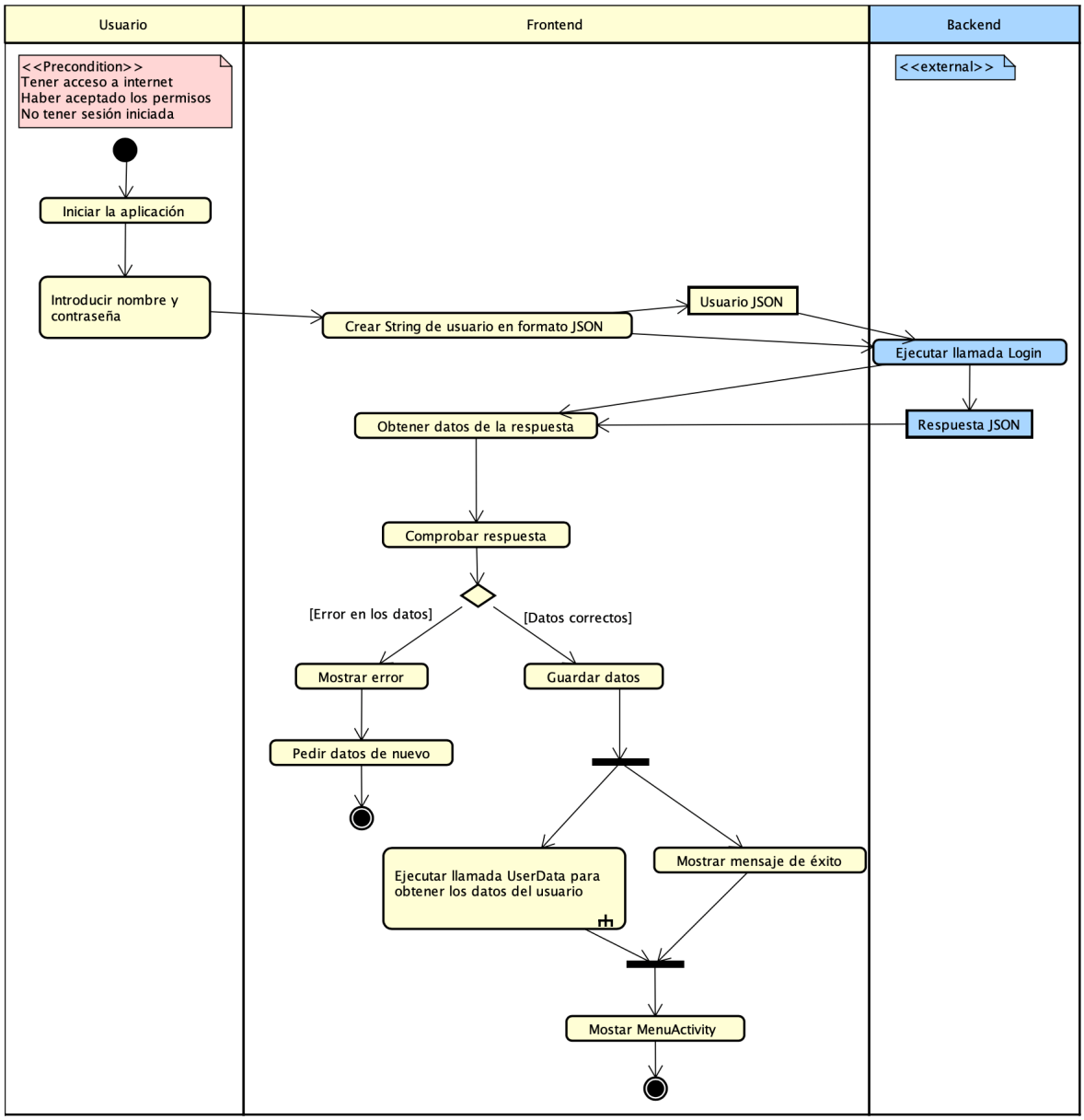


Figura 4.2: Diagrama de la historia de usuario del login (HU01)

Datos de usuario (HU02)

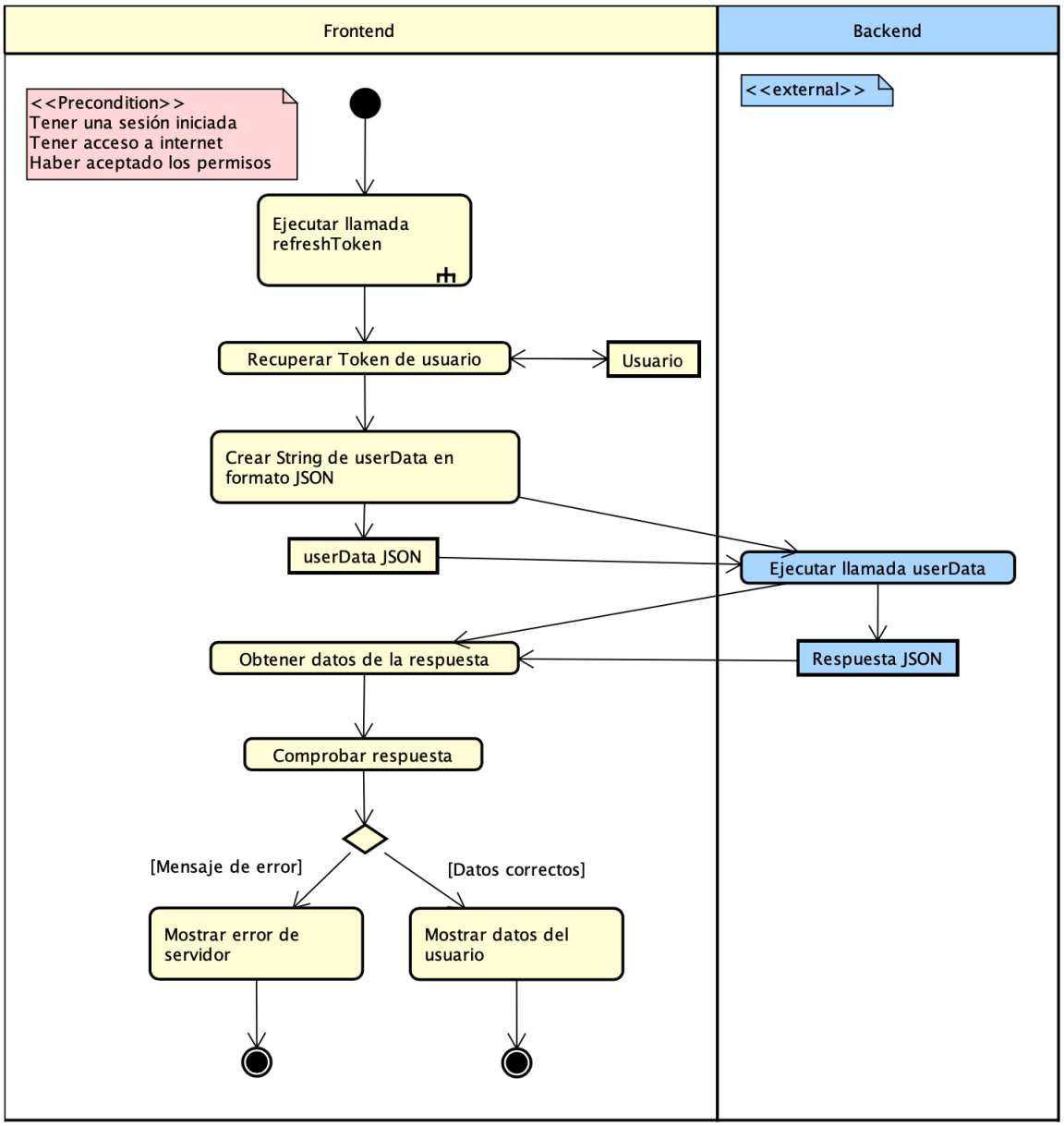


Figura 4.3: Diagrama de la historia de datos de usuario (HU02)

Ranking de usuarios (HU03)

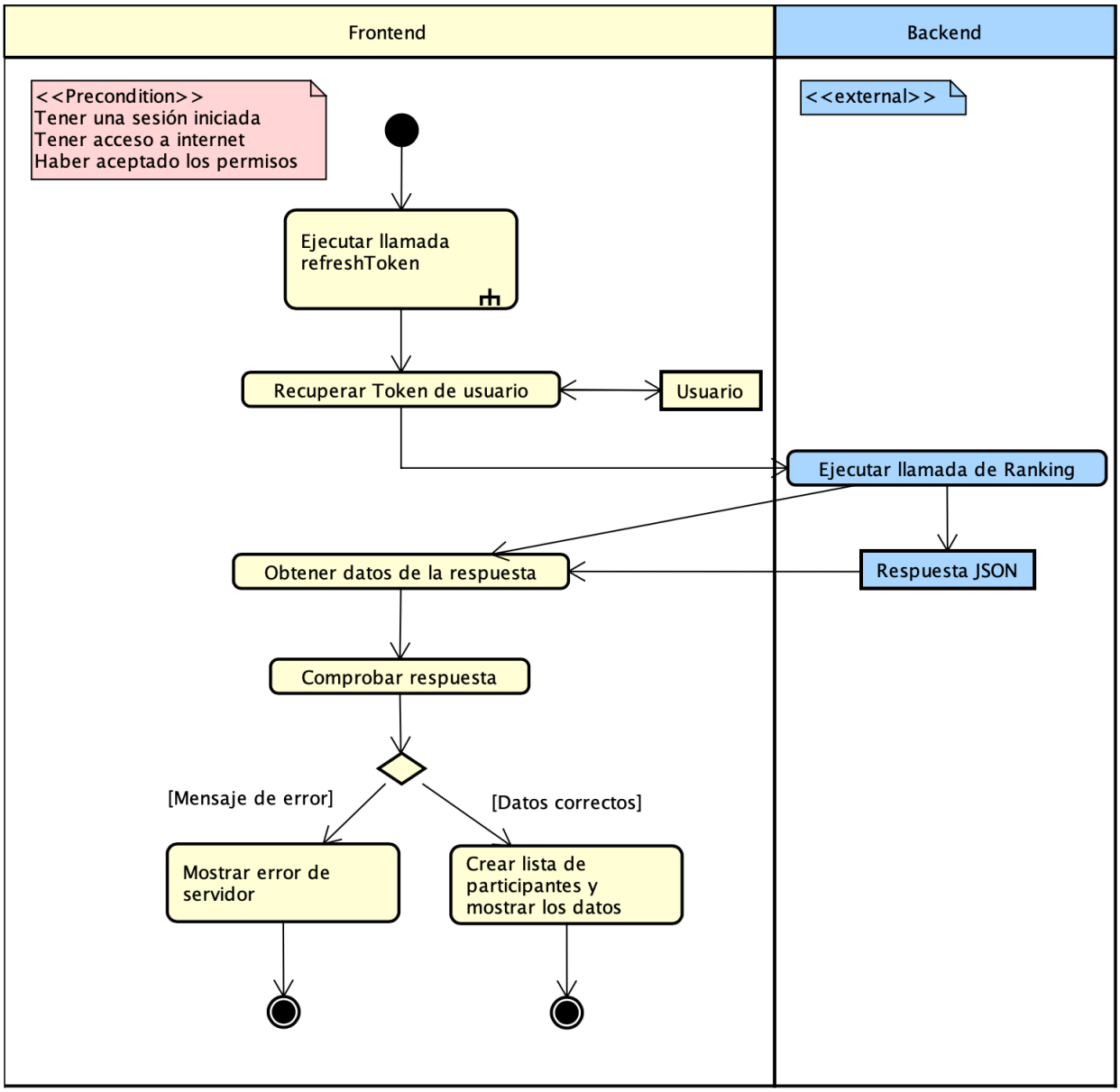


Figura 4.4: Diagrama de la historia de ranking de usuarios (HU03)

Registro de un nuevo usuario (HU04)

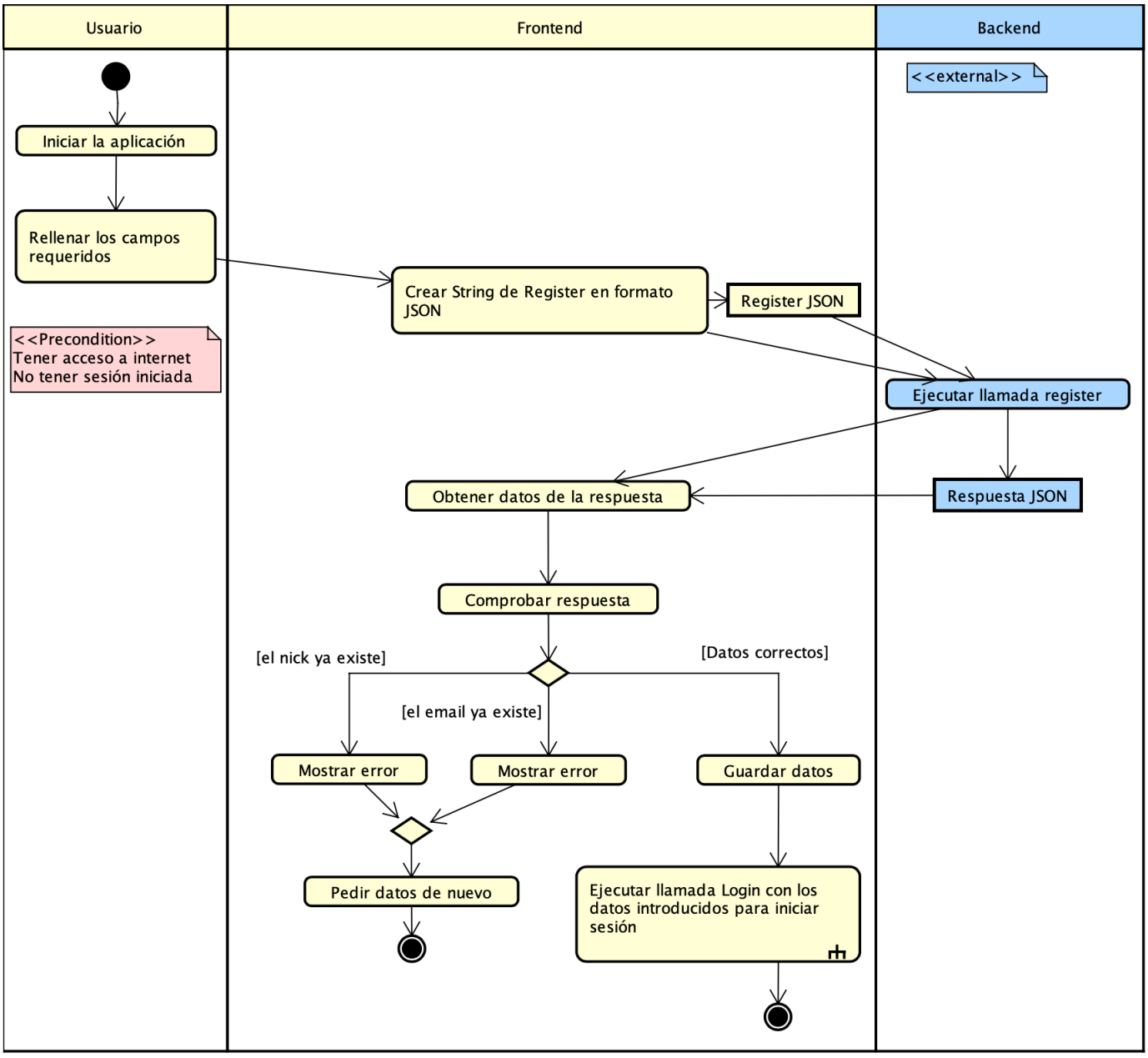


Figura 4.5: Diagrama de la historia de registro de un usuario (HU04)

Lista de partidas (HU05)

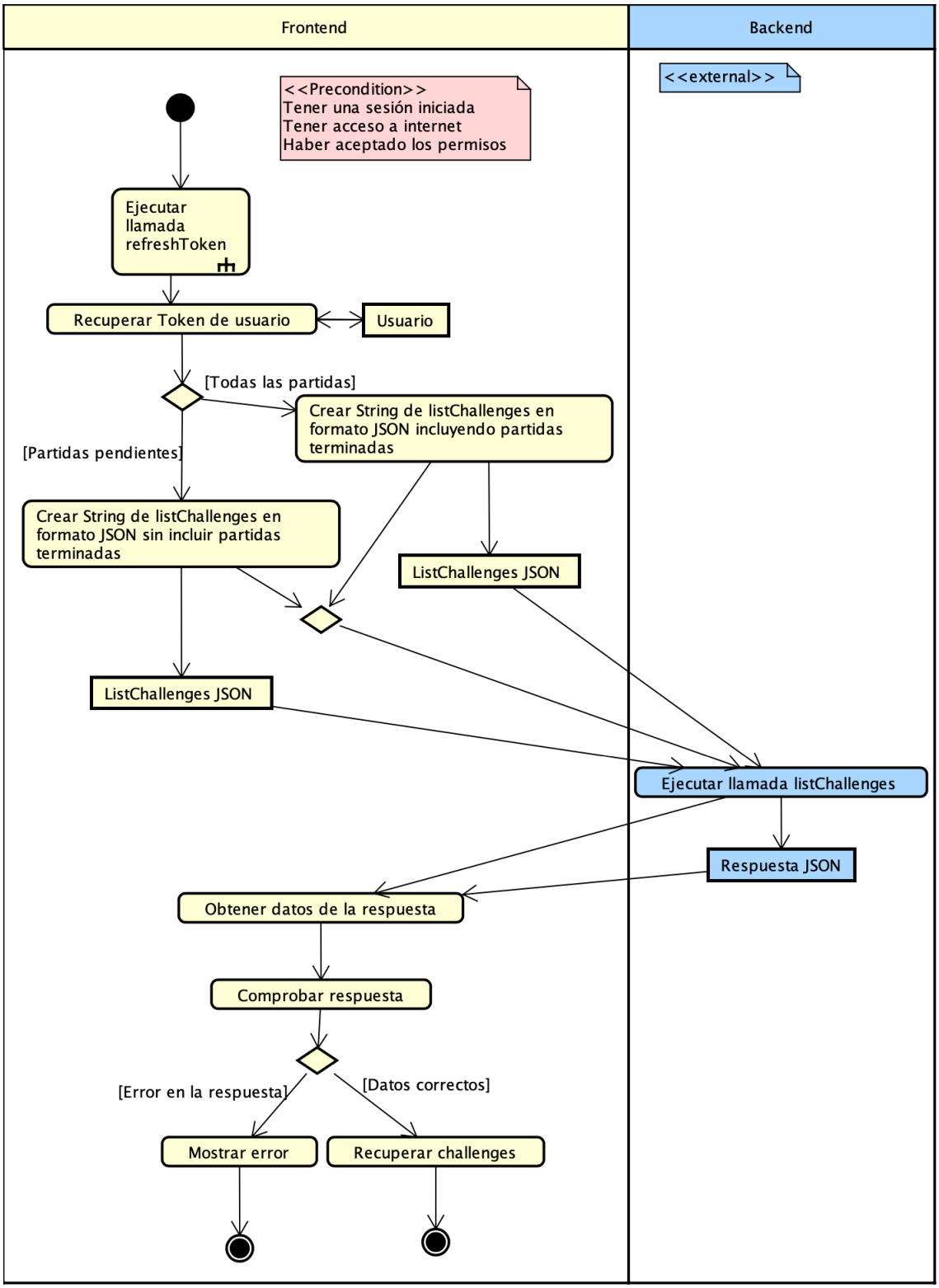


Figura 4.6: Diagrama de la historia de usuario de listar partidas (HU05)

Crear partida (HU06)

Como se puede ver a continuación (ver Figura 4.7), la historia de usuario crear partida está compuesta de varias acciones y en cada una de ellas se ejecuta una llamada de la API diferente. La primera acción para poder crear una partida es escoger los oponentes de esta para lo que se ejecuta la llamada de opponents (ver Figura 4.12). A continuación, si el usuario sigue el proceso tendrá que aceptar los oponentes y comenzará la partida. Antes de que la partida comience en el dispositivo se ejecutará la llamada a la API de createChallenge (ver Figura 4.7) y startMatch (ver Figura 4.13). Una vez que el usuario ha terminado la partida o la ha cerrado sin acabar se ejecutará la llamada de finishMatch (ver Figura 4.14).

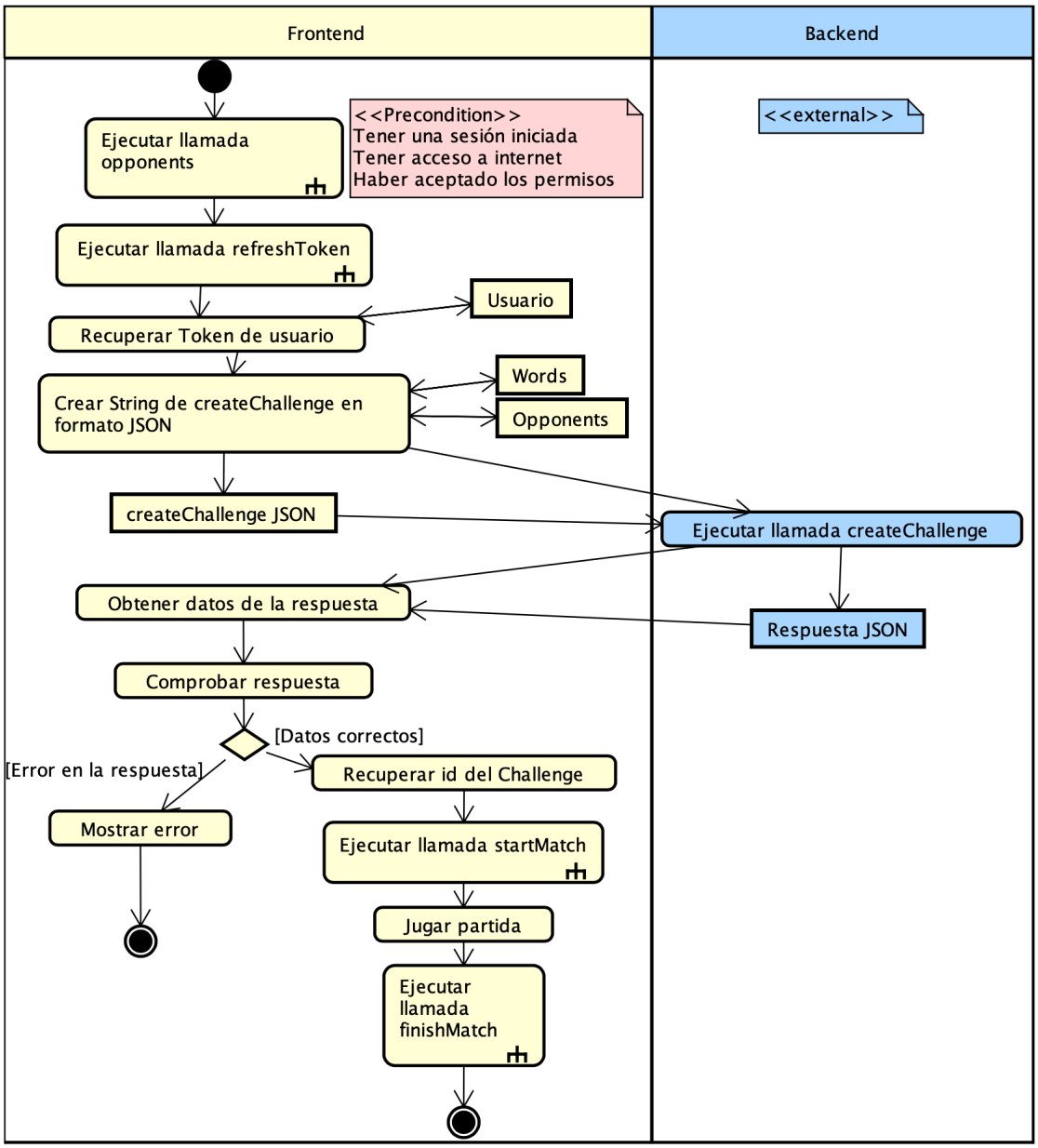


Figura 4.7: Diagrama de la historia de usuario de crear partidas (HU06)

Ver información de partida (HU07)

El usuario podrá acceder a la información de una partida concreta. Si aún no ha jugado su turno, podrá acceder a la partida pulsando el botón de jugar desencadenando las llamadas a la API de la Figura 4.13 al iniciar la partida y la llamada de la Figura 4.14 al terminar o cerrar la partida.

