Universidad de Valladolid

Escuela de Ingeniería Informática

Trabajo de fin de grado

Grado en Ingeniería Informática
Mención en Ingeniería del Software

API REST para la gestión de partidas multijugador de un juego serio

Autor:
Andrés de la Maza Valles

Universidad de Valladolid

Escuela de Ingeniería Informática

Trabajo de fin de grado

Grado en Ingeniería Informática
Mención en Ingeniería del Software

API REST para la gestión de partidas multijugador de un juego serio

Autor:
Andrés de la Maza Valles

Tutores:
Cristian Tejedor García
Mario Corrales Astorgano

Agradecimientos

Este proyecto ha salido hacia adelante gracias al gran apoyo por parte de numerosas personas. En primer lugar, quiero agradecer a mis tutores Cristian Tejedor García y a Mario Corrales Astorgano por su gran ayuda y esfuerzo depositado en todas las revisiones y correcciones del proyecto que lo han llevado por el buen camino en todo momento.

En segundo lugar, quiero agradecer a mi madre Ana María, a mi padre Andrés, a mi hermano Pablo y a toda mi familia que siempre ha estado ahí apoyándome y animándome a seguir siempre hacia adelante.

Y en último lugar quiero agradecer también a mis amigos de la universidad, los cuales me han acompañado durante toda la carrera, especialmente a Patricia y Silvia que han estado siempre ahí en los buenos y en los malos momentos.

Resumen

En la actualidad, la revolución tecnológica ha provocado un gran cambio en la sociedad en la que vivimos, como por ejemplo la generalización y uso de dispositivos móviles por la gran mayoría de las personas. Con el tiempo, ha permitido que el mundo esté más conectado que nunca y eso ha conllevado a la importancia de aprender nuevos idiomas. A la vez, esta facilidad de acceso a los dispositivos también ha facilitado un aumento en la popularidad de los juegos para esta plataforma, haciendo posible a su vez la competición entre sus jugadores. Este Trabajo de Fin de Grado pretende dar soporte en la parte de *backend* del proyecto existente CoP (Clash of Pronunciation), un juego multijugador multiplataforma en el que los jugadores se enfrentan entre ellos al mismo tiempo que refuerzan su aprendizaje en un idioma distinto al suyo. En concreto, proporciona los servicios para el juego necesarios para poder administrar el modo multijugador y poder gestionar las partidas, así como que los usuarios tengan su propia cuenta y sistema de logros. Este servicio antes era proveído por la plataforma Google Play Games, pero desde el cierre de sus servidores para tareas multijugador, el proyecto CoP quedó sin su principal funcionalidad, por lo que este proyecto trata de solucionar este problema y diseñar una plataforma extensible para futuras funcionalidades.

Abstract

Nowadays, the current technological revolution has greatly changed the society since it has altered the way we live, work and study by means of smart devices. This has allowed the world to be more connected than ever and that means that learning different languages is very important. At the same time, this ease of access to smart devices has led an increase of videogame popularity, and therefore, competitions among players. This final degree project tries to support the backend part of the existing project CoP (Clash of Pronunciation). CoP is a multiplayer and multiplatform game in which users play against each other while they improve their second language proficiency level. In particular, this project provides the necessary services to maintain the multiplayer mode and to manage the challenges, personal accounts and achievements information. These services were provided by Google Play Games until the company shut down them in 2019. This is the main problem this project addresses with an extensible platform for future functionalities.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

1. Introducción

1.1. Motivación

Este proyecto parte de un trabajo previo realizado por el alumno Rafael Sillero Navajas [1] para la aplicación CoP (Clash of Pronunciation). CoP es un juego multijugador que se enfoca en la ayuda y el aprendizaje de la lengua inglesa, adaptable a cualquier otro idioma. Su principal motivación era el gran aumento de tiempo que pasaba la población frente a su teléfono móvil principalmente en ocio y entretenimiento, por lo que se pensó en combinarlo con el aprendizaje de idiomas. Para esto se decidió utilizar elementos de juego (*gamificación* en inglés) como tablas de puntuación o logros a fin de que el usuario se sintiera inmerso en un juego, a la vez que utiliza ese tiempo en aprender una materia nueva. Con esto en mente, se buscaba enfatizar el aspecto social de la aplicación, permitiendo la interacción entre los jugadores mediante la competición a la vez que se fomenta el aprendizaje. Asimismo, se pudo realizar una campaña experimental durante un mes utilizando CoP [2].