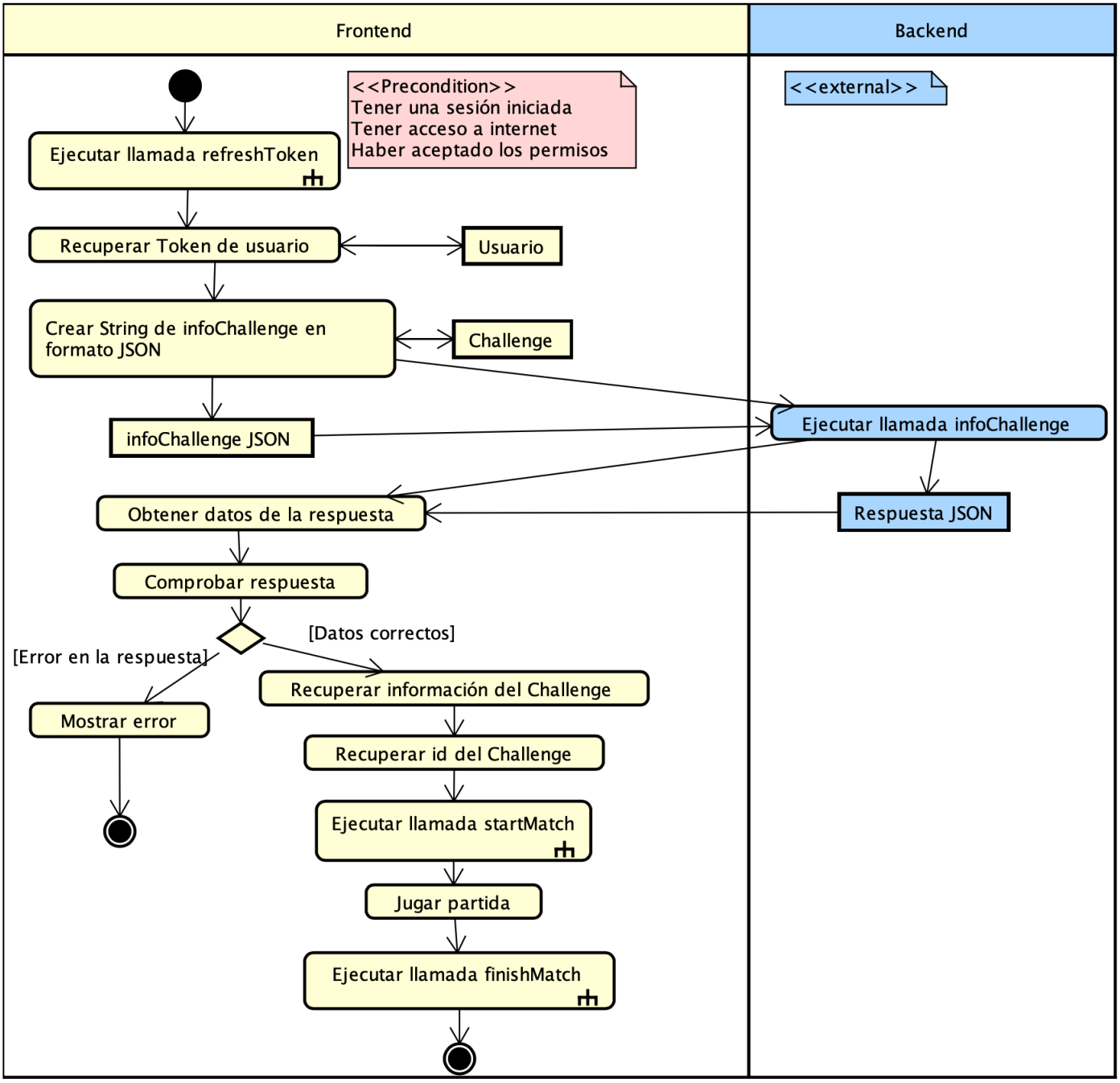


Figura 4.8: Diagrama de la historia de usuario de información de partida (HU07)

Ver logros (HU08)

En cualquier momento el usuario podrá ver sus logros pulsando en el icono del trofeo del menú. Esto desencadenará la llamada showAchievements que podemos ver en la Figura 4.9. Si no tiene ningún trofeo es porque aún no se ha ejecutado 4.15. Que se basa en ciertas condiciones de las puntuaciones y partidas jugadas para otorgar nuevos logros.

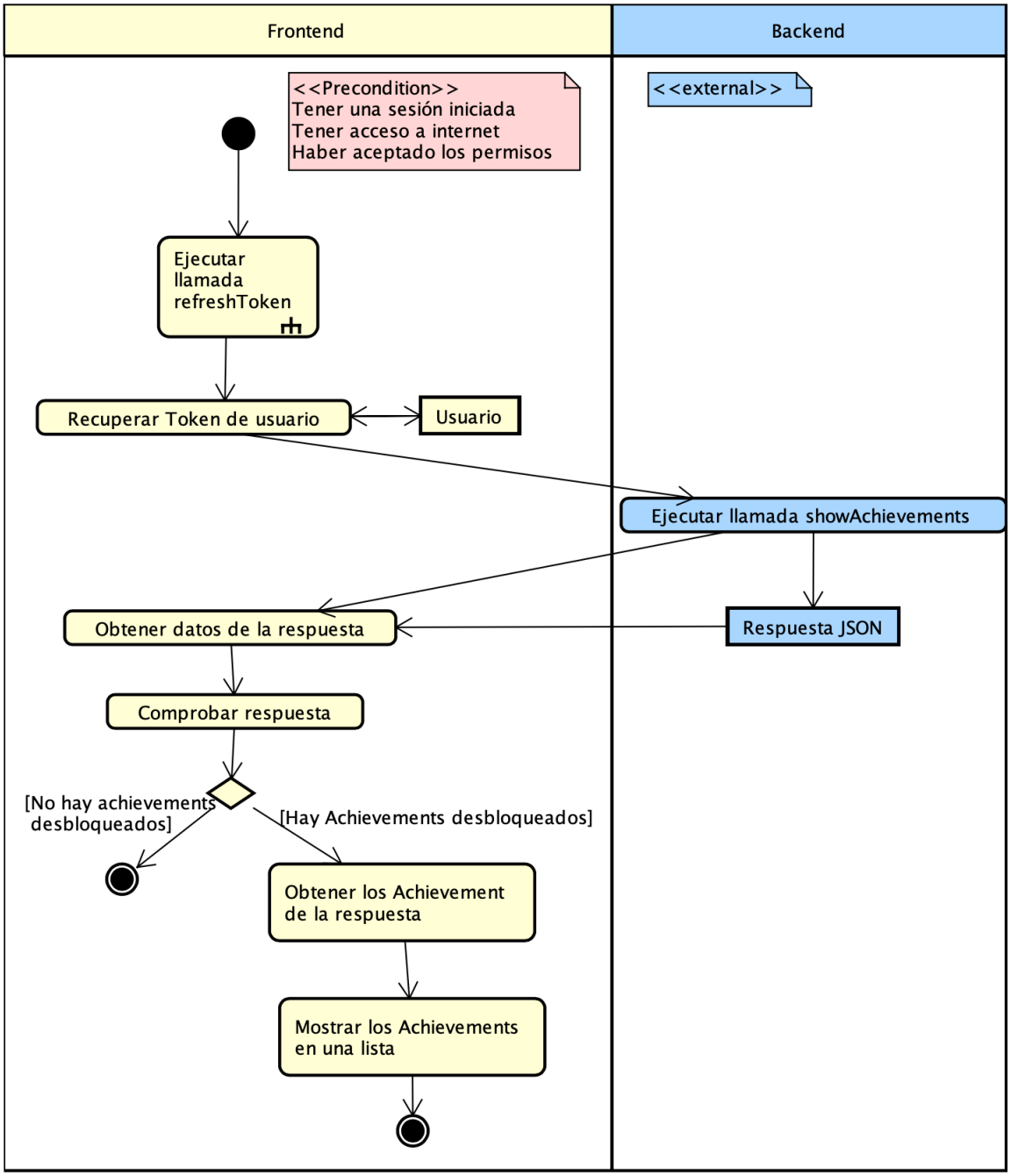


Figura 4.9: Diagrama de la historia de usuario de ver logros (HU08)

Modificar el perfil de usuario (HU09)

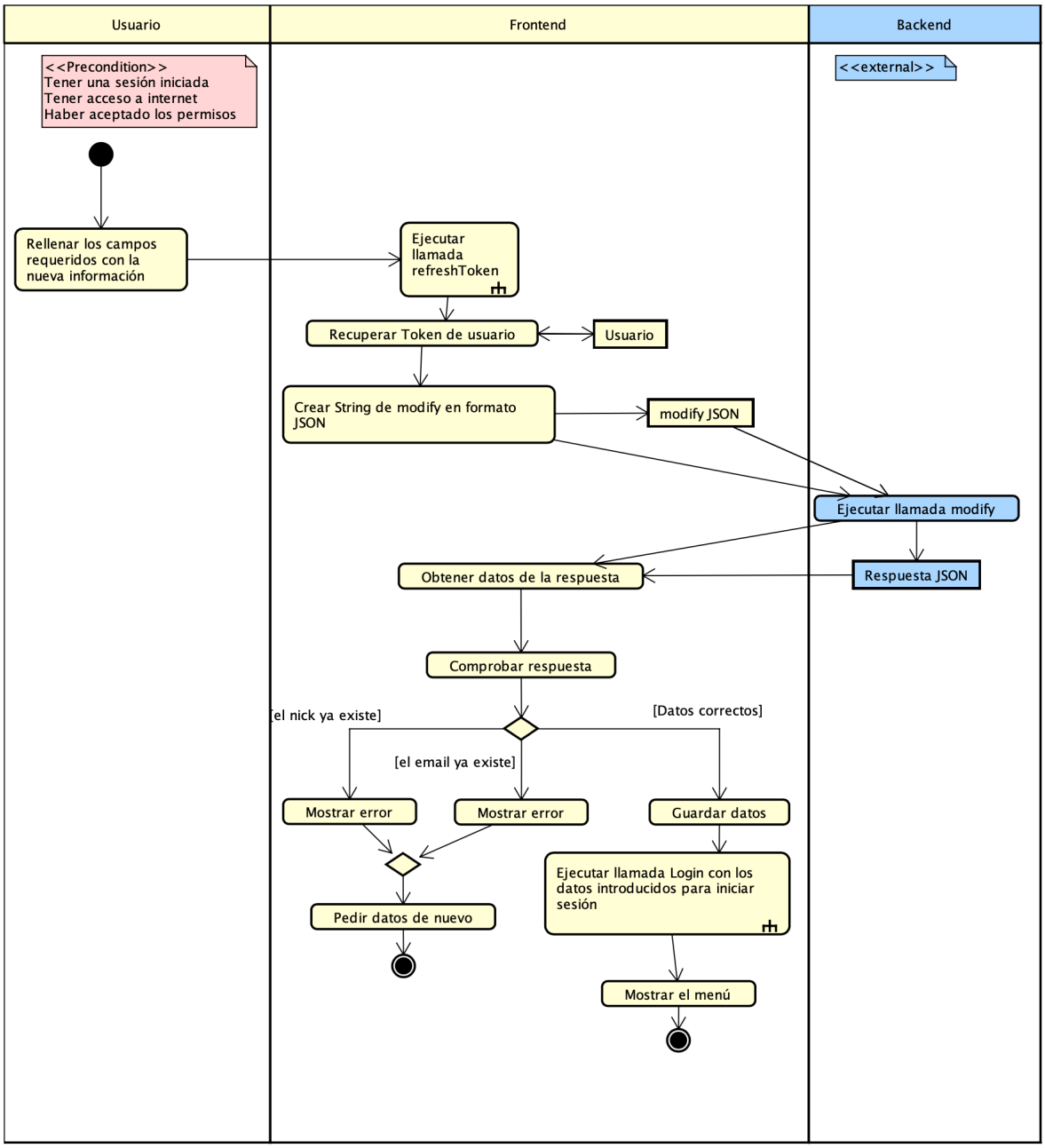


Figura 4.10: Diagrama de la historia de usuario de modificar el perfil (HU09)

Diagramas extra

Los siguientes diagramas no corresponde con ninguna historia de usuario. El diagrama de la Figura 4.11 es usado en la mayoría de estas. Como se ha comentado con anterioridad, se ejecuta refreshToken antes de cada llamada a excepción del login y registro. Esto permite solo mantener almacenado en el dispositivo el refreshToken. Con dicho refreshToken se obtiene el token de la sesión y siempre se mantiene actualizado.

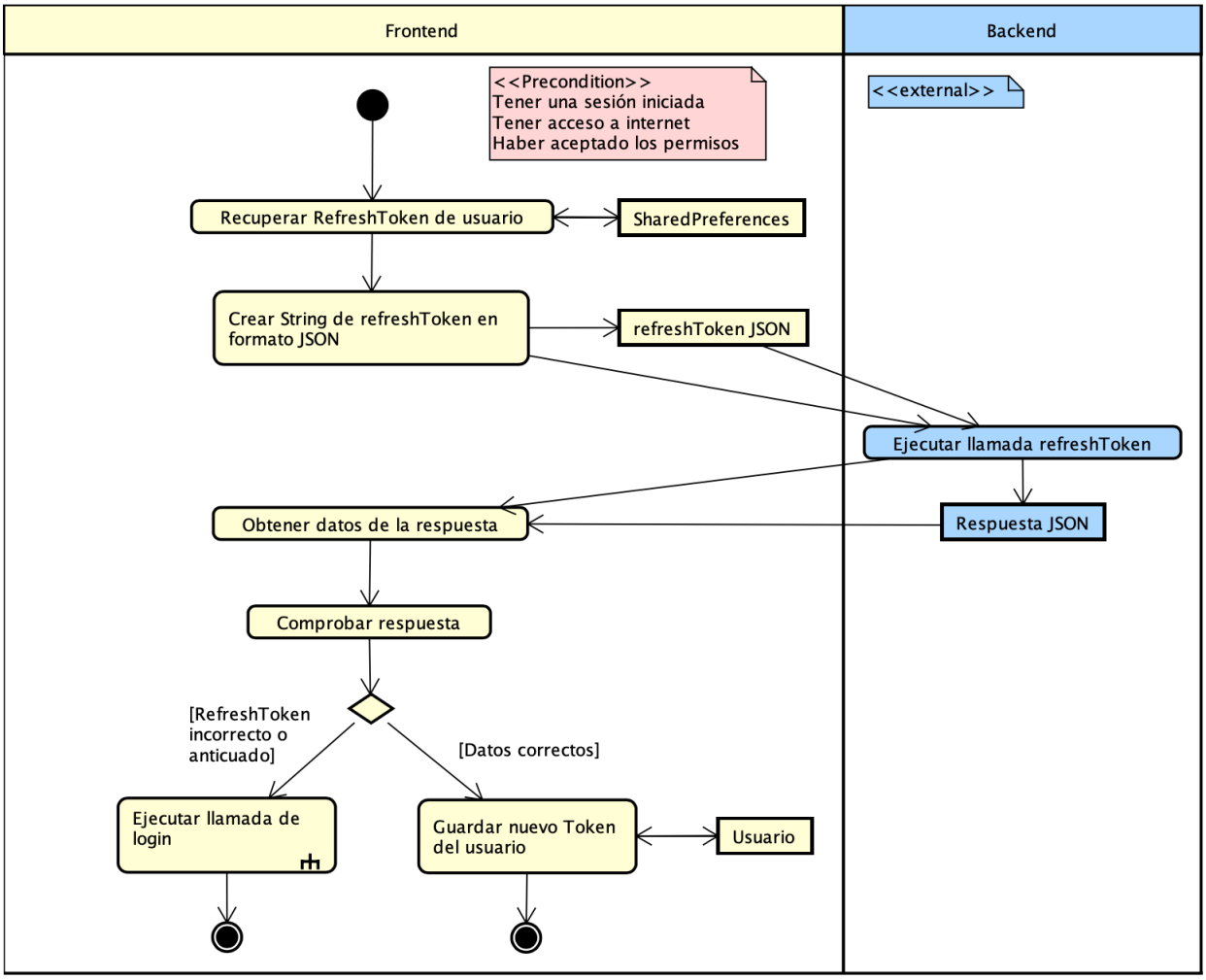


Figura 4.11: Diagrama de la llamada refreshToken de la API

A continuación, se muestran los diagramas extra que se ejecutan en la historia de usuario de crear partida (HU6) (ver Figura 4.12, Figura 4.13 y Figura 4.14). Tanto la Figura 4.13 como la Figura 4.14 se usan también en ls historia de ver información de una partida (HU07).

Por último, se presenta el diagrama que muestra como añadir nuevos logros a un usuario, lo que permite luego verlos (ver 4.15) en la historia de usuario de ver logros (HU08).

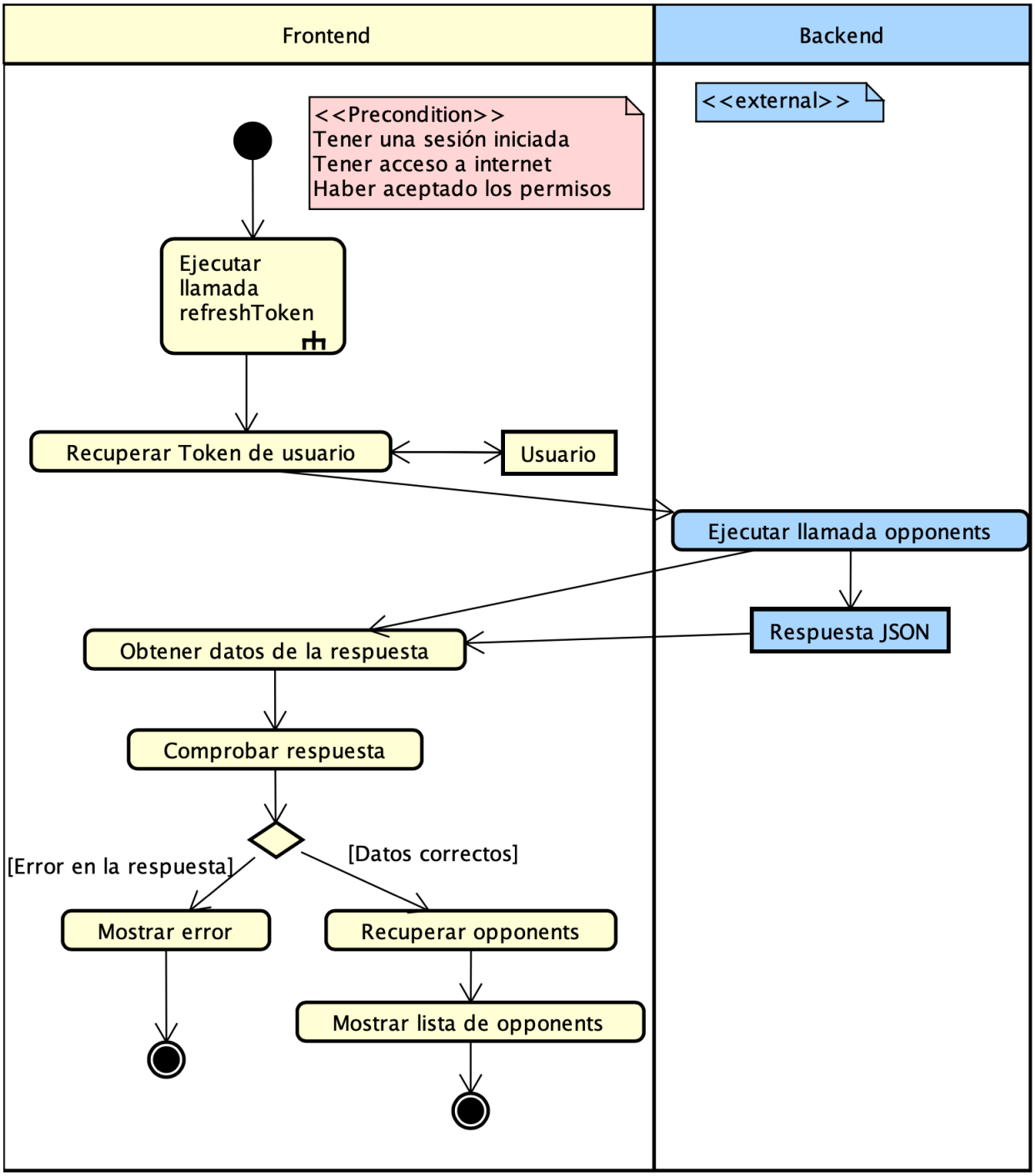


Figura 4.12: Diagrama extra de la historia de usuario de crear partidas (HU06)

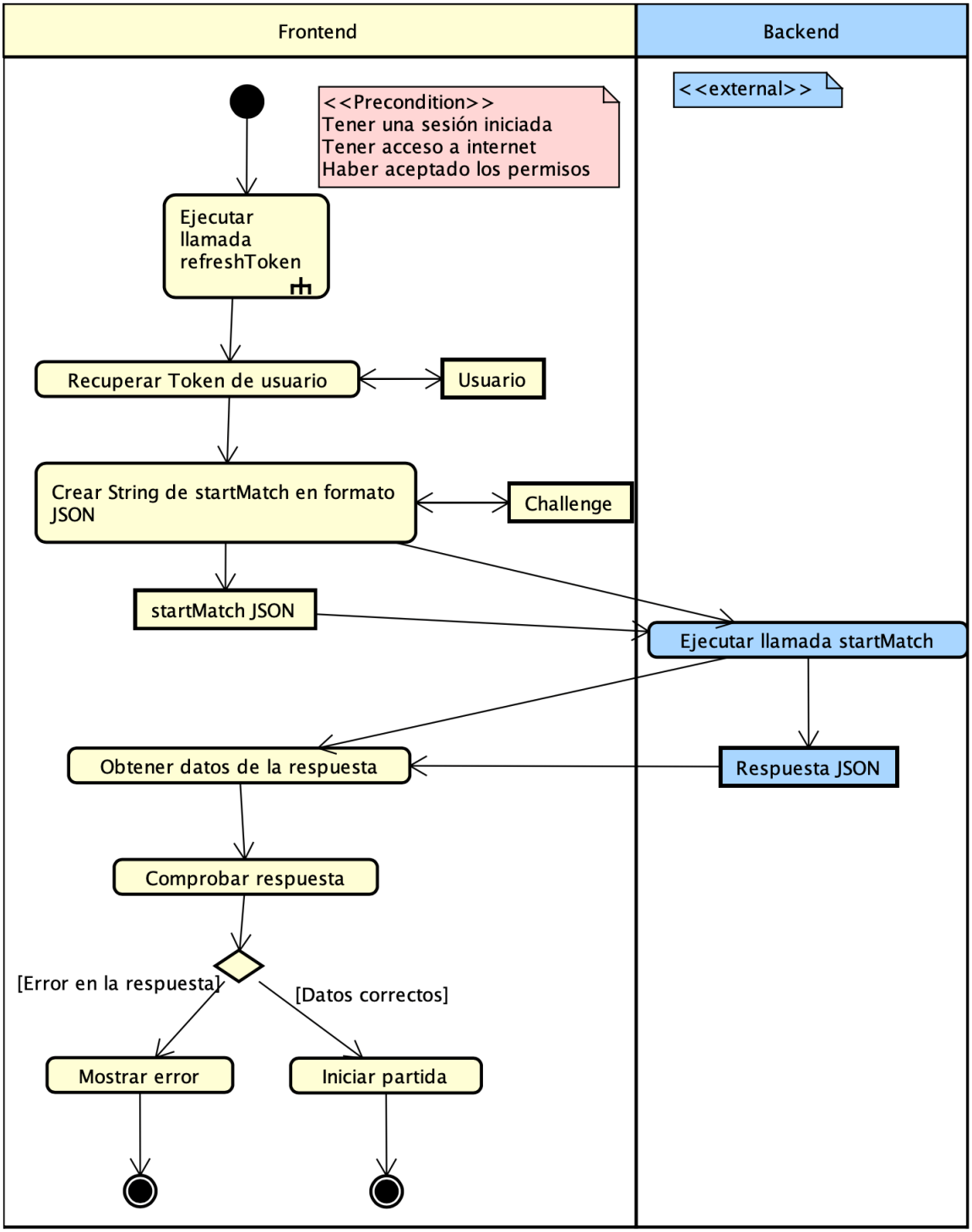


Figura 4.13: Diagrama extra de la historia de usuario de crear partidas (HU6) y de la historia de usuario de ver la información de una partida (HU07)

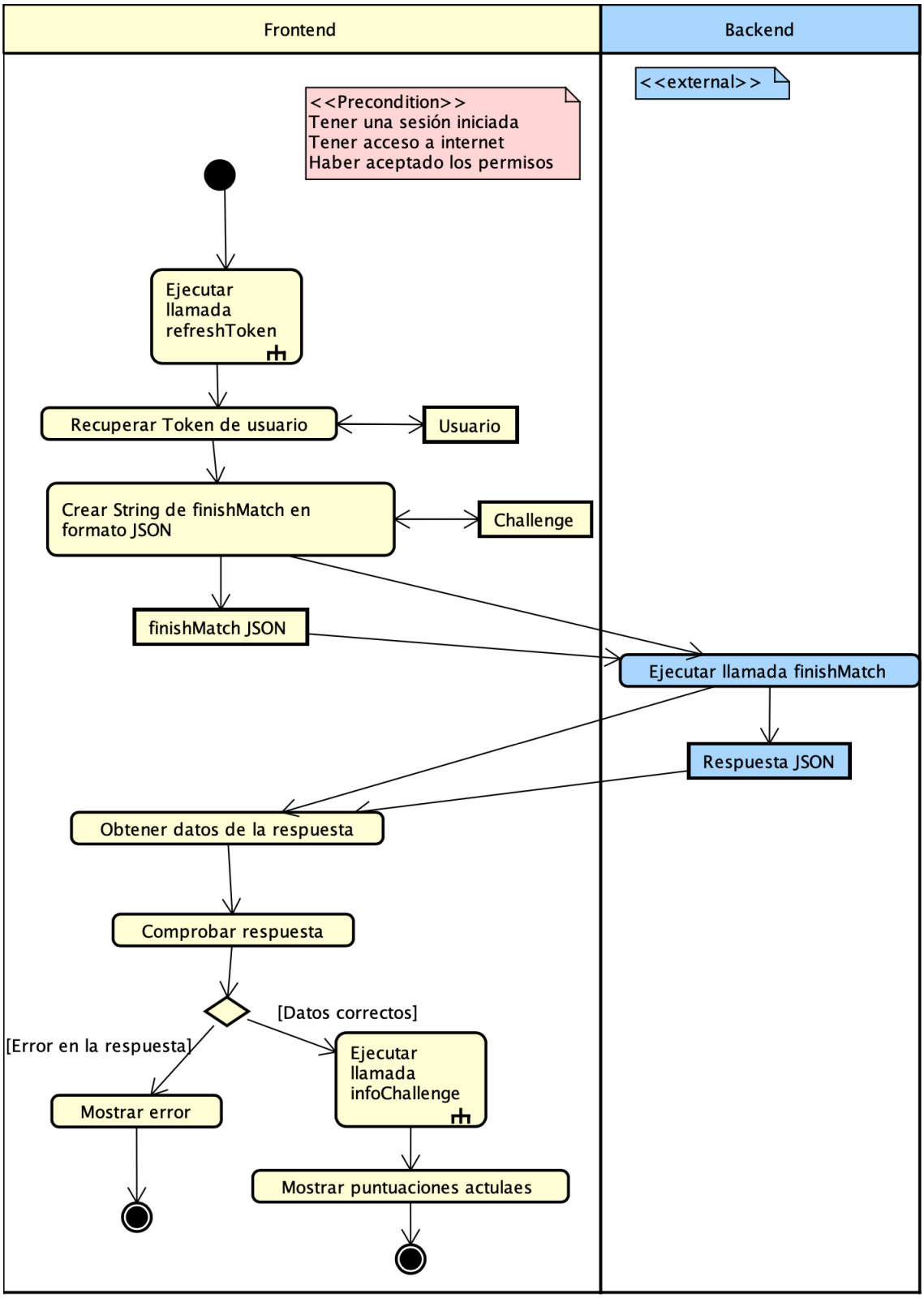


Figura 4.14: Diagrama extra de la historia de usuario de crear partidas (HU06) y de la historia de usuario de ver la información de una partida (HU07)

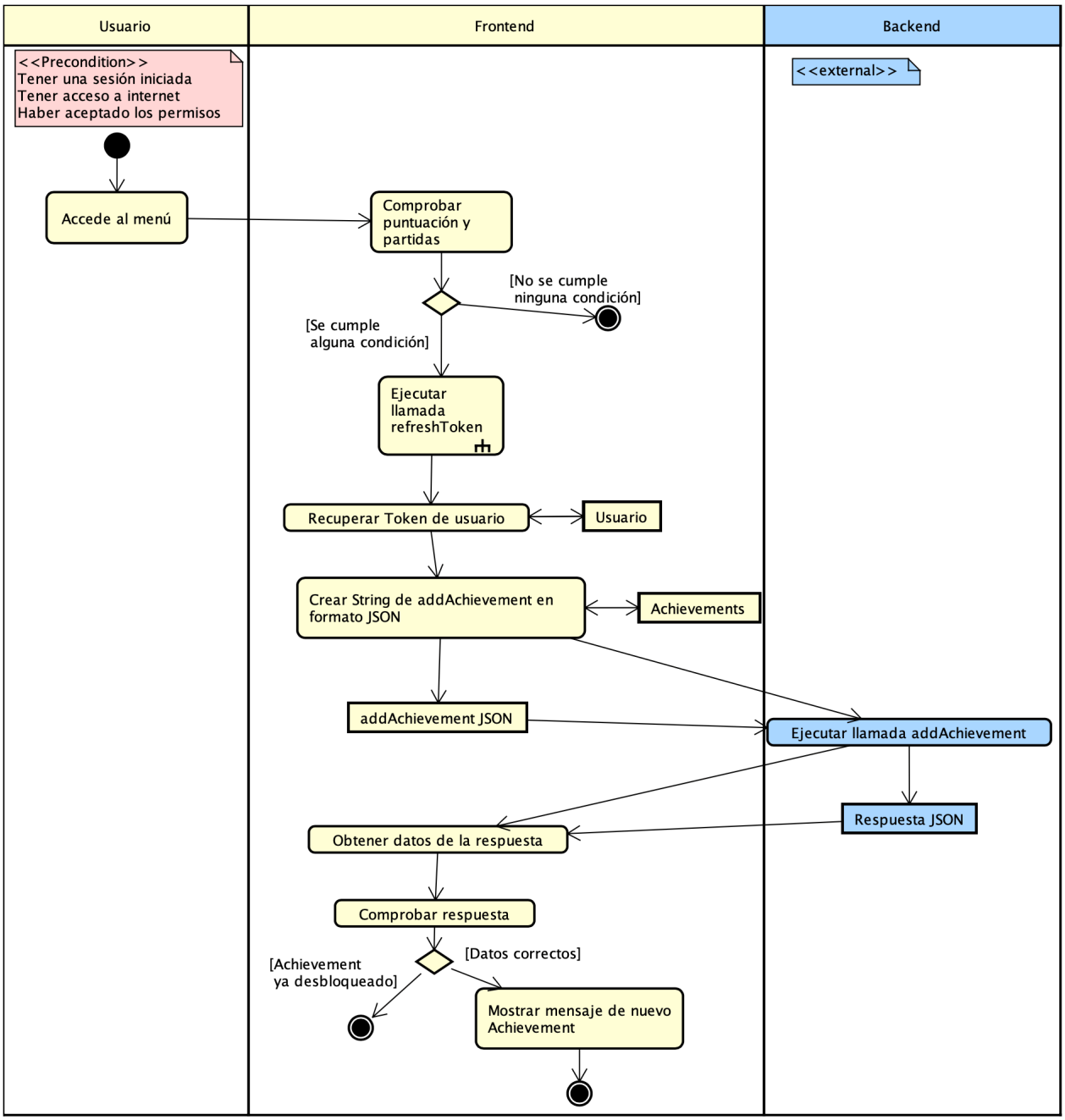


Figura 4.15: Diagrama extra de la historia de usuario de ver logros (HU08)

4.2. Estructura del proyecto

En la Figura 4.16 podemos ver la estructura de la aplicación CoP. Es necesario destacar que esta no es la estructura de ficheros real, ya que estamos utilizando la vista de Android (por defecto en Android Studio) que facilita ver todas las carpetas y recursos del proyecto.

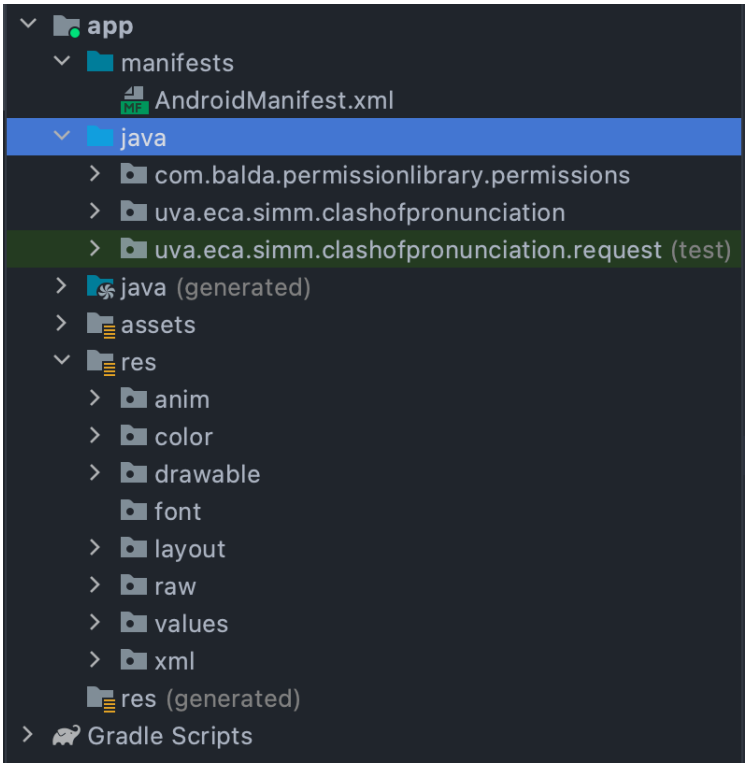


Figura 4.16: Estructura de CoP

En la figura podemos identificar en primer lugar la carpeta ”manifests”. En ella se encuentra el Manifest. En este fichero se describen los permisos a utilizar en la aplicación así como las Activities que la aplicación contenga. A continuación encontramos la carpeta java. En esta carpeta están contenidos todos los paquetes de CoP. Los primeros paquetes que encontramos son el de permisos (”permissions”) y el paquete ”clashofpronunciations”. Dentro de este último es donde encontramos la mayoría de paquetes de CoP que serán explicados en los siguientes apartados. Además de estos, podemos ver el paquete de test en el que se encuentran las clases con los test realizados.

Una vez vistos los paquetes principales de la aplicación se van a explicar las carpetas de recursos de Android. Como se puede en la Figura 4.16 todos los recursos del proyecto se encuentran bajo la carpeta ”res”. En ella encontramos varios directorios de recursos diferentes:

- El primer directorio es el de ”anim” que contiene las animaciones que tenga el proyecto.
- A continuación vemos el directorio ”color”, que contiene los xml de colores que definamos.
- El directorio ”drawable” almacena diferentes formas o elementos visuales que luego pueden añadirse a las vistas.

- El directorio ”font” almacena las diferentes fuentes de texto que contenga nuestro proyecto.
- El directorio ”layout” contiene todos los xml de todas las vistas de la aplicación. Estos xml son cargados desde las Activities en su creación para dar lugar a la vista.
- El directorio ”raw” contiene los ficheros de audio.
- El directorio ”values” almacena varios directorios más entre los que hay que destacar el directorio ”string” que guarda todas los textos del proyecto y sus traducciones y el directorio ”styles” que almacena configuraciones de estilos que luego pueden ser usados en las diferentes vistas para editar el estilo de un botón o de cualquier otro elemento.
- El directorio ”xml” contiene un fichero con datos de dominios de red.

4.3. Diagramas de diseño

A lo largo de esta sección se presentarán los patrones y técnicas de diseño utilizadas durante el proceso de migración y actualización de la aplicación. Además, se expondrán los paquetes nuevos o actualizados, las clases de cada paquete que han sido modificadas o creadas de cero, el diagrama de la base de datos y finalmente un diagrama de despliegue de la aplicación.

4.3.1. Arquitectura y patrones de diseño

Antes de comenzar es necesario resaltar que durante el proceso de migración y actualización solo nos hemos centrado en la arquitectura de las nuevas clases a implementar, manteniendo el diseño anterior del resto de clases (ver en [9] y [1]).

Arquitectura de la aplicación

La aplicación se encuentra distribuida en 3 capas (ver Figura 4.17), siendo la primera la capa de presentación, en la que encontramos el patrón MVVM. La segunda es la capa de negocio, que mantiene la mayoría de paquetes de la aplicación y es donde se ejecuta la lógica de los casos de uso. La última capa será la de servicios y en ella encontramos los paquetes que se encargar de la escritura de ficheros y de realizar llamadas http. Este tipo de arquitectura es el recomendado por Android como podemos ver en la Figura 4.18.

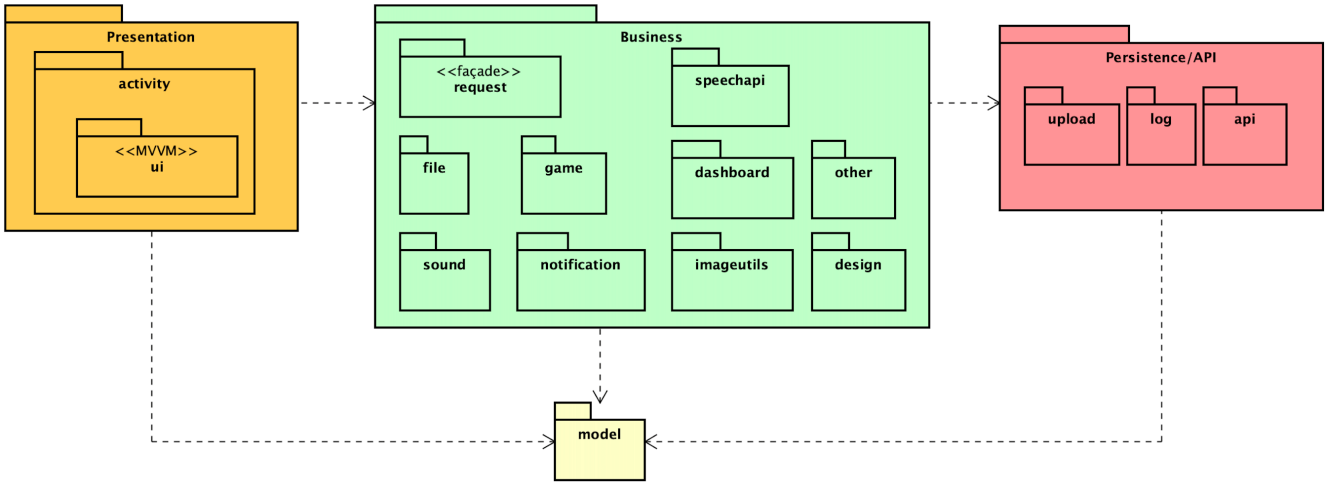


Figura 4.17: Arquitectura de CoP

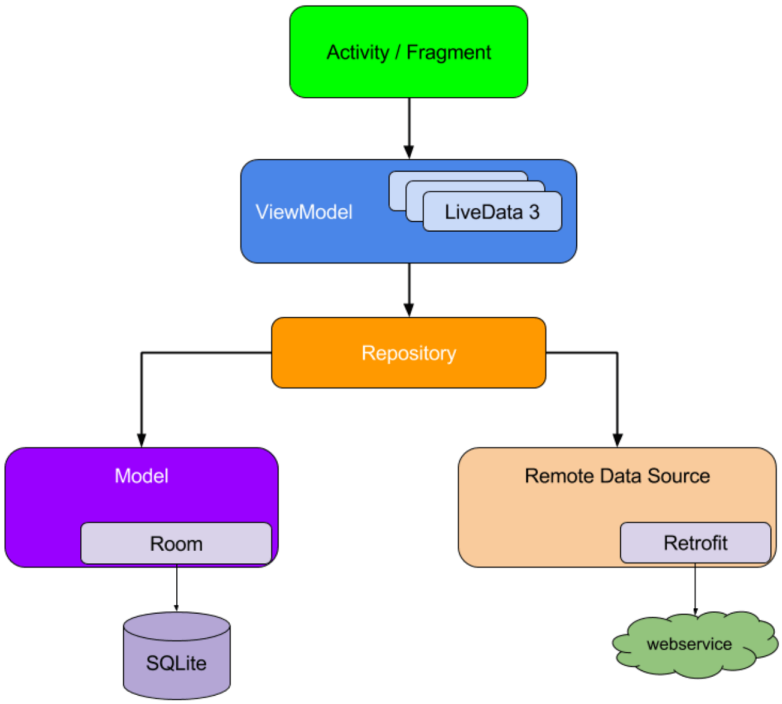


Figura 4.18: Arquitectura de una aplicación Android. Fuente: Android developers [58]

Patrón de presentación MVVM y clase LiveData

Como diseño de las nuevas vistas se ha propuesto el patrón de presentación MVVM (Model View ViewModel). Este es el patrón recomendado por Android en la actualidad [58]. Este patrón permite aislar la lógica que se encarga de la vista únicamente en una clase (Activities en nuestro caso) y mantener el estado de los datos mostrados en la vista en el ViewModel junto con el resto de la lógica. Por tanto, cada pantalla de la aplicación debería tener su Activity y su ViewModel y además, poder acceder a los

datos del modelo. Cuando el ViewModel ejecuta diferentes acciones en segundo plano, como realizar una llamada a una API o una consulta en una base de datos, este almacena los cambios en variables propias de tipo MutableLiveData [59]. Los ViewModel deben tener métodos que retornen variables LiveData (como MutableLiveData pero sin que pueda ser modificada). Estas variables LiveData puedes ser observadas desde las vistas. De esta manera, una Activity observa las diferentes variables LiveData de su ViewModel. El ViewModel ejecuta diferentes acciones en segundo plano y cuando recupera su información la almacena en los diferentes MutableLiveData que mantiene. El MutableLiveData notificará a todos los observadores (recordamos que la Activity está observando estas variables) que ha habido un cambio de estado y la Activity será capaz de responder a este evento realizando algún tipo de acción en la vista.

Patrón Observador

El patrón observador permite que un objeto A se suscriba a otro B, de manera que cuando el objeto B sufre algún cambio en su estado, este notifica a los objetos que se han suscrito a él que ha habido cambios. De esta manera, el objeto A puede reaccionar a estos cambios (ver Figura 4.19). En al actualidad la mayoría de interfaces de usuario de frameworks modernos hacen uso de este patrón. En Android permite no bloquear el hilo principal para que este pueda seguir realizando acciones o escuchando eventos de usuario mientras se ejecutan acciones en segundo plano y reaccionar a estas una vez se han llevado a cabo.

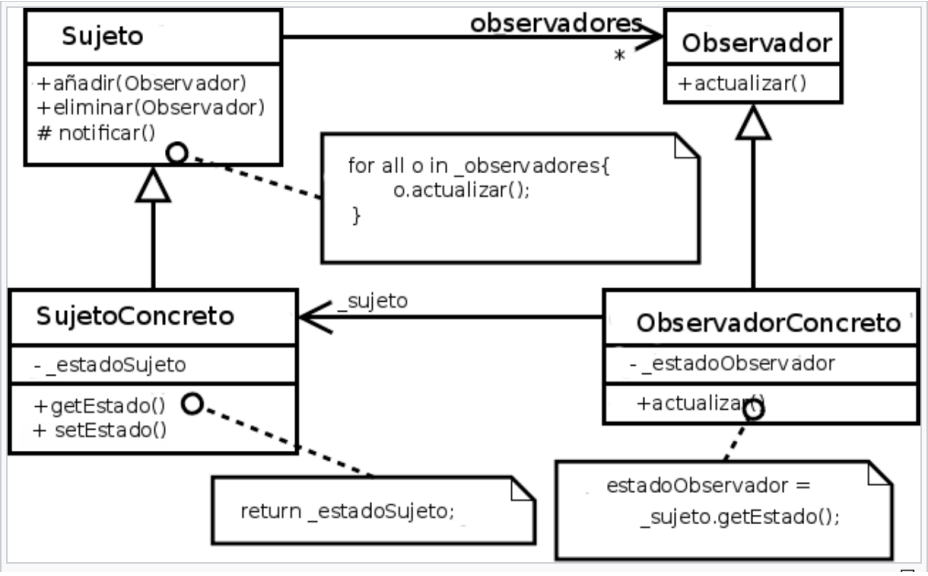


Figura 4.19: Estructura del Patrón Observador. Fuente: wikipedia [60]

Como se ha comentado en el apartado anterior, este patrón se ha usado en CoP para observar las diferentes acciones que ejecuta el ViewModel y actuar una vez se notifican los cambios.

Patrón Singleton

El Patrón Singleton permite mantener una única instancia de un objeto en memoria. Para llevarlo a cabo el patrón almacena su propio estado como atributo y mantiene privado su constructor. De esta manera, si otro objeto B desea hacer uso del objeto Singleton, dicho objeto Singleton creará la instancia o devolverá la creada anteriormente (ver Figura 4.20). En CoP este patrón ha sido utilizado para mantener el estado del usuario registrado una vez se ha recuperado su información.

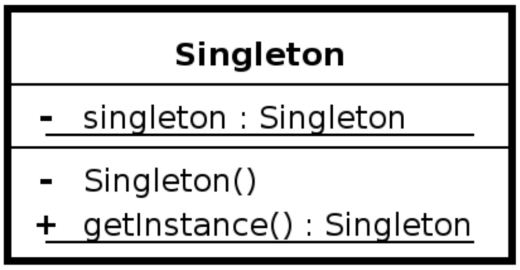


Figura 4.20: Estructura del Patrón Singleton. Fuente: wikipedia [61]

Patrón Fachada

El Patrón Fachada permite acceder a un paquete con varias clases a través de una única clase y así centralizar todas las llamadas entre paquetes en un único lugar. En CoP este patrón se ha usado en el nuevo paquete creado para ejecutar las llamadas al servidor. Ese paquete (Request) contiene todas las acciones necesarias para ejecutar las diferentes llamadas y se encarga de acceder al resto de clases (que contienen funciones para obtener los datos) (ver Figura 4.21).

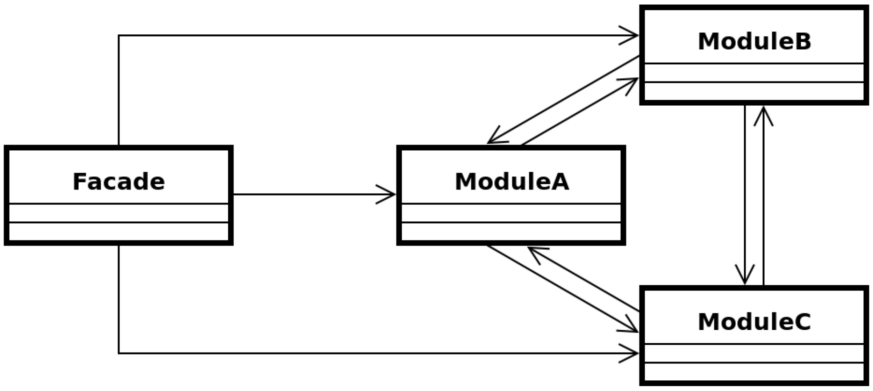


Figura 4.21: Estructura del Patrón Fachada. Fuente: wikipedia [62]

4.3.2. Diagrama de paquetes

En la Figura 4.22 se muestra la distribución de las clases en los paquetes de la aplicación. Es necesario destacar que aquí se muestran todos los paquetes que contiene la aplicación pero no todos ellos han sido actualizados. Los paquete que han recibido modificaciones se encuentran en color verde y los paquetes que se han creado de cero el color azul.