Aproximadamente dos años y medio después del proyecto de Rafael se llevase a cabo surgió un grave inconveniente que provocó la creación de un segundo proyecto a partir del primero. Esto fue debido a que Google Play Games anunció el cierre definitivo de sus APIs multijugador por lo que se dejarían de dar ese tipo de servicios tanto para juego en tiempo real como juego por turnos [3]. Por lo tanto, el cierre de estas APIs conllevó a que no se pudiera jugar en la aplicación y fuese necesario la migración de la aplicación a otra plataforma que pudiera mantener estos servicios. La solución que se ha considerado y que ha dado lugar a este proyecto es el desarrollar una API propia que pueda conseguir una funcionalidad lo más similar posible a la que ofrecia Google Play Games y de esta manera configurarla a gusto del cliente y no ajustarse a lo que hacía la plataforma anterior.

Otro problema que surgió de que el proyecto estuviera alojado en Google Play Games era que se dependía de tener una cuenta de Google para poder jugar. Si bien en Android esto es muy generalizado, en otras plataformas como iOS puede que no se disponga de una, lo cual podría disuadir al usuario de dejarlo de lado. El crear una API propia también terminaría con este problema, ya que es una solución multiplataforma en la que no se necesita una cuenta específica, sino que con un correo electrónico cualquiera se podría jugar.

La principal diferencia entre el proyecto de Rafael y este es que el suyo se centra exclusivamente en la parte del usuario y en el funcionamiento del juego, utilizando los servicios de Google Play Games que realizaba las operaciones pertinentes y devolver una respuesta con ello. Sin embargo, este proyecto se centra exclusivamente en la parte del servidor que reemplaza a Google Play Games creando una API propia con el fin de poder proveer los servicios multijugador necesarios para la aplicación CoP, independientemente de la plataforma donde se esté ejecutando. Este proyecto se diferencia también del anterior en que se gestiona sin restricciones externas los elementos del sistema

multijugador, como por ejemplo la gestión de los usuarios, la gestión de logros o creación de las partidas que se efectuarán en el lado del cliente.

1.2. Objetivos

El objetivo general de este proyecto es el desarrollo y utilización de un servicio REST API para gestionar un juego multijugador orientado a la mejora de la pronunciación de un idioma. La principal meta de este proyecto es crear un servidor propio para la aplicación CoP, que permita 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.

El objetivo general puede dividirse en varios objetivos específicos:

- **Objetivo 1:** Implementación de un sistema de identificación seguro para los usuarios.
- **Objetivo 2:** Desarrollo de un sistema de gestión de logros para los usuarios.
- **Objetivo 3:** Creación de un ranking por puntuación, búsqueda de oponentes a partir de esto y creación de partidas según modo de juego, idioma y dificultad.
- **Objetivo 4:** Recopilación de información de los usuarios según sus historiales de partidas.

1.3. Estado de la cuestión

Este proyecto surge a raíz de un trabajo anterior llamado CoP (Clash of Pronunciation) que se trataba de un juego multijugador de teléfono móvil donde el principal objetivo es competir contra otros jugadores a la vez que se refuerza el aprendizaje de un idioma. Por esto, el enfoque de este proyecto se basa en crear un servidor API REST que realice todas que gestione todas las transacciones que necesita el juego, así como a su vez, mantener la comunicación con la base de datos.

1.3.1 Clash of Pronunciation

Clash of Pronunciation (CoP) es un juego en el que se combinan elementos de gamificación con la finalidad de practicar la pronunciación en idiomas extranjeros, llevado a cabo por Rafael Sillero Navajas como trabajo de fin de carrera en la Universidad de Valladolid [1].