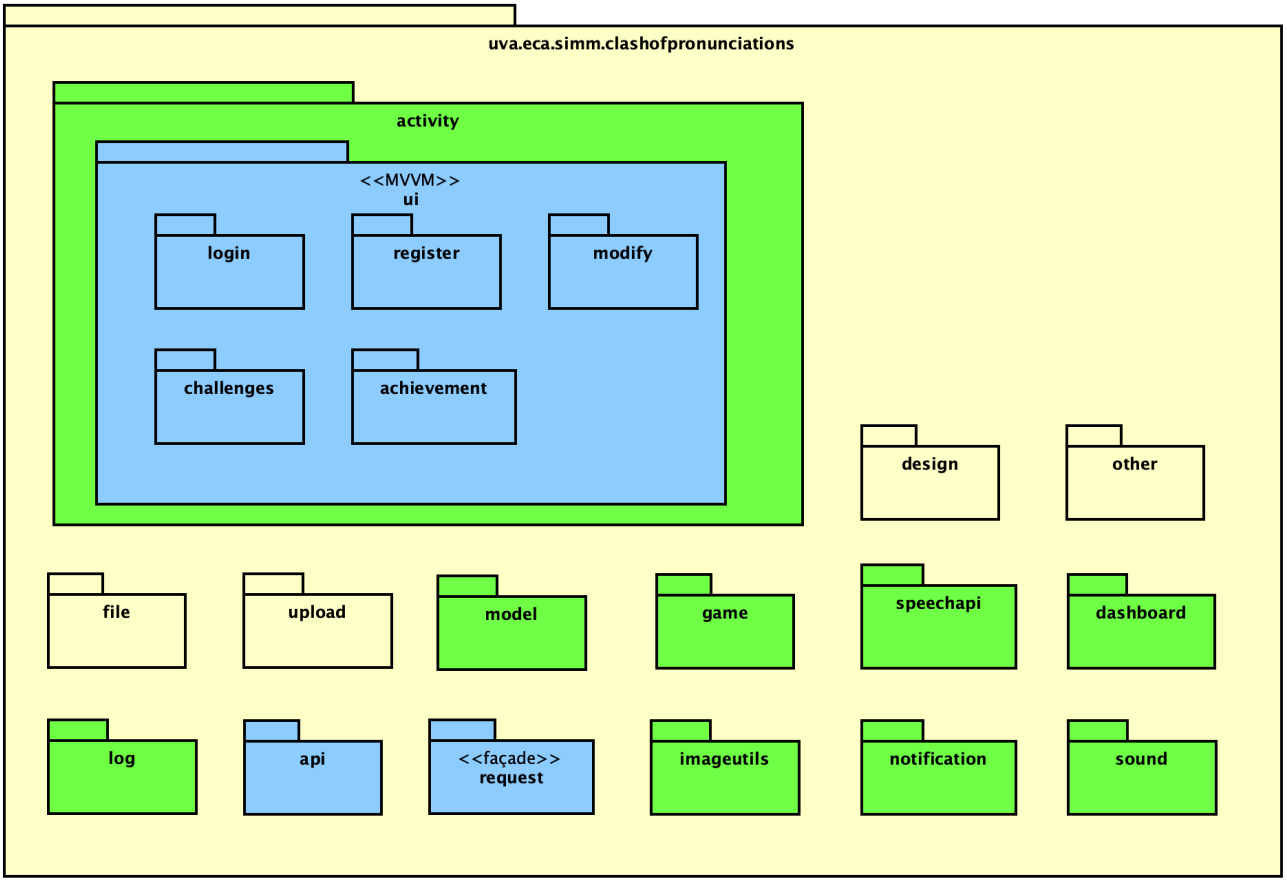


Figura 4.22: Diagrama de paquetes de Clash of Pronunciations

El paquete activity contiene todas las clases de tipo Activity de la CoP y además, contiene el nuevo paquete creado (ui) en el que se incluyen las nuevas Activities creadas y sus ViewModels siguiendo el patrón MVVM. Además de activity, tenemos el paquete model, que almacena todas las clases del modelo. Y los paquetes pertenecientes a la lógica de negocio. Dentro de esta lógica de negocio encontramos los paquetes design, que contiene clases encargadas de la funcionalidad de ciertos elementos visuales; game contiene clases que ayudan a ciertas acciones como obtener datos de los logros dado un id; speechapi contiene las clases encargadas del reconocimiento; dashboard se encarga de recibir tareas desde un dashboard web para mostrarlas en la app y que los usuarios puedan realizarlas (relacionado con unas prácticas en empresa); imageutils contiene clases encargadas de la vista usada para mostrar la imagen del perfil; notification es el paquete encargado de las notificaciones; sound es el paquete que contiene las clases que permiten reproducir o reconocer la voz; other que contiene clases encargadas de la valoración de la aplicación; finalmente el paquete request contiene las clases relacionadas con

los casos de uso de las llamadas a la API, así como las clases que contienen las funciones de cada llamada. En request se crean los mensajes a enviar a la API a través de las clases del modelo y se leen y tranforman las respuestas de la API para obtener las clases del model necesarias para las vistas.

El siguiente paquete es file, que contiene las clases encargadas de leer o crear ficheros. El paquete log y upload contienen las clases que crean los logs y las clases encargadas de subir los ficheros al servidor de ECA-SIMM. El paquete api es el que contiene los ficheros para realizar conexiones http usadas en las llamadas a la API.

4.3.3. Diagrama de clases

En la Figura 4.23 y Figura 4.24 se muestran las diferentes clases que encontramos en cada paquete. Como podemos ver, al igual que en el apartado anterior, las clases en color verde han sido modificadas o editadas durante el desarrollo y las clases en azul han sido creadas de cero.

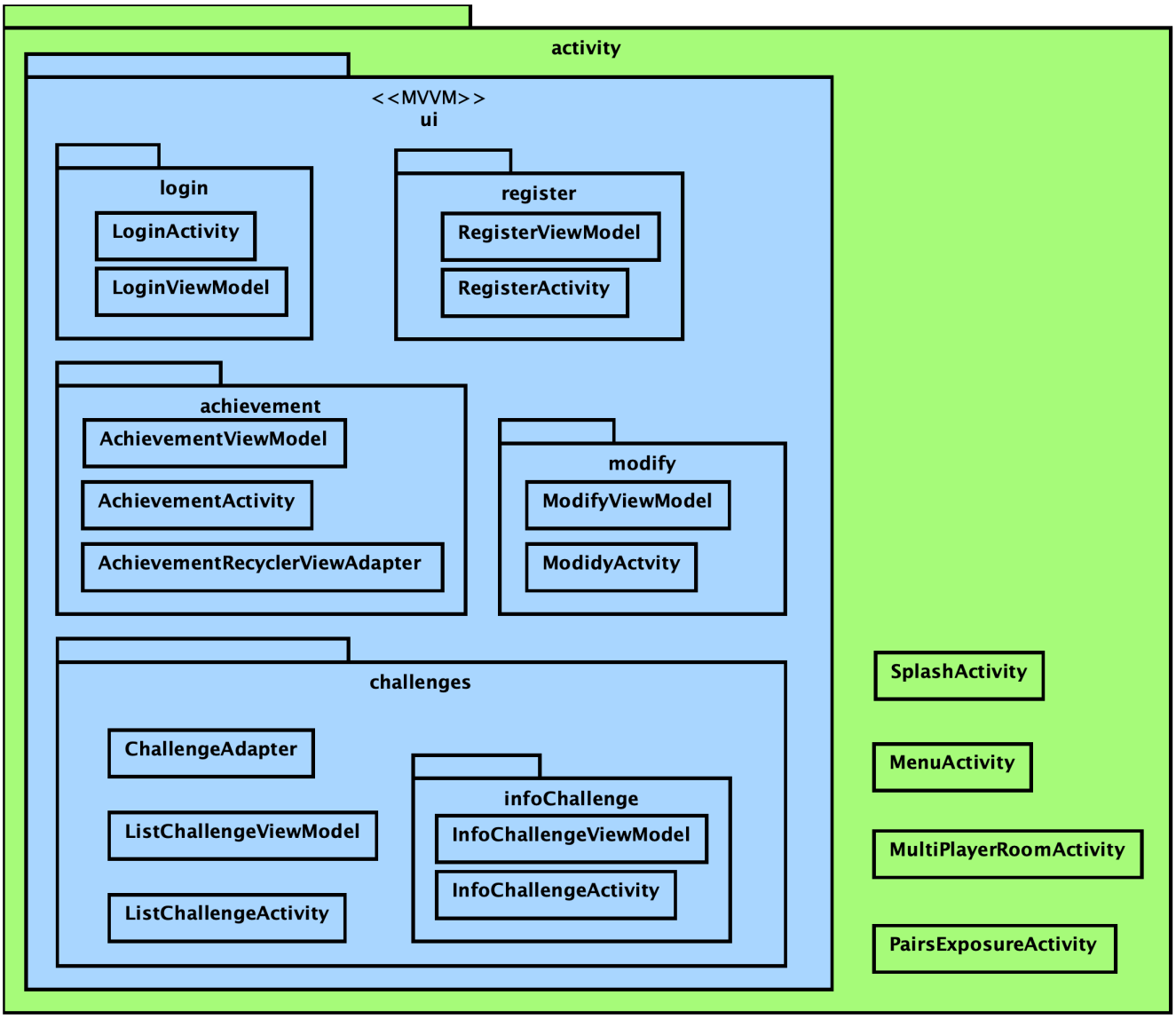


Figura 4.23: Diagrama de clases del paquete activity.ui

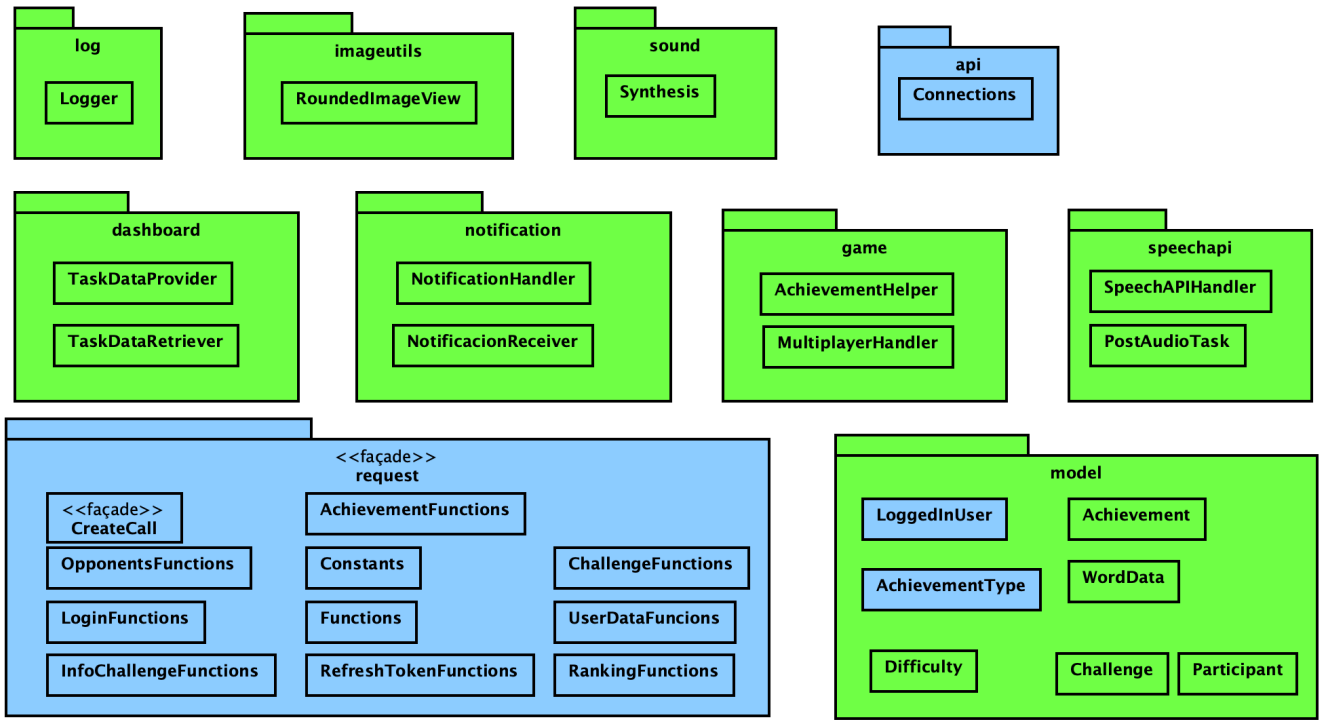


Figura 4.24: Diagrama de clases del resto de paquetes

Dentro del paquete activity.ui encontramos los diferentes paquetes de cada nueva vista implementada. Además podemos ver las Activities que ya existían y que han sido modificadas. Hay que resaltar que las Activities son las clases que contienen la lógica exclusivamente encargada de la vista. Los ViewModel almacenan el estado de los datos representados para evitar su pérdida si hay que rehacer la vista (como al pasar al rotar el dispositivo móvil). Por último, las clases que contienen la palabra ”Adapter” en su nombre se encargan de definir la información que se muestra en los listados que sus Activities contienen.

El paquete ”model” solo contiene clases del modelo. A destacar la clase LoggedInUser que, usando el patrón Singleton, mantiene los datos del usuario en memoria.

El último paquete que es necesario explicar más en profundidad es el paquete ”request”. Este presenta una clase principal (”CreateCall”), que se encarga de aglutinar todas las llamadas de la API. Desde fuera del paquete solo se accede a esta clase principal y es ella la que se encarga de llamar al resto de clases del paquete. El resto de clases son funciones que forman objetos **json** para enviar datos a la API o bien recuperan la información de las respuestas obtenidas. Este paquete se comunica con el paquete ”api” de la capa de persistencia/API para realizar las conexiones http.

4.3.4. Diagrama de la base de datos

En la ejecución de este proyecto no se ha creado una base de datos ya que no ha sido necesaria porque los datos son almacenados por la API en el servidor. A pesar de esto, sí que se han tenido que crear scripts para insertar datos en la base de datos y se ha tenido que desplegar y crear un manual

despliegue. Por esta razón, en la Figura 4.25 se muestra el diseño de la base de datos procedente del TFG de Andrés de la Maza Valles [6].

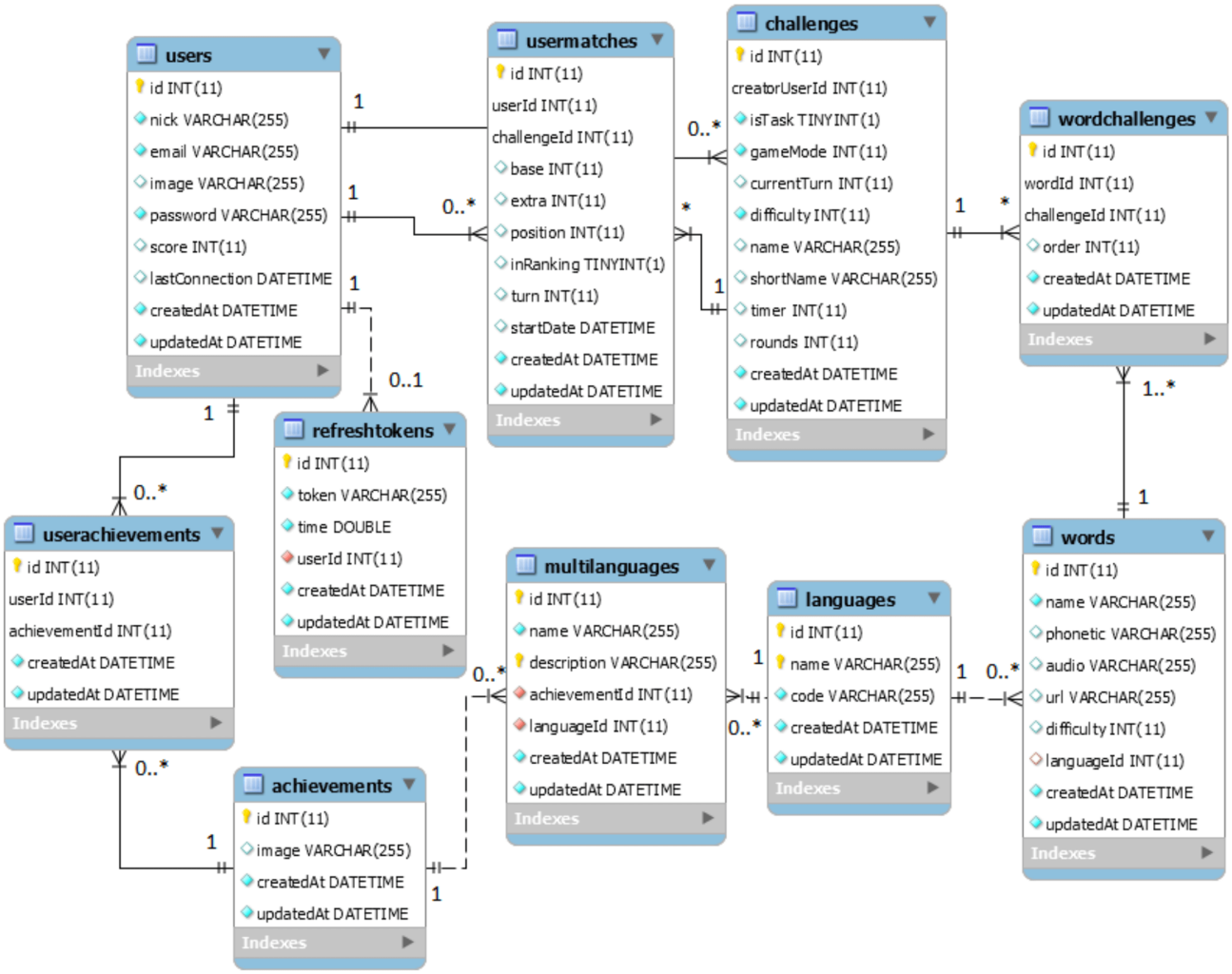


Figura 4.25: Diagrama relacional de la base de datos. Fuente [6]

Como podemos ver en la Figura 4.25 los usuarios se almacenan en la tabla Users. Cada usuario tiene relación con la tabla RefreshTokens y con la tabla UserAchievements, que contiene los logros de cada usuario. Cuando un usuario crea un nuevo challenge se crea en la tabla UserMatches la asociación entre ese challenge y cada usuario y en ella se almacenan los resultados una vez el usuario haya completado su turno. Cada challenge además tiene relación con la tabla WordChallenges, que es una tabla que contiene las palabras de cada challenge y su orden. WordChallenges a su vez tiene relación con Words, donde se almacenan todos los datos de las palabras y que a su vez tiene un lenguaje asignado (tabla Languages). Por último, la tabla MultiLanguages es una tabla que modela la información del lenguaje de un logro.

Es importante resaltar que la mayoría de tablas puede permanecer vacías en el arranque de la aplicación a excepción de las tablas MultiLanguages, Languages y Achievements.

4.3.5. Diagrama de despliegue

El último diagrama de diseño es el diagrama de despliegue. En él podemos ver como un cliente desde un dispositivo Android puede establecer una conexión http con el servidor de ECA-SIMM mediante los endpoints de la API y recuperar o mandar información. En la Figura 4.26 podemos ver como es el flujo de la información y sobre qué se está corriendo la API. En este caso podemos ver como Nginx y PM2 se encargan de mantener la API activa y recibiendo los paquetes que lleguen por la red. Además, podemos ver como la API puede recuperar o almacenar información en una base de datos que también se encuentra en la misma máquina virtual. Los datos recopilados (ficheros de log y de audio) en CoP se almacenan en un servidor Apache. También podemos apreciar como el cliente (AndroidDevice) entabla comunicación con las APIs de Google de TTS y ASR.

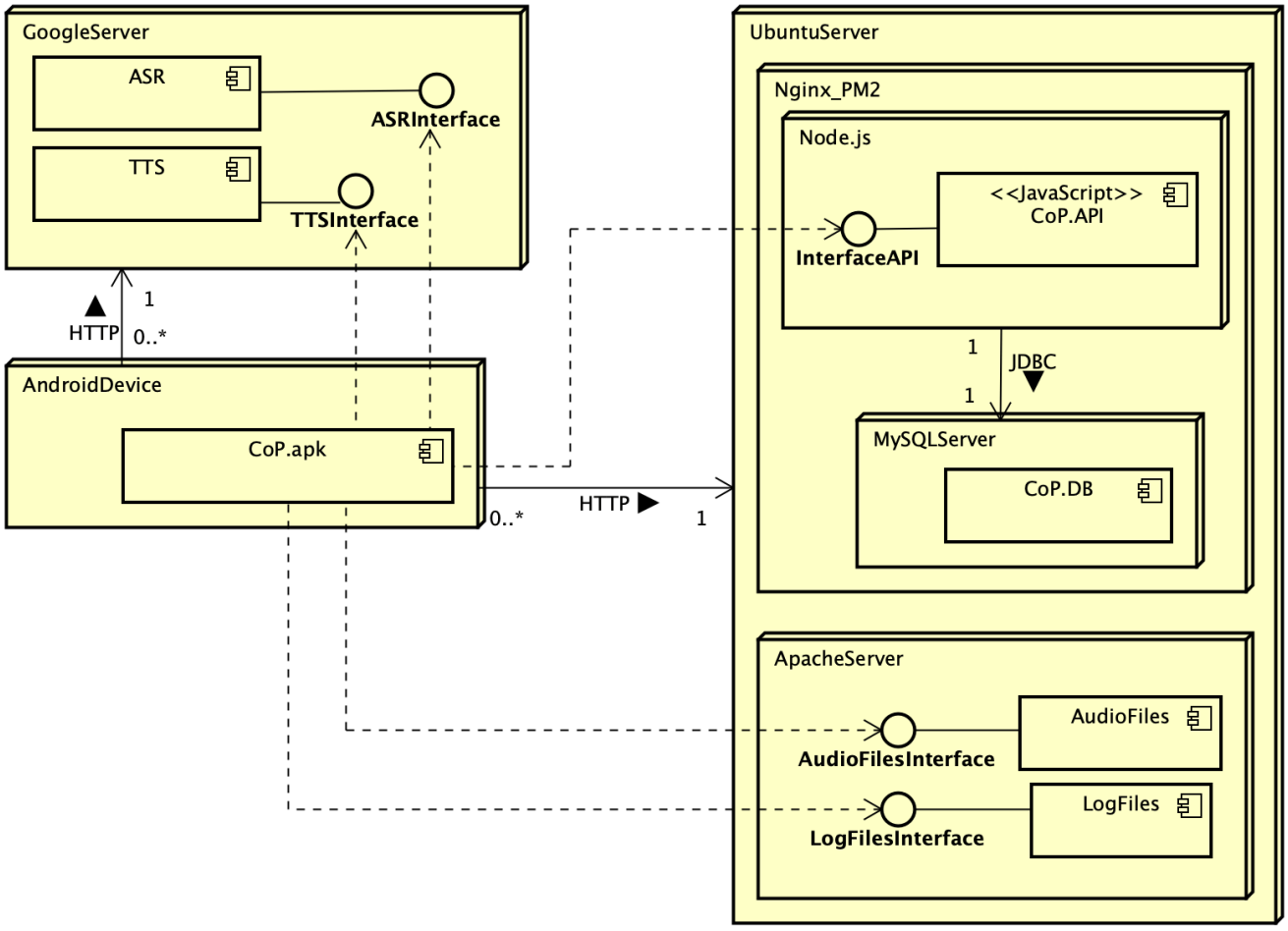


Figura 4.26: Diagrama de despliegue

4.4. Métricas del trabajo realizado

En este apartado se mostrará las tablas con todos los cambios ejecutados tanto en CoP (ver Tabla 4.14) como en la API (ver Tabla 4.15). Como puede verse en ambas el trabajo mayoritario ha recaído sobre la aplicación Android (CoP). Esto es evidente pero es necesario resaltar el trabajo realizado en

la API también. Como se ha ido comentando a los largo de las iteraciones, los cambios en la API eran mínimos pero no triviales y surgían como consecuencia de bugs o la necesidad de añadir campos extra en las respuestas. Además, en el servidor se han añadido todos los ficheros públicos que antes se almacenaban en un servidor propio de ECA-SIMM. Esta es la razón del elevado número de ficheros añadidos.