El nacimiento de este nuevo proyecto viene dado de que el juego se apoyaba en la plataforma Google Play Games para realizar toda la funcionalidad de back end necesaria para poder jugar como hacer posibles las partidas, un sistema de logros o un ranking. La plataforma dejó de dar servicio a APIs que eran de tipo multijugador, por lo que el proyecto perdió su principal funcionalidad que era poder jugar partidas con otros jugadores por lo que se necesitaba una solución a este problema. Una

de las posibles soluciones era migrar el proyecto a otra plataforma que diese los mismos servicios, pero teniendo que modificar el proyecto para adecuarlo, o crear una propia REST API en el que poder proveer los mismos servicios que antes, pero pudiendo añadir y configurar el servicio de forma independiente y por tanto ganando una mejor adaptación de las funcionalidades que se busca obtener.

El juego utiliza tecnologías como TTS (*Text-To-Speech* en inglés) para que el usuario pueda reconocer la pronunciación correcta de las palabras que se muestran. También utiliza la tecnología ASR (reconocimiento automático del habla en inglés) para que las palabras pronunciadas por el jugador sean interpretadas por el teléfono móvil y comparadas con la versión correcta de la palabra.

Las correcciones de las palabras están basadas en la comparación mediante pares mínimos de los fonemas de dos palabras.

La pantalla inicial del juego muestra el perfil del usuario y sus datos personales además de tres opciones diferenciadas que son la de “Ponerse a prueba” en la que pasará a elegir modo de juego, “Acerca de” en el que se mostrarán los detalles de la aplicación o salir de esta. Este primer menú se puede observar en la Figura 1.

Figura 1. Pantalla principal de Clash of Pronunciation, adaptado de [1]

A continuación de pulsar el botón “Ponte a prueba” saldrá un desplegable con una lista de idiomas para elegir, y una lista de modos los cuales aquí se listan:

- **Exposición:** Dos palabras de un par mínimo de fonemas son reproducidas por el teléfono móvil. De esta manera se puede escuchar un ejemplo de esa pronunciación y que el usuario vaya acostumbrando su oído.

- **Discriminación:** El usuario escucha una palabra aleatoria de cada par mínimo y debe elegir cual ha sido sintetizada por el sistema TTS. En niveles avanzados hay una variación de este modo, en el que se tiene que escoger la que no es correcta.

- **Pronunciación:** Se le presentan al usuario dos palabras de un par mínimo de fonemas a pronunciar, en el que se utiliza la tecnología ASR. El resultado de esto es una lista de hasta cinco resultados según la pronunciación del usuario junto con un indicador de confianza. En función de estos mismos se dará como válida o no la intervención.

- **Modo infinito:** Este modo mezcla las características de los modos Discriminación y Pronunciación y de entre todos los modos, es el que más desafío ofrece. Aparecen elementos como “vidas”, contadores de tiempo, puntos, atajos y rondas. El objetivo es avanzar el máximo número de rondas ganando todos los puntos posibles y la partida solo terminará cuando el usuario se quede sin “vidas” con las que continuar.

1.3.2 API REST

REST es un acrónimo de Representational State Transfer [4] (transferencia de representación de estado en español). Su característica principal es que no tiene estado (*stateless*) lo que significa que entre dos llamadas el servicio pierde todos los datos ya que el estado lo tiene que mantener el cliente, de aquí su nombre. Si se necesita que un servicio REST recuerde cierta información, es necesario identificarse en cada llamada, ya sea por un usuario y contraseña, un token u otro tipo de credenciales.

API es un acrónimo del término inglés Application Programming Interface [5] (en español interfaz de programación de aplicaciones). Su principal característica es que permiten que sus productos se comuniquen con otros sin que estos conozcan su implementación. De esta manera se simplifican los costes y programación y se reducen los tiempos. Su funcionamiento se considera que es parecido a un contrato, ya que, si el emisor envía una solicitud con una estructura conocida, el receptor podrá realizar la operación pertinente de forma adecuada y devolver una respuesta con también un cierto formato establecido.

Un ejemplo de esquema utilizado en las API REST sería el que se muestra en la Figura 2. Como se puede observar, muchos clientes con diferentes plataformas e implementaciones pueden acceder a los servicios del servidor API REST sin darse problemas entre estos.

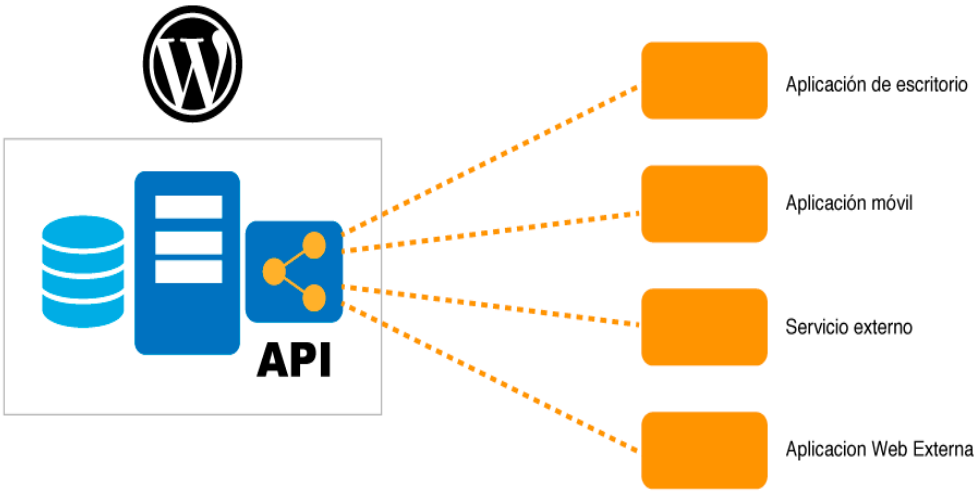

Figura 2. Esquema de ejemplo de una aplicación API REST, adaptado de [6]

Una vez definido los dos conceptos, podemos decir que una API REST es un conjunto de reglas y especificaciones para conectar dos softwares y que permite el intercambio de mensajes entre ellos.

Sus principales características son [7] [8]:

- **Protocolo cliente/servidor sin estado:** Como se ha mencionado antes, no se guarda el estado, por lo que al hacer operaciones HTTP se envía solo la información necesaria, y se relega a elementos como la caché o token si se necesita guardar cierta información.

- **Operaciones HTTP:** Se utiliza la especificación HTTP para realizar las operaciones cuyas cuatro más importantes son POST (crear), GET (leer y consultar), PUT(editar) y DELETE (eliminar)

- **Uso de una URI:** La URI es el identificador único de cada elemento de una API REST. Esto nos facilita acceder a la información que buscamos.

- **Interfaz uniforme**: Para las transferencias se realizan acciones concretas sobre los recursos siempre y cuando estén identificados por una URI. Esto ayuda a sistematizar el proceso de información.

- **Sistema de capas:** arquitectura jerárquica de capas, las cuales cada una realiza una acción.

- **Uso de hipermedios:** El concepto de hipermedia explica la capacidad de una interfaz de desarrollo de aplicaciones de proporcionar al cliente los enlaces adecuados para realizar acciones específicas.

1.3.3 API REST existentes para la gestión de juegos

En esta sección, se describirán brevemente algunos servicios API de juegos:

- **Google Play Games** [9]**:** Es una plataforma de juegos dedicada especialmente para Android (aunque está disponible en iOS y navegadores web) que ofrece a los desarrolladores herramientas para poder añadir funcionalidades a sus juegos como logros, clasificaciones o guardado en la nube entre otras. Actualmente está obsoleto, ya que no ofrece servicios a APIs basadas en multijugador.

- **Google Cloud Open Match** [10]**:** Google y Unity hicieron una colaboración para crear esta plataforma como alternativa al obsoleto Google Play Games y ofrece tres componentes diferentes los cuales son una API de interfaz para el cliente, una API de back-end para el servidor y un orquestador que permite que ejecuta una lógica de emparejamiento personalizada del juego.