Desarrollador	Ficheros añadidos	Líneas modificadas o añadidas	Líneas borradas	Total de líneas	Commits
TipTopTalk! [1]	358	30.354	0	30.354	303
Rafael [9]	31	9.356	2.534	40.240	22
Tutores [5]	31	24.939	20.356	44.823	24
Juan	**45**	**18.707**	**15.823**	**47.204**	**157**
Total	465	83.356	38.713	47.204	506

Tabla 4.14: Evolución del código fuente de la aplicación cliente

Desarrollador	Ficheros añadidos	Líneas modificadas o añadidas	Líneas borradas	Total de líneas	Commits
Andrés [6]	43	20.273	0	41.158	35
Juan	**16**	**1.195**	**1.020**	**41.333**	**32**
Total	59	21.465	1.020	41.333	67

Tabla 4.15: Evolución del código fuente de la API del servidor

Estos datos se han obtenido mediante comandos git. El dato de los ficheros añadidos por cada usuario se ha obtenido con el comando:

```
git ls-tree --name-only -r <<hash\_del\_commit>> > commit.txt
```

En el campo de hash se ha puesto el hash correspondiente al primer y último commit del desarrollador y se han restados los ficheros finales de los iniciales.

En cuanto al resto de datos se ha obtenido mediante el comando:

```
git log --shortstat --author="<<nombre_desarrollador>>" --no-merges | grep -E "fil(e|es) changed"
    | awk '{files+=$1; inserted+=$4; deleted+=$6; delta+=$4-$6; ratio=deleted/inserted} END {
   printf "Commit stats:\n- Files changed (total)..  %s\n- Lines added (total)....  %s\n- Lines
   deleted (total)..  %s\n- Total lines (delta)....  %s\n- Add./Del. ratio (1:n)..  1 : %s\n",
   files, inserted, deleted, delta, ratio }' -
```

Capítulo 5

Pruebas

Durante todo el desarrollo se han llevado a cabo diferentes pruebas que serán definidas más en profundidad en este capítulo.

5.1. Aplicación de prueba de las llamadas a la API

A lo largo de la segunda iteración se utilizó una aplicación Android que permitiera probar cada una de las llamadas de manera aislada y desde un entorno Android, no solo desde la aplicación de pruebas Postman. De esta manera se podrían comprobar los posibles errores que surgieran por el hecho de añadir el sistema operativo Android. Esto permitió conocer la necesidad de crear hilos para cada llamada, ya que no se pueden ejecutar llamadas bloqueantes en el hilo principal de una aplicación, ya que es el que sostiene la interfaz de esta. La aplicación permitió corregir pequeños fallos que están expuestos a lo largo del trabajo realizado en el Apartado 3. En el Apéndice E se pueden ver todas las pruebas finales que se ejecutaron antes de dar por válida la aplicación y el inicio de la migración de las llamadas en CoP.

5.2. Pruebas unitarias en CoP

Se han añadido algunos test unitarios a la aplicación. Debido a la falta de tiempo y al enorme tamaño de la aplicación se considera imposible cubrir todo el código de la aplicación. Simplemente se han añadido unos pocos ejemplos que sirven para poder comprobar como realizar test unitarios en Android bajo el lenguaje de programación Java y el uso de hilos para ejecutar tareas asíncronas. Según Android developers [63] las pruebas de una aplicación Android deben ser de 3 tipos. El primero de ellos se corresponde con los test unitarios (ver Figura 5.1). Estos se encargan de validar el comportamiento de los métodos de cada clase. A continuación vienen los test de integración. Estos validan el comportamiento de un módulo en su conjunto. Por último tenemos los test de UI (User Interface). Estas pruebas validan el comportamiento de extremo a extremo de varios módulos simulando acciones que un usuario

ejecutaría.

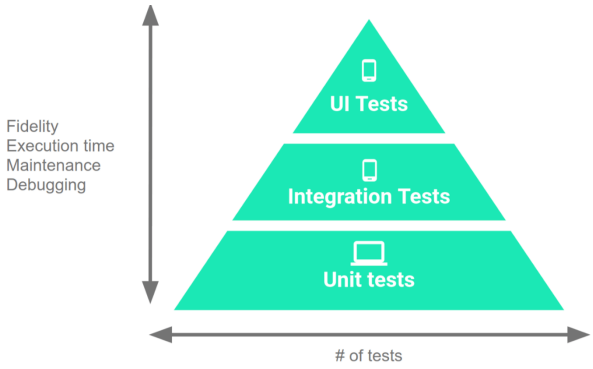


Figura 5.1: Pirámide de pruebas en una aplicación Android. Fuente [63]

Los test unitarios implementados son los correspondientes a las llamadas de login y userData 5.2. Debido a que estas llamadas se ejecutan en hilos independientes de hilo principal los resultados obtenidos no son accesibles desde el hilo de ejecución de los test. Este problema que se solventó en las clases de la aplicación con el uso de la clase LiveData. El problema es que los test solo ejecutan un hilo y no se pueden testar correctamente las llamadas API. Esto se soluciona con el uso del Kotlin y corrutinas, pero no no sirve en nuestro caso, ya que usamos Java. Para poder acceder a los valores de los hilos creados desde el hilo de las pruebas usando LiveData fue necesario el uso del framework Roboelectric [64]. Mediante este framework y su clase Shadows se puede recrear toda la funcionalidad que no existe en los test y obtener las respuestas del servidor. También hemos usado CountDownLatch [65] que permite activar un contador que mantiene el test parado hasta que se de la orden de continuar. La orden la colocamos dentro de las acciones del LiveData y una vez que este es notificado de una cambio se ejecutan los asertos y la orden de continuar el test.

```java
@Test
public void testUserData() throws Exception {
    MutableLiveData<Result> result = new MutableLiveData<>();
    final CountDownLatch latch = new CountDownLatch(1);

    result.observeForever((Observer) (result) -> {
            if (result instanceof Result.Error) {
                fail();
            }
            if (result instanceof Result.Success<?>) {
                String response = ((String) ((Result.Success<?>) result).getData());
                assertNotNull(response);
                JSONObject convertedObject = null;
                try {
                    convertedObject = new JSONObject(response);
                    Assert.assertTrue(Functions.checkResponse((String) convertedObject.get("Status")));
                    String data= "\"Data\": {\n" +
                            "        \"nick\": \"juan\",\n" +
                            "        \"email\": \"juan@mail.com\",\n" +
                            "        \"image\": \"http://157.88.124.247:65232/images/profile/car.jpg\"";

                    Assert.assertEquals(data, convertedObject.get("Data"));

                } catch (JSONException e) {
                    e.printStackTrace();
                }
                latch.countDown();
            }
    });

    CreateCall.userData( result, sharedPrefs);
    Thread.sleep( millis: 2000);
    Shadows.shadowOf(Looper.getMainLooper()).idle();

    if (!latch.await( timeout: 5, TimeUnit.SECONDS)) {
        throw new RuntimeException("LiveData value was never set.");
    }
}
```

Figura 5.2: Ejemplo de test de las llamadas de la API

5.3. Test de usabilidad

Para poder comprobar que la aplicación es fácil de utilizar y sencilla en aprender, se ha definido un test de usabilidad. En él vamos a proponer a dos usuarios reales realizar dos acciones de la aplicación. Una la consideramos más compleja de ejecutar y la otra más sencilla. Durante ese proceso estaremos observando las acciones que realizan estos usuarios y anotaremos en una tabla las métricas utilizadas (ver Tabla 5.1).

Tarea	Tarea 1
¿Completado?	Sí
Valoración	10
Descripción	Descripción de la tarea a realizar
Métricas	▪ tiempo dedicado a realizar la tarea en segundos: ▪ toques para completar la acción (necesarios X): ▪ errores:
Observaciones	Posibles situaciones o comentarios realizados por los usuarios del test

Tabla 5.1: Plantilla utilizada para los test de usabilidad

Los atributos de usabilidad que vamos a medir son la eficiencia, la eficacia y la satisfacción. La eficiencia será correcta si el usuario consigue terminar la tarea sin ayudas, la eficacia si el usuario consigue realizar la tarea sin toques innecesarios y en tiempo moderado y la satisfacción se medirá con las preguntas finales.

Las dos tareas que deben realizar los usuario serán:

- Arrancar la aplicación y registrarse en ella, crear una partida con 4 oponentes y jugarla. Una vez terminada volver al menú principal y acceder a la información de esta.
- Arrancar la aplicación y mostrar el ranking de usuarios. Después mostrar la lista de logros.

Obtenidas las tareas, debemos crear los escenarios que son presentados a los usuarios para que conozcan el contexto de la aplicación y comprendan las tareas que tienen que realizar. Lo escenarios son:

- *Imagina que te has bajado la aplicación y quieres jugar tu primera partida multijugador, por turnos, frente a 4 jugadores más. En primer lugar, deberás registrarte en la aplicación introduciendo tus datos. Tras esto ya podrás acceder a la aplicación y jugar partidas. Crea y juega una partida multijugador con 4 oponentes. Una vez has jugado la partida, quieres ver los resultados que has obtenido tú y ver si algún jugador más ya ha jugado.*

- *Imagina que llevas con la aplicación descargada en tu dispositivo varios días y ya has jugado partidas y conseguido logros. Esta vez quieres acceder a la aplicación y ver el ranking con todos los usuarios. Después quieres ver tu lista de logros.*

Al finalizar las tareas se realizarán las siguientes preguntas a los usuarios con el fin de recuperar la máxima información posible. De esta manera se podrán proponer cambios en la interfaz de usuario para mejor la experiencia y usabilidad de la aplicación. Las preguntas serán:

- *¿La interfaz le ha parecido intuitiva?*
- *¿En algún momento no ha sabido que hacer?*
- *¿Los tamaños de letras e iconos le parecen adecuados?*
- *¿Desea añadir algún comentario más?*

Los usuarios escogidos corresponden a dos grupos muy diferentes, uno de ellos tiene 24 años y una habilidad gran habilidad con la tecnología (en el uso diario de aplicaciones y el móvil) propia de este sector de la población. El otro usuario tiene una edad de 60 años y aunque no parte de cero con la tecnología, su habilidad es mucho más limitada. No consideramos un tercer grupo que no tenga habilidad con la tecnología porque consideramos que la aplicación se oriente a este tipo de personas. Para llegar a proponerse trabajar un idioma con una aplicación móvil y descargarla desde la Play Store es necesario partir de un cierto grado de experiencia.

Antes de comenzar con el test se explicó a los dos usuarios en qué consiste CoP y las acciones que deberían realizar en cada tarea de manera detallada. Hecho esto, se comenzó con el test. En el Apéndice C se muestran las tablas con los datos obtenidos en cada apartado de cada tarea. Como puede verse los resultados muestran que la interfaz es intuitiva y fácil de aprender. A pesar de ser el primer contacto de los usuarios con la aplicación, los resultados obtenidos en pulsaciones de la pantalla son muy similares a toques mínimos y el tiempo de cada subtarea es relativamente bajo. Además, todas las tareas fueron completadas y sin errores. Como se han completado todas las tareas si problemas la eficiencia se considera máxima. Respecto de la eficacia, el tiempo, los toques en pantalla y inexistencia de errores permite afirmar que esta también es muy alta. Por último, la satisfacción de los usuarios, que medimos de acuerdo a las respuestas de las preguntas, podemos afirmar que también es muy alta, y que sería necesario un estudio mayor para conocer si de verdad es necesario añadir nombres en los iconos o crear un tutorial inicial de acuerdo a los comentarios.

5.3.1. Conclusiones obtenidas en los test de usabilidad

Como era de esperar antes de realizar el test, el usuario más joven y con mayor manejo tecnológico realizó la mayoría de acciones en un tiempo más corto y sin errores. A pesar de esto hay que destacar que el usuario menos experimentado tampoco encontró problemas para realizar las tareas de manera correcta.

Al finalizar el test de usabilidad se pueden extraer varias conclusiones de acuerdo a los comentarios y observaciones de los usuarios:

- Al arrancar la aplicación por primera vez el usuario puede estar un poco perdido. Sería conveniente añadir un tutorial inicial mostrando todos los botones de la aplicación y explicando las diferentes modalidades de juegos.
- Es necesario realizar un estudio mayor antes de editar la interfaz.
- Los botones de ranking e historial de partidas no han sido del todo intuitivos en uno de los dos casos. De nuevo, realizar un tutorial inicial solventaría este problema.

Capítulo 6

Conclusiones

En este capítulo se exponen las diferentes ideas que se han ido recopilando a lo largo de la realización del proyecto. Además se arroja luz a posibles implementaciones y mejoras futuras del mismo. Primero, en este TFG se ha modificado la aplicación móvil CoP basada en la tecnología de Android 11 (API 30), Java 8, TTS y STT y que se comunica con un servidor web que basado en Node.js y que utiliza las herramientas Nginx, PM2 y MySQL como base de datos. Gracias a esta migración, se ha conseguido volver a hacer funcional la aplicación que anteriormente usaba la API de Google Play Games y que llevaba casi tres años parada como consecuencia del no mantenimiento de dicha API y del código fuente de Android. Debemos destacar que el código fuente de CoP ha evolucionado a lo largo de cinco años (desde 2016) y está directamente relacionado con otros dos TFGs [9, 6], un TFM [1] y una tesis doctoral [4].

Durante el proceso de migración se han descubierto nuevas herramientas, ya sean Nginx y PM2 para el despliegue del servidor web a lo largo de la primera iteración del proyecto. También el uso de sequelize en la base de datos y el propio entorno de ejecución de JavaScript (Node.js), que ha sido necesario entender para poder corregir los diferentes problemas encontrados. En la segunda y tercera iteración, en las que se han desarrollado las aplicaciones de pruebas de las llamadas y la migración de las llamadas de CoP, se ha podido conocer en profundidad el framework de Android, y se ha comprendido la necesidad de mantener una arquitectura compleja en una aplicación, ya que, como consecuencia de las iteraciones que la aplicación ha ido sufriendo con el paso del tiempo, desde su desarrollo original, varias clases se han convertido en clases poco manejables y demasiado complejas.

Respecto al código de la aplicación Android, aunque final de este capítulo se detallarán posibles mejores y trabajos futuros para la misma, se ha echado en falta una arquitectura inicial bien definida y separar la lógica de las clases que se encargan de las vistas de la interfaz móvil (Activities). Esto no estaba bien definido cuando la aplicación vio la luz hace más de 5 años y se ha quedado un poco atrasado. A pesar de esto, en las nuevas vistas que se han creado sí se ha usado un patrón de presentación MVVM (la recomendación actual de Google para Android) y además, se ha usado un paquete extra dedicado únicamente a las llamadas al servidor y las funciones relacionadas con estas llamadas.

En cuanto a la planificación, se ha podido constatar la dificultad de realizar una buena planificación, a

pesar de estar relativamente cerca en horas reales sobre las planificadas. Además, se ha comprobado la necesidad de una planificación precisa. Partiendo de ella, cumpliendo los objetivos por iteraciones y dejando algunos días de margen, se puede llegar a la fecha de entrega de manera correcta y sin enormes esfuerzo finales. Junto con la planificación, es necesario destacar la metodología utilizada. Gracias al método iterativo se ha podido mostrar el progreso a lo largo de las iteraciones y incluso dentro de ellas. De esta manera los tutores han podido corregir conceptos erróneos o dar nuevas ideas sin tener que tirar todo el trabajo realizado. El hecho de comprobar el progreso en las iteraciones con las fechas y tiempo planificado permite conocer si hay retrasos y cómo de grandes son.

Por otro lado, las pruebas realizadas han permitido obtener retroalimentación sobre el sistema. Los posibles usuarios reales ofrecen sus puntos de vista en los test de usabilidad y se proponen o sugieren mejoras con sus comentarios. En nuestro caso, los usuarios han reportado que un pequeño tutorial introductorio al iniciar la aplicación por primera vez podría ayudar a comprender antes la interfaz y con ello la aplicación en su conjunto. Otro posible cambio, sería añadir nombres a los botones de ranking, historial y logros para facilitar su interpretación. Respecto a los test unitarios, a pesar de que no se ha podido cubrir gran cantidad de código se ofrecen varios ejemplos de cómo poder testar el resto de llamadas a la API de manera automática.

Finalmente, en lo personal, el trabajo ha sido muy interesante, ya que me ha permitido recuperar un juego, que estaba parado y volver a hacerlo funcionar, actualizándolo a la última versión de Android. Además, he sido capaz de planificar un proyecto a cuatro/cinco meses vista y que las estimaciones fueron relativamente realistas, por supuesto, con la ayuda de los tutores. Desde el punto de vista del sistema operativo (Android), el TFG me ha permitido aprender a crear aplicaciones Android con Java, a realizar tareas en segundo plano así como reaccionar a los resultados obtenidos en esas llamadas. Respecto del mantenimiento de aplicaciones, queda como manifiesto la necesidad de actualizar el código fuente de la aplicación más a menudo. Si esto no se hace, aparecen, como fue nuestro caso, un gran cantidad de problemas asociados debido a la eliminación o cambio de muchas bibliotecas. Es necesario resaltar la necesidad de una arquitectura bien definida que permite diferenciar los módulos de la aplicación por capas y extraer la lógica de negocio de toda las vistas. Por último, en cuanto a las tecnologías utilizadas y con mi experiencia en las prácticas de empresa utilizando Kotlin se hace necesario el cambio a este lenguaje para programar. El uso de las ya nombradas corrutinas permiten ejecutar código en segundo plano de una manera mucho más sencilla y testarlo mediante métodos propios de estas. Por esta razón sugeriría una actualización del código Java a Kotlin junto con la definición de una arquitectura mejor definida [58] antes de seguir aumentando las funcionalidades de la aplicación.

6.1. Trabajo futuro

Como ya se ha ido recogiendo a lo largo de proyecto, se han propuesto posibles líneas de trabajo futuro a realizar para mejorar la aplicación. En este último apartado se agrupan todos ellas por orden de prioridad:

- Paginar las llamadas que devuelven listas en la API, para evitar posibles problemas en los dis-

positivos. De esta manera se devolverá como respuesta una lista limitada en tamaño y, en caso de querer recuperarse los siguientes elementos de la lista, será necesario realizar otra nueva consulta desde la aplicación.

- Limitar el número de intentos de acceso por login con una misma IP. El servidor deberá bloquear esa IP para prevenir frente a ataques de denegación de servicios.
- Crear una llamada en la API para cancelar partidas. De esta manera se podrá crear un botón en la información de cada partida para cancelarla si no se desea jugar.
- Añadir nuevos ficheros con palabras a la aplicación, con diferentes idiomas e incrementar el número de palabras por ronda, con el fin de variar la dificultad de las partidas. Actualmente esta dificultad está fijada en media, lo que significa que solo se usan las 3 primeras palabras de la lista obtenida del reconocedor. Variando este número de palabras o el número de intentos podemos crear nuevas dificultades.
- Creación de una comprobación en segundo plano en el servidor que elimine las partidas pendientes tras un periodo de tiempo dado. De esta manera se eliminarán de la base de datos partidas sin terminar.
- Definir y migrar la aplicación a arquitectura definida por capas y con patrón de presentación MVVM.
- A la vez que se ejecuta el punto anterior, sería conveniente migrar la aplicación de Java a Kotlin, ya que Android desde hace varios años se ha convertido en Kotlin-first [18].
- Añadir fecha de nacimiento a rango de edad para poder segmentar a los usuarios.
- Parametrización de subida de ficheros (audio y logs) al servidor deseado.
- Añadir la posibilidad de seguir a usuarios y ver su ranking, su perfil, o sus partidas jugadas. De esta manera se podrá ver la mejora de esos jugadores y así incitar al usuario a mejorar también.
- Recuperar el modo de juego de 1 jugador, modalidad de juego que funcionaba antiguamente, que permite jugar de manera indefinida partidas similares a las de multijugador pero contra la máquina y que también otorga puntos.

Apéndices

Apéndice A

Acrónimos

- **API**: Application Programming Interface.
- **CoP**: Clash of Pronunciations.
- **COVID-19**: Coronavirus-Disease-19.
- **CPU**: Central processing unit.
- **ECA-SIMM**: Grupo de investigación de entornos de computación avanzada y sistemas de interacción multimodal.
- **GB**: Gigabyte.
- **IP**: Internet Protocol.
- **IVA**: Impuesto sobre el Valor Añadido.
- **JSON**: JavaScript Object Notation.
- **JWT**: Json Web Token.
- **MVVM**: Model-View-ViewModel.
- **RAM**: Random access memory.
- **REST**: REpresentational State Transfer.
- **SDK**: Software Development Kit.
- **SSH**: Secure SHell.
- **STT (TTS)**: Speech-to-text.
- **TFG**: Trabajo Fin de Grado.
- **TFM**: Trabajo Fin de Máster.
- **TTS**: Text-to-speech.

Apéndice B

Detalle de la funcionalidad implementada

Durante la tercera iteración se han arreglado los problemas encontrados en el análisis de código de los primeros ficheros de la tabla 3.5. También se realizaron cambios en los ficheros del paquete Permissions asociados a métodos obsoletos en las nuevas versiones. Corregidos estos problemas, se pasó al segundo elemento de la lista de cambios. Aquí fue necesario hacer cambios en la pantalla de carga de la aplicación, que requería ejecutar llamadas a tres servicios de ECA-SIMM y que fueron creados mediante **la biblioteca Thread**, ya que Android no permite sobrecargar el hilo principal (hilo de la interfaz) con llamadas bloqueantes. Esto **permitió corregir los problemas asociados al uso de AsyncTask**, que como ya hemos comentado, quedó obsoleto en la API 30.

A continuación, se comenzó con el **análisis y desarrollo** de la primera llamada al servidor (llamada de **login**). En primer lugar, se llevaron a cabo los trabajos de análisis de la llamada para diferenciar bien los pasos necesarios que debe realizar antes de tratar de pasar a su implementación. Todos los diagramas pueden encontrarse en la sección 4.1.3. Como puede verse, tras el inicio de la aplicación y siempre que el usuario no haya iniciado sesión anteriormente, se le pedirán el nombre y contraseña para crear el *json* que requiere el servidor de acuerdo a la documentación del TFG de Andrés de la Maza Valles [6]. Tras recibir la respuesta del servidor se comprueba que esta no contenga un mensaje de error, en cuyo caso se muestra por pantalla y se vacían los campos de nombre y contraseña del login. En caso de no haber error, se almacena la información del usuario para siguientes inicios, se muestra un mensaje de bienvenida y se comienza a realizar la llamada de userData que se describirá más adelante. Tras esto se presenta la vista del menú de la aplicación.

Para la llamada de login ha sido necesario **crear una pantalla de vista de login**, que podemos ver en la Figura B.1 ya que anteriormente el login estaba integrado con los servicios de Google.

En cuanto a la biblioteca de *json* escogida, en este caso ha sido **Gson** [66], ya que permite crear un string del objeto modelo del usuario que ha iniciado sesión para almacenarlo directamente y, a la hora de recuperarlo, crear el objeto directamente desde el string almacenado.

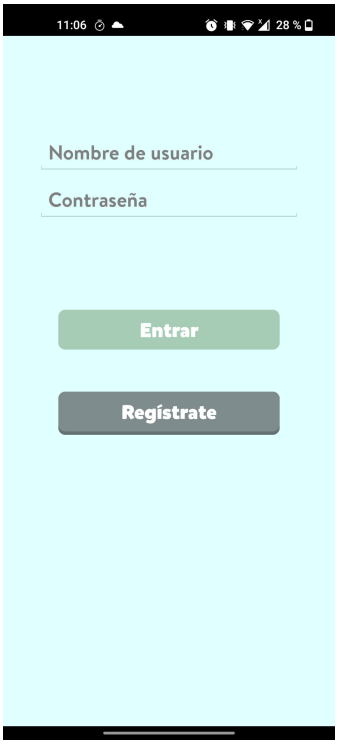


Figura B.1: Vista del login de CoP

Una vez que el usuario ha iniciado sesión en la aplicación, es necesario recuperar otra serie de datos del usuario que se muestran en la vista del menú principal y que solo se pueden obtener si efectuamos la **llamada userData** de la API. Fue necesario llevar a cabo el análisis de la llamada, con lo que se obtuvo el diagrama que podemos ver en la sección 4.1.3.