- **Firebase RealTime Database** [11]**:** Es una base de datos NoSQL que permite el acceso seguro a la base de datos desde cualquier cliente. Utiliza un lenguaje de reglas basadas en expresiones que define cómo se deben de estructurar los datos y cuando leer o escribir. Esta es una de las alternativas que plantea Google Play Games después de quedar obsoleto.

- **Steamworks** [12]**:** Es una plataforma de juegos dedicada a ofrecer apoyo y distribución de juegos para la biblioteca virtual de Steam. Ofrece diversas herramientas para gestionar todo el proyecto tanto interno como externo, ya que ofrece servicios de distribución y marketing y también funcionalidades para el juego como chats de voz, logros o servidores multijugador.

1.3.4 Entorno y herramientas tecnológicas

1.3.4.1 Node

Node.js [13] [14] es un entorno en tiempo de ejecución multiplataforma desarrollado por Ryan Dahl en 2009, que está basado en ECMAScript y que suele ser utilizado en la capa del servidor, pero no limitado a esta. Fue creado con el objetivo de crear plataformas web de alta escalabilidad ya que otras tecnologías tradicionales sufren cuando se veían a una gran concurrencia de usuarios. Al estar basado en ECMAScript el lenguaje en el que se codifica es JavaScript ya que utiliza el mismo motor de código (V8) que Google Chrome.

Las principales características de Node.js son las siguientes:

- **Entorno asíncrono y controlado por eventos:** Todas las APIs de la biblioteca de Node.js son asíncronas, es decir, no se pueden bloquear debido a operaciones como las de entrada y salida. Un servidor basado en Node.js nunca espera que una API devuelva sus datos, si no que continúa atendiendo otras peticiones y utiliza un mecanismo de notificación de eventos que ayuda a gestionar las peticiones una vez que se han realizado las operaciones pertinentes y la API está preparada para devolver el resultado de la operación.

- **Rapidez:** Al estar construido en el mismo motor de código que Google Chrome, la biblioteca de Node.js es muy rápida en la ejecución del código.

- **Sin almacenamiento en buffers:** Las aplicaciones de Node.js nunca guardarán ningún dato en buffer (característica del REST). Su funcionamiento se basa en generar y devolver datos en fragmentos.

- **Procesos de un solo hilo:** Otras tecnologías tradicionales generaban un hilo del sistema operativo para cada petición recibida. Node.js rompe con este esquema ya que cada conexión ejecuta un evento dentro del entorno que se convierte en un solo hilo del sistema operativo que se repite en bucle. Este ahorro en hilos permite superar la limitación que tienen otros entornos y poder satisfacer un número mayor de peticiones y evitar bloqueos. La Figura 3 muestra un esquema del funcionamiento de Node.js en comparación con otras tecnologías.

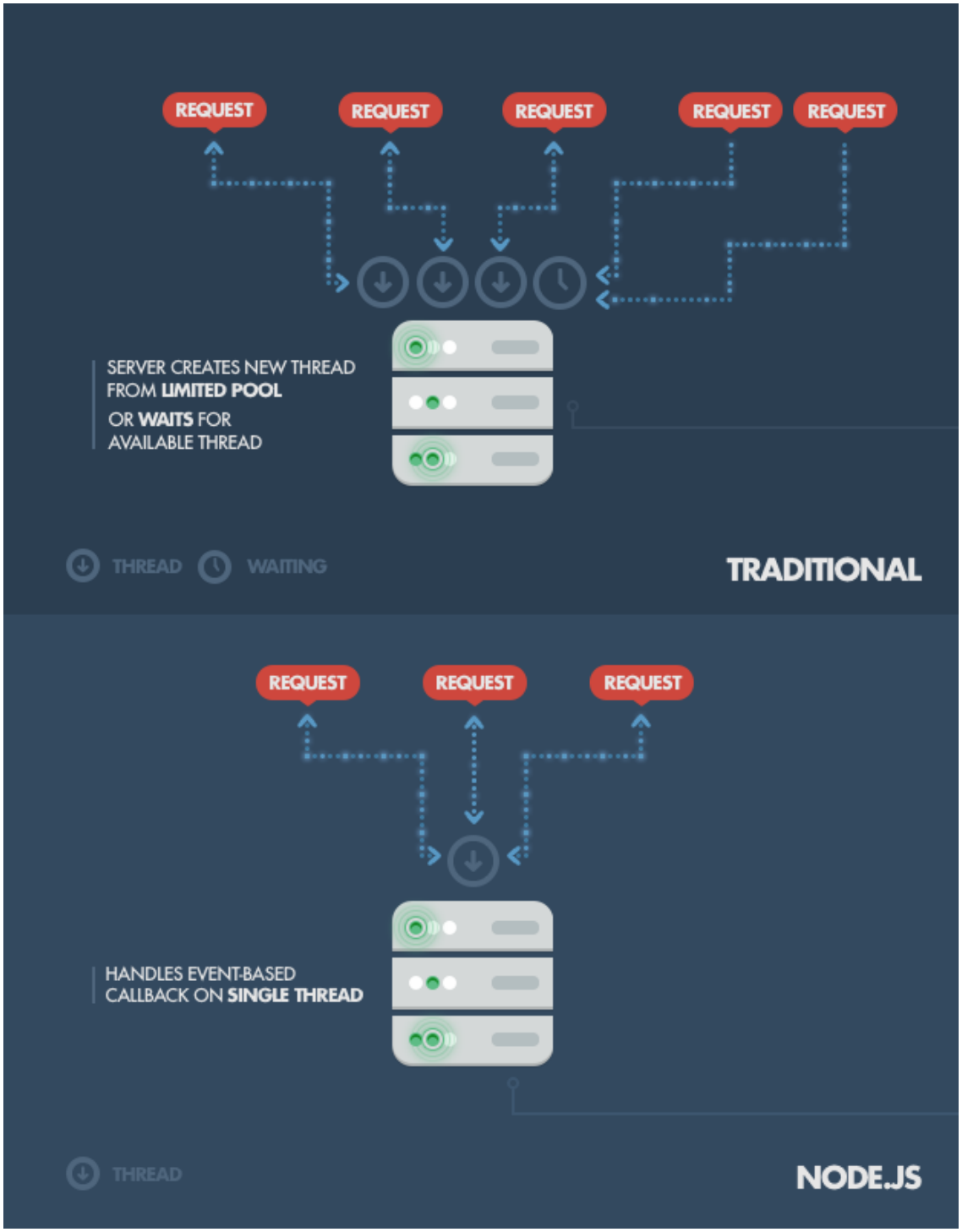

Figura 3. Comparación de Node.js con otras tecnologías, adaptado de [15]

- **Licencia de código abierto:** Node.js es de carácter público y cualquiera puede utilizarlo sin coste alguno. Node.js se distribuye gracias a una licencia MIT.

- **Modular:** Node.js tiene una estructura en la que se pueden cargar y descargar diferentes módulos que amplían o simplifican funcionalidades del entorno base lo que permite una mejor gestión y ahorro de tiempo.

La principal motivación de la utilización de Node.js viene de la necesidad de crear un servicio de API REST que permite gestionar por completo todas las funcionalidades del juego que queremos lanzar. Para poder controlar todo lo necesario se pensó en utilizar la tecnología de Node.js ya que como se indicó en las características del apartado anterior, lo hacen indicado para gestionar numerosas operaciones de diferentes clientes en tiempo real, ya que otras tecnologías, el servidor empezaría a crear hilos a la vez que aumenten los clientes, haciéndolo poco práctico. Node.js combina los requisitos que se buscan para crear un servicio REST API rápido y eficiente.

Node.js [16] se está convirtiendo en una plataforma que cada vez más empresas buscan tener desarrolladores que la controlen. Según la oficina de estadísticas laborales de EEUU se indica que la demanda de desarrolladores web crecerá un 15% entre 2016 y 2026 y si bien este dato no parece ser muy grande, lo que sí se destaca es que, entre estos, la demanda de informáticos que dominen Node.js es mayor a la de otras tecnologías como por ejemplo PHP.

Estos cambios vienen impulsados por cuatro factores clave a nivel empresarial:

- La creciente popularidad de productos en los que Node.js destaca como dispositivos y aplicaciones de IoT con uso intensivo de datos.