Tras la sexta reunión con los tutores, **se optó por no almacenar ningún dato del usuario en el almacenamiento** del dispositivo por seguridad y solo mantener el refreshToken. De esta manera, cada vez que se quiere realizar alguna llamada al servidor, es necesario ejecutar dicha llamada para obtener el token y a continuación, usar ese token en la llamada, como puede verse en la sección 4.1.3. Para poder **implementar** completamente **userData**, se creó el diagrama de la llamada **refreshToken**, que podemos ver la sección comentada anteriormente y se pasó a su implementación. Si refreshToken recibe un error porque el refreshToken ya caducó, entonces será necesario ejecutar login de nuevo para poder generar un token, si la llamada es exitosa, entonces se actualizará el token del singleton del usuario que ha iniciado sesión para poder recuperarlo y ejecutar cualquier llamada a continuación.

Creadas las tres primeras llamadas a la API, se pasó a trabajar en uno de los elementos más complejos de la aplicación, el **menú**. En primer lugar, **se corrigieron problemas** asociados a los permisos de acceso al micrófono. Para ello se tuvo que editar el Manifest de la aplicación siguiendo los pasos que Android developers recomiendan [67]. Además de la corrección de los permisos, se eliminó código relacionado con la antigua API de Google Play Games.

Los siguientes pasos realizados fueron la creación de los diagramas de actividad de las llamadas a la API **ranking y register**. Como ya se ha comentado anteriormente, todas las llamadas requieren ejecutar refreshToken a excepción del login, que genera un nuevo token y la llamada register, ya que obviamente solo crea un nuevo usuario en la base de datos para después iniciar sesión con él. La sección

4.1.3 contiene los diagramas creados y usados posteriormente para crear el flujo de las llamadas.

Para poder **implementar** estas llamadas fue necesario, en el caso del **ranking**, editar varias clases ya existentes que usaban la API de Google Play Games así como editar la API para poder devolver en el *json* del ranking las imágenes de cada usuario, ya que estas no estaban incluidas en la respuesta *json* diseñada en el TFG de Andrés de la Maza Valles [6]. Además de esto, se optó por almacenar en la base de datos la URL a las imágenes y no el fichero codificado en base 64 como estaba originalmente creado, por lo que fue necesario editar el fichero del registro. De esta manera el servidor devuelve la URL en la llamada userData y la aplicación se encarga de crear la imagen mediante la biblioteca Picasso [68].

En el caso del **registro, fue necesario crear la vista desde cero**, ya que anteriormente se usaba una vista de la API de Google (ver Figura B.2). En la implementación de la vista del registro se usó el patrón MVVM igual que en el login. Se añadieron una serie de imágenes (sin derechos de autor) en la carpeta public del servidor para que el usuario escogiera entre una serie de imágenes como imagen de perfil. Tuvo que añadirse también un fichero que contiene un array de objetos en formato *json* que contienen la URL de cada imagen. De esta manera se pueden añadir o quitar imágenes de una manera sencilla. Hecho esto se tuvo que crear una llamada desde la aplicación para poder leer el fichero y recuperar las URLs.

Figura B.2: Vista del registro de un usuario de CoP

La siguiente tarea a realizar es la creación de la llamada a la API para obtener la lista de partidas (**listChallenges**). Antes de comenzar con el diagrama se analizó con los tutores lo que debía mostrarse en la vista, ya que de nuevo esta vista se debe crear. Observando el *json* de la respuesta de la API se comprobó que faltaban 2 datos esenciales, por lo que se pasó a editar esta llamada en el código del servidor. Los datos considerados necesarios fueron la fecha de cada partida y los nicks de los jugadores

de la partida. La fecha fue sencillo añadirla ya que existe en la misma tabla de partidas, pero para poder obtener el nick de los usuarios fue necesario editar las consultas a la base de datos de esta llamada. Una vez hecho esto, se comenzó con el diagrama de actividad de la llamada en la aplicación (ver en la sección 4.1.3).

En este caso, **se ejecuta esta llamada pulsando en dos lugares distintos** de la aplicación. El primero de ellos al pulsar **sobre la puntuación y nombre del usuario** y muestra la pantalla de todas las partidas (ver en Figura B.3). El segundo lugar donde se ejecuta la llamada es accediendo a las partidas pendientes mediante el botón de **"Retos pendientes"** del menú principal. En este segundo punto solo se mostrarán las partidas que no han sido completadas.

Figura B.3: Vista de la lista de partidas de un usuario de CoP

Fue necesario crear la vista con el listado de partidas ya que este apartado dependía antes de la antigua API de Google Play Games y no disponíamos de ninguna vista similar. Además de la vista ha sido necesario crear la vista de un item en particular, para poder indicar al listado que ese es el tipo de items que va a almacenar.

Es importante destacar las **combinaciones de colores de la lista de partidas**, ya que, dependiendo de si la partida está terminada, ya ha sido jugada por el usuario o aún está pendiente de jugar, el fondo tendrá un color u otro. El rojo indica que la partida está aún sin jugar por el usuario, el amarillo indica que la partida ya se ha jugado y hay que esperar a que los oponentes la jueguen y, en verde, que la partida ha sido completada por todos los jugadores.

A la hora de recuperar los datos de la lista de partidas se ha comprobado que **el servidor no realiza una paginación de la lista**, por lo que será necesario, para **trabajo futuro**, añadir esa característica. En cuanto a la aplicación y para evitar problemas, se crea una paginación una vez que se ha obtenido

el array de partidas en un JSONArray. De esta manera se evita crear un gran cantidad de objetos y agotar la memoria del dispositivo.

Durante las pruebas de esta llamada **se encontró otro problema en el servidor** que provocaba que no se realizaran las conexiones con la base de datos. El problema surgía porque a medida que la aplicación realizaba llamadas al servidor, este creaba conexiones con la base de datos y en ningún momento se cerraban. Como MySQL tiene un límite de conexiones abiertas, cuando este se alcanzaba no admitía ninguna más y devolvía error en todas las nuevas consultas. Tras encontrar el problema e investigar una solución [69], **se optó por sustituir todas las creaciones de conexiones del servidor por la petición de una conexión a un pool de conexiones** que se configuró en un nuevo fichero de la aplicación de Node. Cuando se requiere ejecutar una conexión se llama al método pool.query que se encarga de pedir una conexión al pool, realizar la consulta y al finalizar devolver esa conexión al pool para que pueda ser utilizada de nuevo.

El **siguiente paso** importante a completar era el de poder **crear un partida y jugarla** de manera correcta por parte del creador de la partida. El trabajo que se llevaría a cabo hasta la siguiente reunión con los tutores trataría de permitir esta funcionalidad básica de la aplicación y la corrección de varias clases obsoletas en la aplicación así como subir los ficheros de audio y el *json* de información al servidor de ECA-SIMM. Lo primero fue realizar el análisis y creación de los diagramas necesarios para poder ejecutar una partida al completo. Para ello fue necesario crear los diagramas de las siguientes llamadas: **opponents, createChallenge, startMatch y finishMatch**. Podemos encontrar estos diagramas en la sección 4.1.3.

Tras el análisis pasamos de nuevo a la implementación y corrección de los problemas que encontramos en las clases que llevaban a cabo esta función. Implementando la llamada de oponentes **nos vimos obligados a editar de nuevo la API**, ya que esta no proporcionaba el ranking de los oponentes en el *json* de la respuesta, por lo que tuvo que crearse una **función que obtuviera dicho ranking** y añadiera la posición en el ranking a cada oponente. Corregido esto se implementó el método que ejecutaba la llamada opponents desde la aplicación reutilizando parte del código existente y se obtuvieron los oponentes sin mayores problemas.

La llamada de crear partida volvió a dar muchos problemas, ya que **el servidor no guardaba ciertas palabras**. Tras una serie de pruebas se pudo comprobar que lo que ocurría es que al haber cambiado la creación de una conexión por palabra e insert en la base de datos, a la petición de una conexión al pool, se generaba un error. El error aparecía al tratar de insertar palabras a la base de datos cuando estas ya existían, en ese momento el método de insertar se cortaba y dejaba de añadir palabras, por lo que las siguientes no se añadían y generan problemas a posteriori. **Se tardó mucho en encontrar el error pero su corrección se completó con tan solo añadir un IGNORE al insert de la consulta**. De esta manera si alguna palabra ya había sido guardada anteriormente no se guardaría de nuevo, pero la consulta continuaría.

De nuevo un **nuevo error** bastante extraño apareció cuando se enviaban ciertas palabras al servidor. En este caso se trataba de **palabras que tenían un apostrofe** como por ejemplo ”She’s”. Lo que sucedía es que la consulta se paraba porque ese mismo carácter se usa como separador de las palabras en las consultas MySQL. La solución pasó por escapar ese carácter. El resto de la llamada no dio

demasiados problemas y se pudo implementar de manera rápida. Cabe destacar que a diferencia de lo que antes ocurría, aquí fue necesario recuperar la lista de palabras por pares y transformarla en dos listas con el tamaños igual al número de rondas en la que cada índice contenía la palabra del mismo par. Esto es necesario ya que el servidor debe almacenar estas palabras para poder mandarlas en la llamada infoChallenge y que los oponentes puedan jugar la misma partida.

Las llamadas de **startMatch y finishMatch no supusieron ningún tipo de problema**, ya que son dos llamadas muy sencillas que solo requieren obtener el id de la partida y, en el caso de finalizar partida, la puntuación base obtenida, por lo que pudieron crearse en poco tiempo ambas.

Los **siguientes pasos** a realizar antes de llegar a la siguiente reunión con los tutores del TFG fueron la corrección de errores en la aplicación así como la **actualización de ciertas clases que se usaban para subir los ficheros de audio de las partidas y los datos** de estas en un *json* al servidor de ECA-SIMM. El problema aparece en Android 11, ya que en la versión más moderna de la actualidad de Android ya no se puede acceder a cualquier ruta escogida de la memoria, porque lo que los ficheros generaban errores al intentar guardarse. Según [70] es necesario pedir a Android la ruta mediante el método getExternalFilesDir(Environment.DIRECTORY_MUSIC) y que este te retorne la ruta en la que alojar los diferentes archivos creados. Tras estos arreglos, al jugar una partida de entrenamiento los logs ya se mandan al servidor de ECA-SIMM y en el caso de los entrenamientos de pronunciación, también los audio grabados durante la práctica.

En la siguiente reunión se planteó la implementación de las cuatro llamadas restantes y la corrección de todas las clases restantes para las siguientes dos semanas. La llamada más complicada durante este periodo sería la llamada de **infoChallenge**, que debe recuperar los datos de una partida concreta y generar la partida tal y como si la hubiera creado el creador mediante createChallenge. Para ello, antes de comenzar con la implementación se crearon, como siempre, los diagramas de cada una de las cuatro llamadas restantes. La sección 4.1.3 contiene los cuatro diagramas correspondientes a las cuatro últimas llamadas al servidor.

A tener en cuenta que, como se puede ver en la Figura 4.8, la llamada se realiza en dos lugares, tanto al pulsar sobre uno de los elementos de la lista de partidas, para mostrar toda la información de esa partida, y desde el final de una partida, para poder recuperar las puntuaciones de todos los usuarios y. mostrarlas antes de volver al menú principal. Fue **necesario crear una vista nueva** para mostrar toda esta información ya que antes las partidas dependían de la antigua API de Google Play Games y no existe tal pantalla (ver Figura B.4).

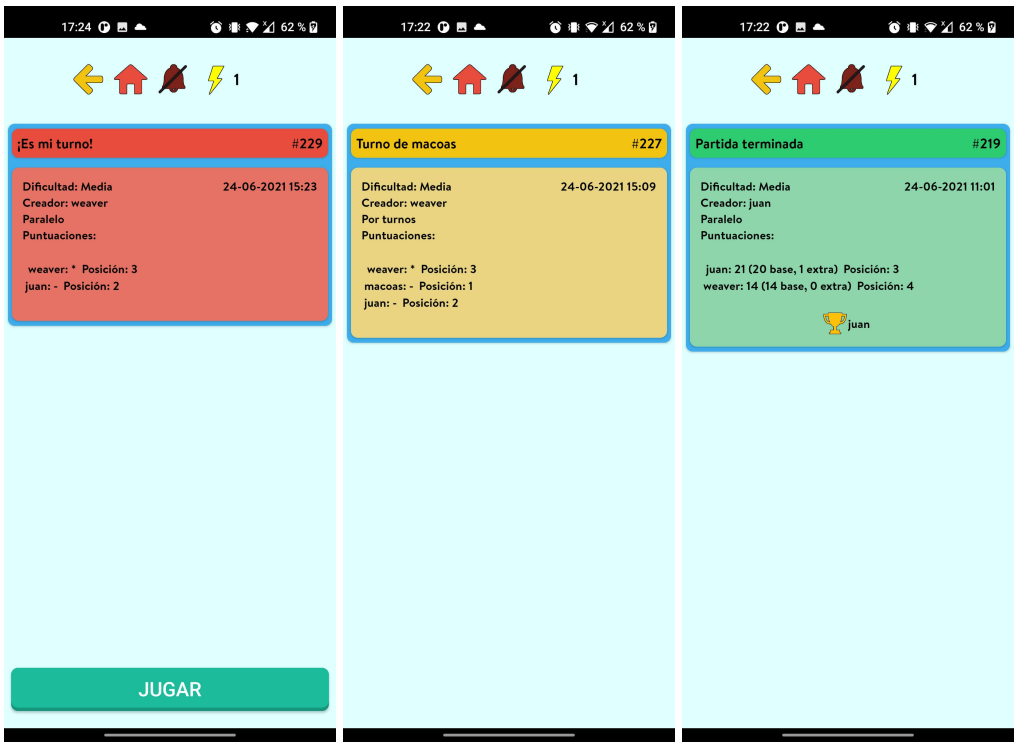


Figura B.4: Vista de la información de una partida de un usuario de CoP. De izquierda a derecha, partida en la que aún no se ha jugado el turno, partida en la que se ha jugado el turno y falta por jugar algún usuario y partida finalizada en la que el usuario ha resultado ganador

Con la llamada de **infoChallenge completa**, se pudieron **probar varias partidas de manera completa**, encontrando de nuevo un **problema** con las palabras que llegaban en junto con el resto de información. El problema era que la palabra ”Stomp” no se guardaba en la base de datos del servidor al crear la partida, y por lo tanto, tampoco se devolvía al pedir la información de la partida. Comprobando los logs y los datos de la base de datos se puede observar que ya existía una palabra guardada que era ”stomp”. La base de datos la consideraba igual y no la guardaba, pero acto seguido se recuperan todas esas palabras de la base de datos y Node no la recupera porque la aplicación web sí distingue entre mayúsculas y minúsculas. Para evitar este y futuros problemas **se decidió enviar todas las palabras desde la aplicación móvil con la primera letra de la palabra en mayúscula y las siguientes en minúscula**.

Respecto del **resto** de llamadas, añadir logro y ver los logros desbloqueados **no supusieron ningún problema**, pero se llegó a la conclusión de que estas comprobaciones **se deberían haber creado en el servidor** y que fuera este el que comprobara si se cumplían las condiciones necesarias para añadir un nuevo logro a un usuario o no. Como esto requeriría editar las llamadas y añadir bastante código se añadirá como trabajo futuro más adelante en este documento. Para comprobarlo se ejecuta en el menú de la aplicación una serie de condiciones y cada vez que se accede a él se realizan una serie de comprobaciones y si se cumplen los requisitos se ejecutan las llamadas para añadir un nuevo logro. El servidor se encarga de añadir ese logro o no dependiendo de si ya se ha añadido anteriormente. En cuanto a **la vista de logros ha sido necesario crearla** (ver Figura B.5 junto con un la vista de un item de logro. Este item se usa como elemento del listado de logros.

Figura B.5: Vista de la lista de logros desbloqueados de un usuario de CoP

Por último, se añadió la llamada **modify**, que permite modificar los datos de un usuario. Para poder acceder a esta nueva pantalla se ha añadido un botón a la foto de perfil del usuario y pulsado se accede a una vista similar a la del registro, que permite reescribir los campos que se quieren editar. Esta vista ha sido reutilizada de la pantalla del registro y se han editado u bloqueado los campos que no pueden ser editados como el nombre del usuarios (ver Figura B.6).

Figura B.6: Vista de modificación de los datos del perfil de un usuario de CoP

Apéndice C

Resultados del test de usabilidad

A continuación, se muestran los resultados obtenidos por los 2 usuarios en en las 2 tareas del test de usabilidad.

C.1. Tarea 1. Usuario 1

En la Tabla C.1, Tabla C.2, Tabla C.3, Tabla C.4, Tabla C.5, Tabla C.6 y Tabla C.7 se muestran los resultados del usuario 1 al realizar la tarea 1 explicada en el apartado anterior.

Subtarea	Registro en la aplicación
¿Completado?	Sí
Valoración	10
Descripción	El usuario debe acceder a la aplicación y registrarse en ella introduciendo sus datos de usuario
Métricas	▪ tiempo dedicado a realizar la Subtarea en segundos: 30s ▪ toques para completar la acción (necesarios 7): 7 ▪ errores: 0
Observaciones	-

Tabla C.1: Tarea 1. Usuario 1

Cuando el usuario terminó la tarea se pasó a realizar las preguntas. Las respuestas obtenidas fueron:

- ¿La interfaz le ha parecido intuitiva? Sí
- ¿En algún momento no ha sabido que hacer? No
- ¿Los tamaños de letras e iconos le parecen adecuados? Sí
- ¿Desea añadir algún comentario más? No

Subtarea	Acceder a la pantalla de oponentes
¿Completado?	Sí
Valoración	10
Descripción	El usuario debe pulsar en el botón Juega para ver la lista de oponentes que escoger
Métricas	■ tiempo dedicado a realizar la Subtarea en segundos: 5s ■ toques para completar la acción (necesarios 1): 1 ■ errores: 0
Observaciones	-

Tabla C.2: Tarea 1. Usuario 1

Subtarea	Seleccionar y aceptar los oponentes seleccionados
¿Completado?	Sí
Valoración	10
Descripción	El usuario debe seleccionar 4 oponentes y aceptar la lista de oponentes para pasar a la siguiente pantalla
Métricas	■ tiempo dedicado a realizar la Subtarea en segundos: 5s ■ toques para completar la acción (necesarios 5): 5 ■ errores: 0
Observaciones	-

Tabla C.3: Tarea 1. Usuario 1

Subtarea	Crear partida por turnos
¿Completado?	Sí
Valoración	10
Descripción	El usuario debe seleccionar una partida por turnos e iniciar la partida
Métricas	■ tiempo dedicado a realizar la Subtarea en segundos: 7s ■ toques para completar la acción (necesarios 2): 2 ■ errores: 0
Observaciones	-

Tabla C.4: Tarea 1. Usuario 1

Subtarea	Jugar partida
¿Completado?	Sí
Valoración	10
Descripción	El usuario debe jugar la partida creada hasta que esta finalice
Métricas	■ tiempo dedicado a realizar la Subtarea en segundos: 125s ■ toques para completar la acción (pueden variar en función de las intentos): 15 ■ errores: 2
Observaciones	-

Tabla C.5: Tarea 1. Usuario 1

Subtarea	Volver al menú
¿Completado?	Sí
Valoración	10
Descripción	El usuario debe volver al menú principal una vez haya finalizado la partida
Métricas	■ tiempo dedicado a realizar la Subtarea en segundos: 3s ■ toques para completar la acción (necesarios 1): 1 ■ errores: 0
Observaciones	-

Tabla C.6: Tarea 1. Usuario 1

Subtarea	Ver resultados
¿Completado?	Sí
Valoración	10
Descripción	El usuario debe acceder a las partidas pendientes y ver la última partida que jugó
Métricas	■ tiempo dedicado a realizar la Subtarea en segundos: 2s ■ toques para completar la acción (necesarios 2): 2 ■ errores: 0
Observaciones	-

Tabla C.7: Tarea 1. Usuario 1

C.2. Tarea 1. Usuario 2

En la Tabla C.8, Tabla C.9, Tabla C.10, Tabla C.11, Tabla C.12, Tabla C.13 y Tabla C.14 se muestran los resultados del usuario 2 al realizar la tarea 1 explicada en el apartado anterior.

Subtarea	Registro en la aplicación
¿Completado?	Sí
Valoración	10
Descripción	El usuario debe acceder a la aplicación y registrarse en ella introduciendo sus datos de usuario
Métricas	■ tiempo dedicado a realizar la Subtarea en segundos: 50s ■ toques para completar la acción (necesarios 7): 9 ■ errores: 0 (los toques extra fueron para borrar datos incorrectos)
Observaciones	-

Tabla C.8: Tarea 1. Usuario 2

Subtarea	Acceder a la pantalla de oponentes
¿Completado?	Sí
Valoración	10
Descripción	El usuario debe pulsar en el botón Juega para ver la lista de oponentes que escoger
Métricas	■ tiempo dedicado a realizar la Subtarea en segundos: 8s ■ toques para completar la acción (necesarios 1): 1 ■ errores: 0
Observaciones	-

Tabla C.9: Tarea 1. Usuario 2

Subtarea	Seleccionar y aceptar los oponentes seleccionados
¿Completado?	Sí
Valoración	10
Descripción	El usuario debe seleccionar 4 oponentes y aceptar la lista de oponentes para pasar a la siguiente pantalla
Métricas	■ tiempo dedicado a realizar la Subtarea en segundos: 9s ■ toques para completar la acción (necesarios 5): 6 ■ errores: 0
Observaciones	-

Tabla C.10: Tarea 1. Usuario 2

Cuando el usuario terminó la tarea se pasó a realizar las preguntas. Las respuestas obtenidas fueron:

Subtarea	Crear partida por turnos
¿Completado?	Sí
Valoración	10
Descripción	El usuario debe seleccionar una partida por turnos e iniciar la partida
Métricas	▪ tiempo dedicado a realizar la Subtarea en segundos: 8s ▪ toques para completar la acción (necesarios 2): 2 ▪ errores: 0
Observaciones	-

Tabla C.11: Tarea 1. Usuario 2

Subtarea	Jugar partida
¿Completado?	Sí
Valoración	10
Descripción	El usuario debe jugar la partida creada hasta que esta finalice
Métricas	▪ tiempo dedicado a realizar la Subtarea en segundos: 189s ▪ toques para completar la acción (pueden variar en función de las intentos): 21 ▪ errores: 3
Observaciones	-

Tabla C.12: Tarea 1. Usuario 2

Subtarea	Volver al menú
¿Completado?	Sí
Valoración	10
Descripción	El usuario debe volver al menú principal una vez haya finalizado la partida
Métricas	▪ tiempo dedicado a realizar la Subtarea en segundos: 5s ▪ toques para completar la acción (necesarios 1): 1 ▪ errores: 0
Observaciones	-

Tabla C.13: Tarea 1. Usuario 2

- ¿La interfaz le ha parecido intuitiva? Sí

Subtarea	Ver resultados
¿Completado?	Sí
Valoración	7
Descripción	El usuario debe acceder a las partidas pendientes y ver la última partida que jugó
Métricas	■ tiempo dedicado a realizar la Subtarea en segundos: 10s ■ toques para completar la acción (necesarios 2): 2 ■ errores: 0
Observaciones	-

Tabla C.14: Tarea 1. Usuario 2

- ¿En algún momento no ha sabido que hacer? No
- ¿Los tamaños de letras e iconos le parecen adecuados? Sí
- ¿Desea añadir algún comentario más? No

C.3. Tarea 2. Usuario 1

En la Tabla C.15 y Tabla C.16 se muestran los resultados del usuario 1 al realizar la tarea 2 explicada en el apartado anterior.

Subtarea	Mostrar ranking
¿Completado?	Sí
Valoración	10
Descripción	El usuario debe arrancar la aplicación (en la que ya inició sesión anteriormente) y acceder al ranking
Métricas	■ tiempo dedicado a realizar la Subtarea en segundos: 8s ■ toques para completar la acción (necesarios 1): 1 ■ errores: 0
Observaciones	-

Tabla C.15: Tarea 2. Usuario 1

Cuando el usuario terminó la tarea se pasó a realizar las preguntas. Las respuestas obtenidas fueron:

- ¿La interfaz le ha parecido intuitiva? Sí
- ¿En algún momento no ha sabido que hacer? No

Subtarea	Mostrar logros
¿Completado?	Sí
Valoración	10
Descripción	El usuario debe volver al menú principal y desde allí acceder a su lista de logros
Métricas	■ tiempo dedicado a realizar la Subtarea en segundos: 4s ■ toques para completar la acción (necesarios 2): 2 ■ errores: 0
Observaciones	-

Tabla C.16: Tarea 2. Usuario 1

- ¿Los tamaños de letras e iconos le parecen adecuados? Sí
- ¿Desea añadir algún comentario más? No

C.4. Tarea 2. Usuario 2

En la Tabla C.17 y Tabla C.18 se muestran los resultados del usuario 2 al realizar la tarea 2 explicada en el apartado anterior.

Subtarea	Mostrar ranking
¿Completado?	Sí
Valoración	9
Descripción	El usuario debe arrancar la aplicación (en la que ya inició sesión anteriormente) y acceder al ranking
Métricas	■ tiempo dedicado a realizar la Subtarea en segundos: 15s ■ toques para completar la acción (necesarios 1): 3 ■ errores: 3
Observaciones	-

Tabla C.17: Tarea 2. Usuario 2

Cuando el usuario terminó la tarea se pasó a realizar las preguntas. Las respuestas obtenidas fueron:

- ¿La interfaz le ha parecido intuitiva? Sí
- ¿En algún momento no ha sabido que hacer? Al intentar acceder al ranking
- ¿Los tamaños de letras e iconos le parecen adecuados? Sí

Subtarea	Mostrar logros
¿Completado?	Sí
Valoración	10
Descripción	El usuario debe volver al menú principal y desde allí acceder a su lista de logros
Métricas	▪ tiempo dedicado a realizar la Subtarea en segundos: 5s ▪ toques para completar la acción (necesarios 2): 2 ▪ errores: 0
Observaciones	-

Tabla C.18: Tarea 2. Usuario 2

- ¿Desea añadir algún comentario más? Se podrían añadir nombres a los iconos para facilitar su entendimiento o mostrar un ejemplo al arrancar la aplicación por primera vez.

Apéndice D

Manual de despliegue

D.1. Despliegue del servidor

A continuación, se muestran todos los pasos para un correcto despliegue del servidor que contiene la API que vamos a usar. Es importante destacar que la instalación se ha llevado sobre un sistema operativo Linux (Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-73-generic x86_64)), por lo que en otro sistema puede haber pequeñas variaciones. Además, el usuario que ha llevado a cabo la instalación tiene el nombre de ”usuario” y tiene privilegios root.

Instalación de git y clonación del repositorio

1. Abrir una terminal en Linux. En estos momentos no importa el directorio en el que nos encontremos.
2. Actualizar cualquier dependencia con ”sudo apt update”;
3. Introducir el siguiente comando para instalar git ”apt-get install git”.
4. Situarse sobre el directorio en el que se desea clonar el proyecto.
5. Ejecutar el siguiente comando ”git clone «gitlab_url_proyecto»”.

Instalación de Node.js y los módulos necesarios

1. Situarse en cualquier directorio, aunque preferiblemente navegar hasta situarse dentro del directorio cop/cop/ del proyecto clonado.
2. Ejecutar el siguiente comando para descargar la versión requerida de node”curl -sL https://deb.nodesource.com/setup_10.x -o nodesource_setup.sh”. Donde ”10.x” se puede sustituir por la versión que deseamos obtener. Las versiones mínimas requeridas (y usadas para desplegar la aplicación) son v10.19.0 para node y 6.14.4 para npm. [25].

3. Ejecutar ”sudo bash nodesource_setup.sh” para agregar la configuración.
4. Ejecutar ”sudo apt install nodejs” para instalar Node.js y npm.
5. Comprobar las versiones instaladas mediante ”node -v” y ”nmp -v”.
6. Ejecutar ”sudo apt install build-essential” para que todos los paquetes npm funcionen correctamente.
7. Si no se ha hecho, situarse dentro del directorio cop/cop/ del proyecto clonado.
8. Instalar los módulos necesarios mediante ”npm i” y ”npm install -g sequelize-cli”.
9. Configurar el proyecto al estado deseado mediante ”export NODE_ENV=development”.

Instalación y configuración de la base de datos

1. Situarse en cualquier directorio.
2. Instalar MySQL mediante el siguiente comando ”sudo apt-get install mysql-server”.
3. Ejecutar ”sudo systemctl start mysql” para arrancar el servidor MySQL.
4. Ejecutar ”mysql” para acceder a la terminal de MySQL.
5. Ejecutar ”ALTER USER ’root’@’localhost’ IDENTIFIED WITH mysql_native_password BY ’1234’;” para crear un usuario ”root”@”localhost” con contraseña ”1234”, que es el usado en la configuración de la aplicación Node.
6. Crear una base de datos mediante el comando ”create database mydb;” cuyo nombre coincide con la configuración de la aplicación Node.
7. Ejecutar ”use mydb;” para configurar nuestra base de datos por defecto.
8. En otra terminal situarse en el directorio cop/cop/ del proyecto y ejecutar "sequelize db:create".
9. Ejecutar a continuación "sequelize db:migrate"para crear todas las tablas en la base de datos creada anteriormente (mydb).
10. Usar el script que permite poblar la base de datos y que se encuentra en la branch juan bajo el directorio cop/cop/developer_help/bbdd y cuyo nombre es insert.sql. Éste script tiene los datos mínimos necesarios para que la aplicación funcione. En el caso de las pruebas, también será necesario usar el script denominado testUsers.sql y que se encuentra en el mismo directorio que el otro. Para insertar los datos necesarios en las tablas se puede copiar todo el contenido de los archivos y pegarlo en la consola de MySQL.

Instalación y configuración Nginx

1. Situarse en cualquier directorio.

2. Ejecutar ”sudo apt-get install ufw” para instalar ufw.
3. Instalar Nginx mediante el siguiente comando ”sudo apt install nginx” [71].
4. Ejecutar ”sudo ufw enable”.
5. Ejecutar ”sudo ufw app list”.
6. Ejecutar ”sudo ufw allow ’Nginx HTTP’ ” para finalmente permitir el acceso a través del firewall al servicio.
7. Ejecutar ”sudo ufw status” para comprobar que el estado es activo.
8. Ejecutar ”systemctl status nginx” y comprobar que nginx.service está activo. Otra manera de comprobar que el servicio está activo es realizar una llamada al servidor mediante ”curl http://ip_del_servidor”, donde ip_del_servidor es la IP pública de nuestro servidor, y ver que aparece la página de bienvenida de Nginx.
9. Ejecutar ”sudo chown -R $USER:$USER /var/www/html”.
10. Editar, si se desea, el archivo html situado en /var/www/html/index.html.
11. Ejecutar ”sudo nano /etc/nginx/sites-available/default”.
12. Sustituir el contenido del fichero abierto por lo siguiente:

```
    server {
        listen 80;
        listen [::]:80;

        root /var/www/html;
        index index.html index.htm index.nginx-debian.html;

        server_name 157.88.124.247:65232 www.157.88.124.247:65232;

        location / {
            try_files $uri $uri/ =404;
        }
    }
```

    El número de puerto debe ser el correspondiente al puerto http, que por norma general es el 80. El server_name es la dirección IP donde se encuentra escuchando Nginx. En nuestro caso es 157.88.124.247:65232, cuyo puerto es 65232 ya que estamos trabajando en una máquina virtual y este es el puerto asociado al puerto 80.

13. Guardar el contenido del fichero anterior y salir.
14. Ejecutar ”sudo ln -s /etc/nginx/sites-available/default /etc/nginx/sites-enabled/”
15. Ejecutar ”sudo nano /etc/nginx/nginx.conf”;
16. Eliminar el comentario de la línea ”server_names_hash_bucket_size 64;”.

17. Comprobar que todo está bien con ”sudo nginx -t”.
18. Reiniciar Nginx con ”sudo systemctl restart nginx”.
19. Ejecutar ”sudo nano /etc/nginx/sites-available/default”.
20. Sustituir el ”location” anterior por:

```
        location / {
            proxy_pass http://localhost:3001;
            proxy_http_version 1.1;
            proxy_set_header Upgrade $http_upgrade;
            proxy_set_header Connection 'upgrade';
            proxy_set_header Host $host;
            proxy_cache_bypass $http_upgrade;
        }
```

Donde 3001 es el puerto en el que escucha la aplicación y que ha sido configurado en cop/cop/-bin/www.

Instalación y configuración de PM2

1. Situarse en el directorio cop/cop del proyecto.
2. Instalar PM2 mediante ”sudo npm install pm2@latest -g”.
3. Iniciar la aplicación mediante ”pm2 start bin/www”.
4. Ejecutar ”pm2 startup systemd”.
5. Usar el último comando que se ha mostrado en pantalla como respuesta a la ejecución del comando anterior. Este debe ser ”sudo env PATH=$PATH:/usr/bin /usr/lib/node_modules/pm2/bin/pm2 startup systemd -u usuario –hp /home/usuario” cambiando ”usuario” por el nombre del usuario propio de cada máquina.
6. Guardar cambios con ”pm2 save”.
7. Iniciar el servicio con ”sudo systemctl start pm2-usuario”.
8. Reiniciar si existe algún fallo con ”sudo reboot”.
9. Comprobar el estado del servicio con ”systemctl status pm2-usuario”.
10. A tener en cuenta. Podemos reiniciar la aplicación con ”pm2 restart app_name_or_id” cada vez que se realice un cambio en sus ficheros.
11. Reiniciar Nginx con ”sudo systemctl restart nginx”.
12. Nuestra aplicación ya se encuentra corriendo.

Para aclarar posibles dudas sobre los diferentes puertos usados cabe destacar que un ordenador recibe peticiones http por el puerto 80. Como nuestra máquina es virtual, su puerto 80 es en realidad el puerto 65232, por lo tanto, si deseamos comunicarnos con ella necesitamos usar el puerto 65232. Una vez Nginx recibe esa información, la mandará a nuestra aplicación, que está escuchando en el puerto 3001 (dicho puerto no es accesible desde fuera).

Una vez está todo configurado, podemos realizar llamadas a la API a través de la aplicación o de programas como Postman. Una posible llamada sería ”http://157.88.124.247:65232/register”.

Instalación y configuración de phpMyAdmin

Para la realización de este manual se ha utilizado la ayuda de [72].

1. Abrir una terminal y usar el siguiente comando "sudo apt update".
2. Instalar phpMyAmdin mediante ”sudo apt install phpmyadmin”.
3. En la instalación no seleccionar ni apache2 ni ligthtpd. Estamos usando Nginx pero no existe ninguna opción por lo que lo dejamos vacío y le damos a ”ok”.
4. En cuanto a la configuración de la base de datos con dbconfig-common diremos "Yes".
5. Por último, solo hay que introducir una contraseña para el usuario phpmyadmin. No debe ser sencilla, ya que esto va a quedar expuesto a la red.
6. Como vamos a seguir usando el usuario root para acceder a la base de datos debemos cambiar su contraseña por otra mediante .ALTER USER ’root’@’localhost’ IDENTIFIED BY ’New-Password-Here’;". A continuación, ejecutar "FLUSH PRIVILEGES;".
7. Editar la contraseña almacenada en el archivo cop/cop/config/config.json para que sea la misma que acabamos de poner para el usuario root de mysql.
8. Ejecutar "sudo vim /etc/nginx/snippets/phpmyadmin.conf".
9. Copiar el siguiente contenido al archivo abierto.

```
        location /phpmyadmin {
            root /usr/share/;
            index index.php index.html index.htm;
            location ~ ^/phpmyadmin/(.+\.php)$ {
                try_files $uri =404;
                root /usr/share/;
                fastcgi_pass unix:/run/php/php7.2-fpm.sock;
                fastcgi_index index.php;
                fastcgi_param SCRIPT_FILENAME $document_root$fastcgi_script_name;
                include /etc/nginx/fastcgi_params;
            }

            location ~* ^/phpmyadmin/(.+\.(jpg|jpeg|gif|css|png|js|ico|html|xml|txt))$ {
                root /usr/share/;
            }
        }
```

10. Salir y guardar el fichero creado.

11. Ejecutar "sudo vim /etc/nginx/sites-available/defaultz copiar la siguiente línea encima del location ïnclude snippets/phpmyadmin.conf;".

D.2. Despliegue del cliente

Antes de instalar la aplicación necesitamos un dispositivo Android con versión Android 8 (API 26) o superior y disponer de acceso a internet. Si cumplimos estos requisitos solo deberemos descargar la aplicación (CoP.apk) a la memoria de nuestro dispositivo. Una vez hecho esto buscaremos en la memoria de nuestro dispositivo (generalmente la aplicación se encontrará en ”descargas”) y pulsaremos sobre nuestra aplicación. Al ser una aplicación externa a la Play Store nos pedirá permisos para poder instalar dichas aplicaciones. Concedemos los permisos e instalamos la aplicación. Hecho esto la aplicación estará lista para iniciarla en nuestro dispositivo.

Apéndice E

Pruebas realizadas en la aplicación de prueba de llamadas

P_I2_1	Comprobar conexión con la API
Precondición	Tener conexión a internet
Descripción	Realizar una llamada cualquiera a la API (register por ejemplo) y obtener algún tipo de resultado de vuelta
Resultado esperado	Tras el registro, se espera un mensaje en formato json desde la API que indique si se ha podido registrar el usuario
Resultado	Correcto

Tabla E.1: Prueba P_I2_1

P_I2_2	Register
Precondición	Tener conexión a internet
Descripción	Registrar un nuevo usuario en la API y comprobar resultado correcto
Resultado esperado	Tras el registro, se espera un json de la API que indica que ha sido correcto
Resultado	Correcto

Tabla E.2: Prueba P_I2_2

P_I2_3	Login
Precondición	Haber resgistrado al usuario anteriormente
Descripción	Iniciar sesión con un usuario registrado y recibir mensaje json de confirmación
Resultado esperado	Tras el login, se espera un json de la API que indica que 'Los tokens han sido actualizados'
Resultado	Correcto

Tabla E.3: Prueba P_I2_3

P_I2_4	RefreshToken
Precondición	Haber iniciado sesión con el usuario
Descripción	Realizar llamada refreshToken para generar token nuevo
Resultado esperado	Tras la llamada, se espera un json de la API que indica que 'Operación realizada con éxito' y que contiene el nuevo Token
Resultado	Correcto

Tabla E.4: Prueba P_I2_4

P_I2_5	ListChallenges
Precondición	Haber iniciado sesión con el usuario
Descripción	Realizar llamada listChallenges para obtener todos los challenges de un usuario
Resultado esperado	Tras la llamada, se espera un json de la API que indica que 'Operación realizada con éxito' y una lista con todos los challenge
Resultado	Correcto

Tabla E.5: Prueba P_I2_5

P_I2_6	UserData
Precondición	Haber iniciado sesión con el usuario
Descripción	Realizar llamada userData para obtener todos los datos del usuario que ha iniciado sesión anteriormente
Resultado esperado	Tras la llamada, se espera un json de la API que indica que 'Operación realizada con éxito' y que contiene la información del usuario
Resultado	Correcto

Tabla E.6: Prueba P_I2_6

P_I2_7	Modify y userData
Precondición	Haber iniciado sesión con el usuario
Descripción	Realizar llamada modify para editar los datos del usuario que ha iniciado sesión anteriormente
Resultado esperado	Tras la llamada, se espera un json de la API que indica que 'Operación realizada con éxito'. Pinchando en la llamda userData podremos ver los datos del usuario tras haber sido editados
Resultado	Correcto

Tabla E.7: Prueba P_I2_7

P_I2_8	Ranking
Precondición	Haber iniciado sesión con el usuario
Descripción	Realizar llamada ranking para ver el ranking de jugadores en función de su puntuación total
Resultado esperado	Tras la llamada, se espera un json de la API que indica que 'Operación realizada con éxito' y una lista con todos los nicks de los jugadores y sus puntuaciones
Resultado	Correcto

Tabla E.8: Prueba P_I2_8

P_I2_9	AddAchievement
Precondición	Haber iniciado sesión con el usuario
Descripción	Realizar llamada addAchievement añadir un logro al usuario
Resultado esperado	Tras la llamada, se espera un json de la API que indica que 'Operación realizada con éxito'
Resultado	Correcto

Tabla E.9: Prueba P_I2_9

P_I2_10	ShowAchievement
Precondición	Haber iniciado sesión con el usuario
Descripción	Realizar llamada showAchievement para ver los logros del usuario
Resultado esperado	Tras la llamada, se espera un json de la API que indica que 'Operación realizada con éxito' y una lista que contiene los logros obtenidos por el usuario
Resultado	Correcto

Tabla E.10: Prueba P_I2_10

P_I2_11	Opponents
Precondición	Haber iniciado sesión con el usuario
Descripción	Realizar llamada opponents para obtener los oponentes necesarios para crear una partida
Resultado esperado	Tras la llamada, se espera un json de la API que indica que 'Operación realizada con éxito' y una lista que contiene oponentes
Resultado	Correcto

Tabla E.11: Prueba P_I2_11

P_I2_12	CreateChallenge
Precondición	Haber buscado oponentes
Descripción	Realizar llamada createChallenge para una nueva partida
Resultado esperado	Tras la llamada, se espera un json de la API que indica que 'Operación realizada con éxito' y un id que contiene el identificador de la partida creada
Resultado	Correcto

Tabla E.12: Prueba P_I2_12

P_I2_13	InfoChallenge
Precondición	Haber creado una partida nueva
Descripción	Realizar llamada infoChallenge para ver los detalles de la última partida creada
Resultado esperado	Tras la llamada, se espera un json de la API que indica que 'Operación realizada con éxito' y toda la información referente a la partida creada anteriormente
Resultado	Correcto

Tabla E.13: Prueba P_I2_13

P_I2_14	StartMatch
Precondición	Haber creado una partida nueva
Descripción	Realizar llamada startMatch para jugar el turno en la partida creada anteriormente
Resultado esperado	Tras la llamada, se espera un json de la API que indica que 'Operación realizada con éxito' y que contiene el campo 'Play': true
Resultado	Correcto

Tabla E.14: Prueba P_I2_14

P_I2_15	FinishMatch
Precondición	Haber jugado el turno de la partida mediante la llamada startMatch
Descripción	Realizar llamada finishMatch para finalizar el turno jugado en la partida creada anteriormente
Resultado esperado	Tras la llamada, se espera un json de la API que indica que 'Operación realizada con éxito' y que contiene el campo 'Play': true junto con el mensaje 'Puntuación base actualizada'
Resultado	Correcto

Tabla E.15: Prueba P_I2_15

P_I2_16	Jugar partida completa
Precondición	Tener conexión a internet
Descripción	Jugar una partida completa mediante la prueba creada en la aplicación
Resultado esperado	Tras la ejecutar la prueba, se espera un json de la API que indica que 'Operación realizada con éxito' y que contiene el campo 'Play': true junto con el mensaje 'Partida finalizada, actualización de la puntuación total'. Comprobar que el ranking ha sido editado y los jugadores han ganado puntos
Resultado	Correcto

Tabla E.16: Prueba P_I2_16

P_I2_17	UserData sin login
Precondición	Tener conexión a internet
Descripción	Ejecutar la llamada userData sin haber iniciado sesión con un usuario
Resultado esperado	Tras la ejecutar la prueba, se espera un json de la API que indica que 'No se ha encontrado ningún token'
Resultado	Correcto

Tabla E.17: Prueba P_I2_17

P_I2_18	Opponents sin login
Precondición	Tener conexión a internet
Descripción	Ejecutar la llamada opponents sin haber iniciado sesión con un usuario
Resultado esperado	Tras la ejecutar la prueba, se espera un json de la API que indica que 'No se ha encontrado ningún token'
Resultado	Correcto

Tabla E.18: Prueba P_I2_18

P_I2_19	CreateChallenge sin login
Precondición	Tener conexión a internet
Descripción	Ejecutar la llamada createChallenge sin haber iniciado sesión con un usuario
Resultado esperado	Tras la ejecutar la prueba, se espera un json de la API que indica que 'No se puede crear partida sin oponentes'
Resultado	Correcto

Tabla E.19: Prueba P_I2_19

Apéndice F

Manual de uso

El primer paso tras iniciar la aplicación es iniciar sesión en la aplicación (ver Figura F.1). Si se tiene una cuenta ya creada solo habrá que añadir el nick y contraseña de usuario. Si no se ha creado aún una cuenta será necesario registrarse. Para ello habrá que pulsar sobre el botón ”Regístrate” e introducir los datos que se nos pide (nick, mail, contraseña y seleccionar una imagen de perfil) (ver Figura F.1). En ambos casos, una vez la sesión se ha iniciado, se nos pedirá que aceptemos los términos de servicio y accederemos al menú. Para poder realizar acciones se nos pedirá que aceptemos los permisos que se requieren (audio y almacenamiento).

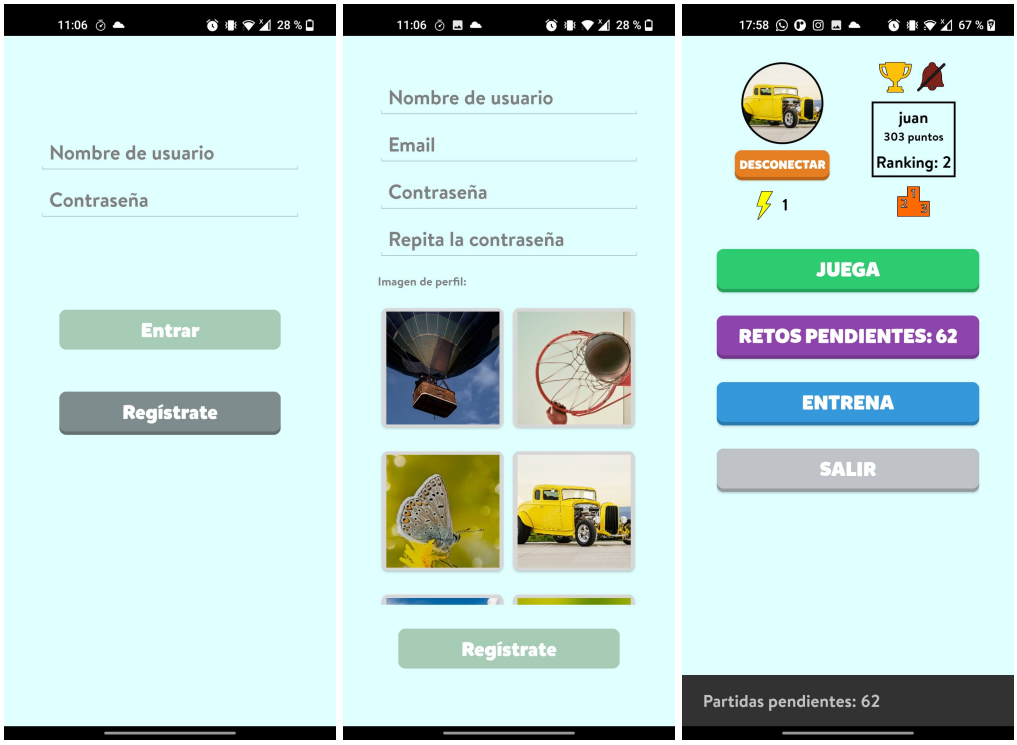


Figura F.1: Login, registro y menú

Una vez en el menú de la aplicación (ver Figura F.1) podemos acceder a la mayoría de su funcionalidad. Podemos modificar el perfil si pulsamos en la imagen de perfil podremos modificar nuestro mail,

contraseño o imagen de perfil (ver Figura F.2). Pulsando en ”Cancelar” o usando el gesto de atrás de Android volveremos al menú.