- Los beneficios comerciales de utilizar Node.js para los productos mencionados antes (como ahorro de costes y tiempos).

- Comodidad de utilizar el mismo lenguaje de programación tanto para el backend como el frontend.

- La utilización de JavaScript como lenguaje para programar en Node.js. Al ser un lenguaje conocido y enseñado alrededor del mundo otorga una gran ventaja a Node.js

Si bien hay que tener en cuenta de que Node.js no es un lenguaje de programación como PHP, por lo que no es posible realizar comparaciones directas entre estos.

Tanto en el año 2017 como en el año 2018 Node.js apareció en el primer puesto en las encuestas de la comunidad Stack Overflow como la tecnología más utilizada en la categoría de “Frameworks, libraries and tools” [17]. Esto nos indica que además del uso empresarial, Node.js tiene una gran comunidad en la gran extensión que es internet.

Además de esto, existen numerosas empresas que utilizan Node.js cuyos nombres son reconocidos en todo el mundo. Aquí se listan algunas de ellas [18]:

- **Netflix** [19]**:** La gran proveedora de películas y series en streaming utiliza esta tecnología para proporcionar servicios a unos 104 millones de suscriptores. Recurrió a Node.js ya que necesitaba una gran escalabilidad y reducir dependencias.

- **Linkedin** [20]**:** La plataforma dedicada a los negocios y empleo traspasó su backend de aplicaciones móviles de Ruby on Rails a Node.js aumentando su velocidad al doble que su predecesora.

- **Trello** [21]**:** La aplicación de gestión de proyectos buscaba un servidor sin bloqueo controlado por eventos que permitiese tener muchas conexiones activas al mismo tiempo para su propagación instantánea de actualizaciones.

- **PayPal** [22]**:** La plataforma para pagos en línea utiliza Node.js como la plataforma que pudo cohesionar los equipos que se dedicaban al código específico del navegador y los equipos que se dedicaban al desarrollo de la capa de la aplicación.

1.3.4.2 Postman

Postman [23] es una plataforma que sirve de apoyo en el desarrollo de APIs. Su principal función es disponer de una interfaz personalizable al usuario en el que pueda construir numerosas peticiones destinadas a una API lo que permite ahorrar tiempo a la hora de realizar las pruebas pertinentes. Esto se realiza mediante la facilidad de poner una dirección web a la que mandar la petición y la forma sencilla de personalizar el conjunto de datos de entrada que se enviaran. Esta herramienta es totalmente gratuita, pero dispone de programas de pago para empresas en las que se amplía su funcionalidad ejecutando también tests o creación de distintos entornos de trabajo.

En este proyecto se ha utilizado la versión gratuita de Postman y se ha utilizado mediante el programa de escritorio una vez descargado de la página principal.

Figura 4. Esquema de las capas de Postman, adaptado de [23]

1.3.4.3 JWT

Json Web Token [24] es un estándar abierto que define una forma compacta y autónoma 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.

1.4. Metodología

Se ha decidido que la metodología que más se adapta a lo que se busca en este proyecto es el modelo iterativo e incremental de la metodología de la ingeniería, debido a que los requisitos y tecnología son muy sensibles a cambios a corto plazo. El principal objetivo que se busca con esto es poder producir versiones más pequeñas del producto que puedan ir validándose periódicamente, con esto se logran mejoras sucesivas del trabajo realizado para poder presentar un producto final con una mayor calidad. La Figura 5 ilustra resumidamente los pasos que se siguen en la metodología aplicada.

El plan inicial se lleva en conjunto con los tutores del TFG donde se acuerdan los requisitos y la meta de la iteración: Después de esto se elabora la propuesta y una vez que se aprueba, se procede a realizar el análisis y diseño del prototipo para finalizar, se realizan las pruebas pertinentes para comprobar que se cumplen con los objetivos propuestos. Por último, se extraen conclusiones gracias al análisis y se procede a realizar la siguiente iteración del proyecto.