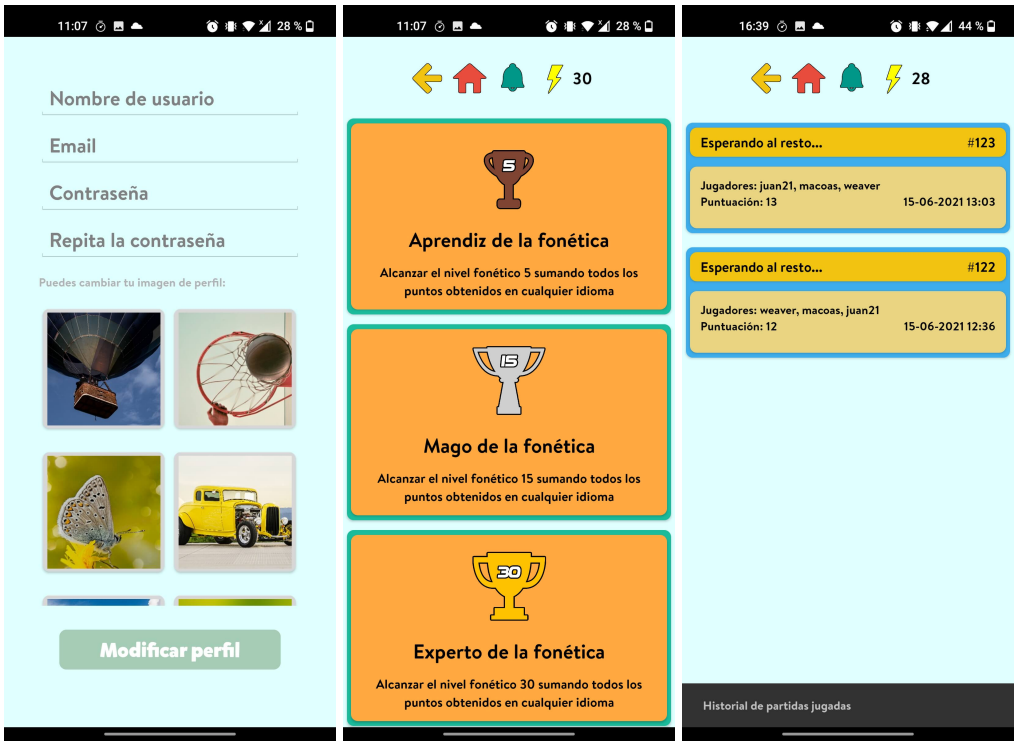


Figura F.2: Modificación del perfil, lista de logros e historial de partidas

Pulsando en ”Desconectar” cerraremos la sesión y se mostrará la pantalla de login de nuevo. Pulsando en la imagen del trofeo accederemos a la lista de nuestros logros desbloqueados (ver Figura F.2). De nuevo, para acceder al menú se podrá pulsar el botón de atrás. Si pulsamos sobre la imagen de la campana podemos silenciar o activar el sonido de la aplicación. En cuanto al historial de partidas, pulsando sobre nuestro nombre y puntuación podremos acceder a él (ver Figura F.2). Se mostrará un lista con todas las partidas que ya hemos jugado, tanto terminadas por todos los participantes como sin terminar por alguno de los oponentes. Podemos navegar en la lista con los botones de ”Recientes” o ”Anteriores”. Podremos pulsar sobre cada partida para obtener más información sobre esta. Para volver hacia atrás solo habrá que pulsar la flecha de atrás o el gesto de atrás de android. Si queremos ver el ranking completo podremos pulsar en el botón del pódium (ver Figura F.3). Aparecerá una lista que nos muestra resaltados en azul y en la que podemos ver la posición y puntuación del resto de jugadores.

El botón ”Juega” permite crear un partida multijugador. Pulsando sobre él se nos pedirá que escojamos de 1 a 4 oponentes para nuestra partida multijugador (ver Figura F.3). Si volvemos a pulsar en ”Crear partida” se nos ofrecerá más información sobre la puntuación y la posibilidad de elegir partida por turnos o en paralelo. Por turnos lo jugadores tienen un turno asignado y no podrán jugar hasta que los turnos anteriores ya hayan terminado su partida. Si pulsamos en ”Comenzar” comenzarán las rondas de la partida. Jugaremos un total de 9 rondas y finalizará la partida (ver sección 1.3.2). Si nos salimos a mitad de partida se nos otorgarán 0 puntos y se terminará nuestro turno. Si completamos la partida podremos ver los datos de las puntuaciones de todos los oponentes y podremos compartir el resultado obtenido con amigos, ver el ranking o volver al menú (ver Figura F.3).

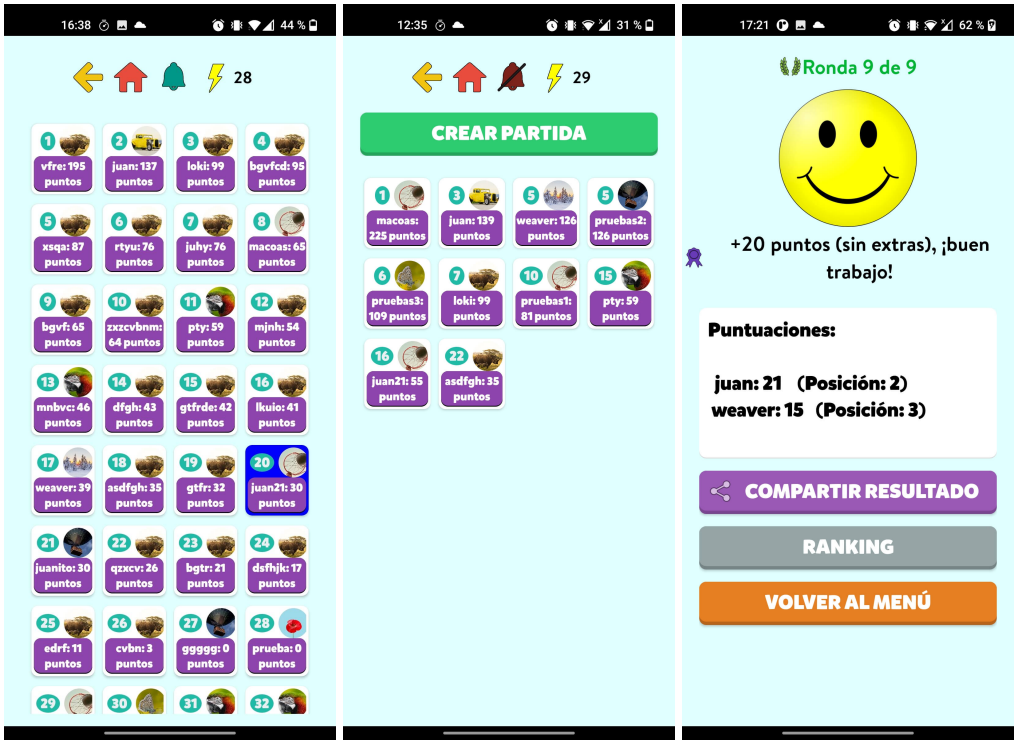


Figura F.3: Ranking, escoger oponentes y vista final de una partida multijugador

Si accedemos a las partidas pendientes podremos ver una lista de partidas similar a la del Historial (ver Figura F.4). A diferencia de este, estas partidas solo muestran partidas sin terminar. Podremos acceder a cada partida en concreto pulsando sobre ella (ver Figura F.4). Si es nuestro turno y no la hemos jugado aún podremos ver un botón de ”Jugar” y pulsando sobre él accederemos a la pantalla de información de la partida. Si pulsamos en ”Comenzar” comenzará la partida igual que si la estuviéramos creando.

Finalmente, pulsando sobre ”Entrena” podremos ver las 3 modalidades de entrenamiento (Exposición, Discriminación y Pronunciación (ver sección 1.3.2). Pulsando en cualquiera de ellas se nos pedirá escoger entre fonemas vocales o consonantes y a continuación se mostrará una lista de diferentes tipos de pares de palabras (ver Figura F.4). Podremos pulsar sobre cualquiera de ellas para realizar el entrenamiento. Al terminar el entrenamiento (ver Figura F.5) podremos acceder a la modalidad de juego que nos recomiende la aplicación, pulsar en ”Continuar” para volver a la lista de palabras, ver el resto de entrenamientos o volver al menú.

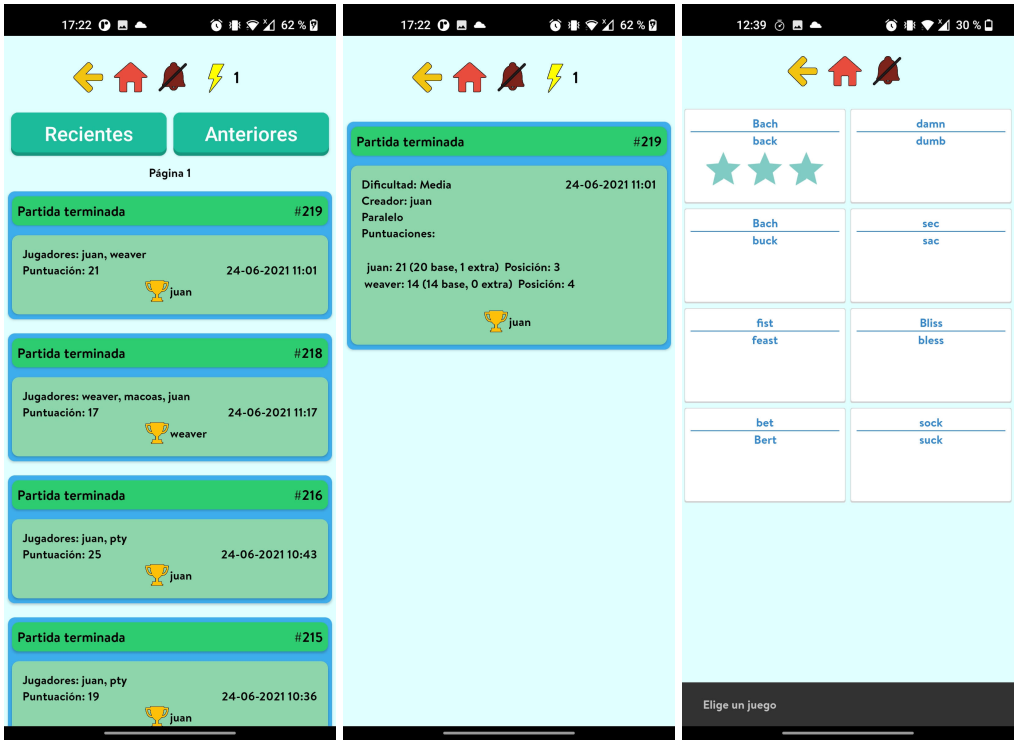


Figura F.4: Lista de partidas, información de una partida y lista de pares de palabras del entrenamiento

Figura F.5: Vista final de un entrenamiento

Apéndice G

Contenido del TFG

Junto con la memoria del TFG, cuyo nombre es **MemoriaTFGJuanGarciaDiguez.pdf** encontramos otro archivo PDF (**DatosTFGJuanGarciaDieguez.pdf**) que contiene una URL a un repositorio del GitLab de la Escuela de Ingeniería Informática de la Universidad de Valladolid. Dentro de este repositorio podemos encontrar:

Directorio raíz del repositorio

CoP.apk: aplicación de Clash of Pronunciations lista para instalar.

mock-up.bmpr: mock-up de la aplicación de pruebas. Formato de Balsamiq Wireframes.

pruebas-cop.apk: aplicación de Android que permite probar las llamadas a la API lista para instalar.

No se incluye el código base ya que está registrado con propiedad intelectual. Para más información contactar con: gir.ecasimm@uva.es.

Bibliografía

[1] C. Tejedor-García, *TipTopTalk! Aplicación móvil para la mejora de pronunciación multilingüe mediante pares mínimos y gamificación*. Universidad de Valladolid, 2016. [Online]. Available: http://uvadoc.uva.es/handle/10324/17469

[2] Google Play Games Services Documentation. Visitado: 2017-06-29. [Online]. Available: https://developers.google.com/games/services/android/quickstart

[3] Clash of pronunciations. Visitado: 2021-06-07. [Online]. Available: https://play.google.com/store/apps/details?id=uva.eca.simm.clashofpronunciation&hl=en_US&gl=US

[4] C. Tejedor-García, "Design and Evaluation of Mobile Computer-Assisted Pronunciation Training Tools for Second Language Learning," Ph.D. dissertation, Departamento de Informática, Universidad de Valladolid, Valladolid, Spain, 9 2020. [Online]. Available: https://doi.org/10.35376/10324/43663

[5] C. Tejedor-García, D. Escudero-Mancebo, V. Cardeñoso-Payo, and C. González-Ferreras, "Using Challenges to Enhance a Learning Game for Pronunciation Training of English as a Second Language," *IEEE Access*, vol. 8, no. 1, pp. 74 250–74 266, 4 2020. [Online]. Available: https://doi.org/10.1109/ACCESS.2020.2988406

[6] A. de la Maza Valles, *API REST para la gestión de partidas multijugador de un juego serio*. Universidad de Valladolid, 2020. [Online]. Available: https://uvadoc.uva.es/handle/10324/44417

[7] Cómo migrar apps a android 11. Visitado: 2021-06-16. [Online]. Available: https://developer.android.com/about/versions/11/migration

[8] Descripción general de las funciones y API. Visitado: 2021-06-16. [Online]. Available: https://developer.android.com/about/versions/11/features

[9] R. S. Navajas, *Desarrollo de la modalidad multijugador para la aplicación Clash of Pronunciations*. Universidad de Valladolid, 2017. [Online]. Available: http://uvadoc.uva.es/handle/10324/27645

[10] Ending support for multiplayer APIs in Play Games Services. Visitado: 2021-06-08. [Online]. Available: https://support.google.com/googleplay/android-developer/answer/9469745?hl=en

[11] Brawl stars. Visitado: 2021-06-19. [Online]. Available: https://play.google.com/store/apps/details?id=com.supercell.brawlstars&hl=es&gl=US

[12] REST API de WordPress: qué es y cómo usarla. Visitado: 2021-06-14. [Online]. Available: https://www.webempresa.com/blog/rest-api-de-wordpress.html

[13] Introduction to json web tokens. Visitado: 2021-06-14. [Online]. Available: https://jwt.io/introduction/

[14] Firebase realtime database. Visitado: 2021-06-15. [Online]. Available: https://firebase.google.com/docs/database?hl=es

[15] Duolingo. Visitado: 2021-06-07. [Online]. Available: https://play.google.com/store/apps/details?id=com.duolingo

[16] Arquitectura de la plataforma. Visitado: 2021-06-07. [Online]. Available: https://developer.android.com/guide/platform

[17] Android e ios dominan el mercado de los smartphones. Visitado: 2021-06-15. [Online]. Available: https://es.statista.com/grafico/18920/cuota-de-mercado-mundial-de-smartphones-por-sistema-operativo/

[18] Enfoque de prioridad de kotlin en android. Visitado: 2021-06-15. [Online]. Available: https://developer.android.com/kotlin/first

[19] Desarrolla apps para android con kotlin. Visitado: 2021-06-15. [Online]. Available: https://developer.android.com/kotlin?hl=es

[20] Preguntas frecuentes de kotlin en android. Visitado: 2021-06-15. [Online]. Available: https://developer.android.com/kotlin/faq?hl=es

[21] Todo lo que necesitas para realizar compilaciones en android. Visitado: 2021-06-16. [Online]. Available: https://developer.android.com/studio/features

[22] Why intellij idea. Visitado: 2021-06-16. [Online]. Available: https://www.jetbrains.com/idea/

[23] Text-to-speech. Visitado: 2021-06-16. [Online]. Available: https://cloud.google.com/text-to-speech/?hl=es

[24] Speech-to-text. Visitado: 2021-06-16. [Online]. Available: https://cloud.google.com/speech-to-text

[25] Cómo instalar node.js en ubuntu 18.04. Visitado: 2021-02-01. [Online]. Available: https://www.digitalocean.com/community/tutorials/como-instalar-node-js-en-ubuntu-18-04-es

[26] Mysql. Visitado: 2021-02-05. [Online]. Available: https://www.mysql.com/downloads/

[27] Postman. Visitado: 2021-02-05. [Online]. Available: https://www.postman.com/downloads/

[28] Visual studio code. Visitado: 2021-06-16. [Online]. Available: https://code.visualstudio.com/?wt.mc_id=DX_841432

[29] Desarrollo iterativo y creciente. Visitado: 2021-06-15. [Online]. Available: http://aprendiendocositasdelsoftware.blogspot.com/p/desarrollo-iterativo-y-creciente.html

[30] Balsamiq wireframes. Visitado: 2021-06-24. [Online]. Available: https://balsamiq.com

[31] Android sdk. Visitado: 2021-02-05. [Online]. Available: https://developer.android.com/studio/index.html

[32] Android studio. Visitado: 2021-02-05. [Online]. Available: https://developer.android.com/studio

[33] Astah. Visitado: 2021-02-05. [Online]. Available: https://astah.net

[34] Excel 365. Visitado: 2021-02-09. [Online]. Available: https://www.microsoft.com/es-es/microsoft-365/excel

[35] Git. Visitado: 2021-02-05. [Online]. Available: https://git-scm.com/

[36] Gitlab. Visitado: 2021-02-05. [Online]. Available: https://gitlab.inf.uva.es

[37] Google drive. Visitado: 2021-02-05. [Online]. Available: https://drive.google.com/

[38] Java. Visitado: 2021-02-05. [Online]. Available: https://www.java.com/es/download/

[39] Nginx. Visitado: 2021-02-05. [Online]. Available: https://www.nginx.com

[40] Node.js. Visitado: 2021-02-05. [Online]. Available: https://nodejs.dev

[41] npm. Visitado: 2021-02-05. [Online]. Available: https://www.npmjs.com

[42] Overleaf. Visitado: 2021-02-05. [Online]. Available: https://www.overleaf.com

[43] phpmyadmin. Visitado: 2021-02-09. [Online]. Available: https://www.phpmyadmin.net

[44] Pm2. Visitado: 2021-02-05. [Online]. Available: https://pm2.keymetrics.io

[45] Safari. Visitado: 2021-02-05. [Online]. Available: https://www.apple.com/es/safari/

[46] Telegram. Visitado: 2021-02-05. [Online]. Available: https://apps.apple.com/es/app/telegram/id747648890?mt=12

[47] Visual studio code. Visitado: 2021-02-05. [Online]. Available: https://code.visualstudio.com/docs/?dv=osx

[48] Cuánto cobra un ingeniero informático. Visitado: 2021-02-09. [Online]. Available: https://universidadeuropea.com/blog/cuanto-gana-un-ingeniero-informatico

[49] Pricing. Visitado: 2021-06-16. [Online]. Available: https://www.digitalocean.com/pricing/#basic-droplets

[50] Precio tarifa internet. Visitado: 2021-02-09. [Online]. Available: https://o2online.es

[51] Precio tarifa luz. Visitado: 2021-02-09. [Online]. Available: https://tarifaluzhora.es/info/calcular-consumo-electrico-casa

[52] Cómo migrar a androidx. Visitado: 2021-02-11. [Online]. Available: https://developer.android.com/jetpack/androidx/migrate

[53] Use legacy support library option in android. Visitado: 2021-02-11. [Online]. Available: https://stackoverflow.com/questions/62325092/use-legacy-support-library-option-in-android

[54] Network security configuration. Visitado: 2021-02-15. [Online]. Available: https://developer.android.com/training/articles/security-config

[55] Asynctask. Visitado: 2021-02-12. [Online]. Available: https://developer.android.com/reference/android/os/AsyncTask

[56] Processes and threads overview. Visitado: 2021-02-12. [Online]. Available: https://developer.android.com/guide/components/processes-and-threads

[57] View. Visitado: 2021-02-18. [Online]. Available: https://developer.android.com/reference/android/view/View#post(java.lang.Runnable)

[58] Guía de arquitectura de apps. Visitado: 2021-06-17. [Online]. Available: https://developer.android.com/jetpack/guide

[59] Descripción general de livedata. Visitado: 2021-06-17. [Online]. Available: https://developer.android.com/topic/libraries/architecture/livedata?hl=es

[60] Observer. Visitado: 2021-06-17. [Online]. Available: https://es.wikipedia.org/wiki/Observer_(patrón_de_dise~no)

[61] Singleton. Visitado: 2021-06-17. [Online]. Available: https://es.wikipedia.org/wiki/Singleton

[62] Facade. Visitado: 2021-06-17. [Online]. Available: https://es.wikipedia.org/wiki/Facade_(patrón_de_dise~no)

[63] Aspectos básicos de las pruebas. Visitado: 2021-06-18. [Online]. Available: https://developer.android.com/training/testing/fundamentals

[64] Roboelectric. Visitado: 2021-06-18. [Online]. Available: http://robolectric.org

[65] Android unit test (11): How to test asynchronous code. Visitado: 2021-06-18. [Online]. Available: https://www.programmersought.com/article/6885635089/

[66] Module com.google.gson. Visitado: 2021-03-18. [Online]. Available: https://www.javadoc.io/doc/com.google.code.gson/gson/latest/com.google.gson/module-summary.html

[67] Cómo administrar la visibilidad de un paquete. Visitado: 2021-04-05. [Online]. Available: https://developer.android.com/training/basics/intents/package-visibility

[68] Picasso. Visitado: 2021-06-17. [Online]. Available: https://github.com/square/picasso

[69] How to make node.js mysql connection pooling available on startup. Visitado: 2021-05-31. [Online]. Available: https://stackoverflow.com/questions/44230641/how-to-make-node-js-mysql-connection-pooling-available-on-startup/44230842

[70] Actualizaciones de almacenamiento en android 11. Visitado: 2021-05-31. [Online]. Available: https://developer.android.com/about/versions/11/privacy/storage?hl=es-419

[71] How to install nginx on ubuntu 20.04. Visitado: 2021-02-01. [Online]. Available: https://www.digitalocean.com/community/tutorials/how-to-install-nginx-on-ubuntu-20-04

[72] How to install phpmyadmin with nginx on ubuntu 18.04. Visitado: 2021-02-12. [Online]. Available: https://linuxize.com/post/how-to-install-phpmyadmin-with-nginx-on-ubuntu-18-04/

[73] Scrum methodology. Visitado: 2017-06-22. [Online]. Available: http://scrummethodology.com/

[74] Gamification. Visitado: 2017-06-08. [Online]. Available: http://www.gamasutra.com/blogs/DanielCook/20140104/208021/What_Ive_learned_about_designing_multiplayer_games_so_far.php

[75] Ciclo de vida ágil. Visitado: 2017-06-22. [Online]. Available: http://www.umsl.edu/~sauterv/analysis/6840_f09_papers/Nat/Agile.html

[76] Scrum en el desarrollo de software. Visitado: 2017-06-23. [Online]. Available: https://en.wikipedia.org/wiki/Scrum_(software_development)

[77] User story characteristics in agile scrum methodology. Visitado: 2017-06-28. [Online]. Available: http://www.yodiz.com/blog/user-story-characteristics-in-agile-scrum-methodology/

[78] INVEST in Good Stories, and SMART Tasks. Visitado: 2017-06-28. [Online]. Available: http://xp123.com/articles/invest-in-good-stories-and-smart-tasks/

[79] Academia de idiomas en valladolid idiomapps. Visitado: 2017-07-06. [Online]. Available: http://idiomapps.es/

[80] Fundación general universidad de valladolid. Visitado: 2017-07-06. [Online]. Available: http://funge.uva.es/

[81] Informe mobile en españa y en el mundo 2016. Visitado: 2017-07-06. [Online]. Available: http://www.amic.media/media/files/file_352_1050.pdf

[82] 6 elements that make a good multiplayer game. Visitado: 2017-07-07. [Online]. Available: https://www.nyfa.edu/student-resources/6-elements-make-good-multiplayer-game/

[83] Black box. Visitado: 2017-07-11. [Online]. Available: https://en.wikipedia.org/wiki/Black_box

[84] What is verification and validation? Visitado: 2017-07-11. [Online]. Available: http://www.softwaretestinghelp.com/what-is-verification-and-validation/

[85] Turn-based multiplayer. Visitado: 2017-07-11. [Online]. Available: https://developers.google.com/games/services/common/concepts/turnbasedMultiplayer

[86] What are serious games? Visitado: 2017-07-11. [Online]. Available: http://www.growthengineering.co.uk/what-are-serious-games/

[87] What is monkey testing in software testing? Visitado: 2017-07-11. [Online]. Available: http://www.softwaretestinghelp.com/what-is-monkey-testing-in-software-testing/

[88] Get the current language in device. Visitado: 2021-04-05. [Online]. Available: https://stackoverflow.com/questions/4212320/get-the-current-language-in-device