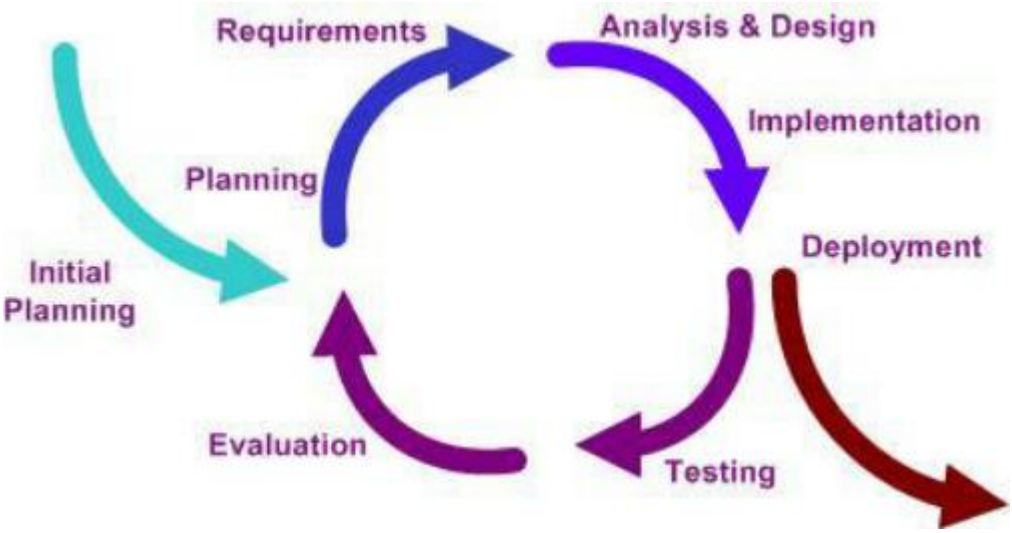

Figura 5. Modelo iterativo e incremental, adaptado de [25]

1.5. Estructura de la memoria

En este documento se detallan todos los aspectos relacionados con el desarrollo teórico y práctico del proyecto. Los bloques en los que se ha dividido este documento son los que aquí se muestran:

- **Introducción:** Resumen con los puntos esenciales del proyecto y donde se explican motivaciones, objetivos y metodología seguida.
- **Planificación:** Estimación y descripción de los planes iniciales y de sus tiempos a la vez que se estiman los costes y riesgos.
- **Descripción de las iteraciones:** Explicación del desarrollo del proyecto dividido por iteraciones.
- **Diseño y estado final de la aplicación:** Diagramas de diseño y análisis del sistema, así como su descripción.
- **Pruebas:** Descripción de las pruebas realizadas durante las iteraciones del proyecto, así como errores detectados.
- **Conclusión y trabajo futuro:** Conclusiones sobre la realización del proyecto e ideas de futuro como posibles mejoras.
- **Apéndices:** Documentos adicionales del proyecto como son el manual de instalación, de despliegue y de usuario, así como el contenido del adjunto al proyecto.
- **Bibliografía:** Referencias de artículos, proyectos y enlaces a páginas web consultadas durante la realización del proyecto.

Capítulo 2

2. Planificación

2.1. Planificación inicial

En el momento en el que se fue asignado este TFG en octubre del curso 2019-2020, el desarrollador se encontraba finalizando sus estudios universitarios por lo que se planteó un trabajo de dos horas diarias hasta comienzos de abril por lo que el tiempo estimado para hacer este proyecto se elevará a unas 364 horas, sobrepasando así las 300 horas mínimas que se especifica en la normativa vigente sobre el tiempo que se debe dedicar a un trabajo de fin de grado en el curso 2019/2020.

A partir de los riesgos estimados y sus planes de acción (Sección 2.3), se detalla una planificación final del proyecto (Sección 2.5). Las 26 semanas de trabajo estimadas se dividirán en cinco interacciones las cuales llevarán pruebas ligeras sobre su funcionamiento básico sin profundizar en estas ya que esto de forma más exhaustiva se realizará en la cuarta iteración. Es importante remarcar en que la memoria se escribirá en paralelo a cada iteración. Las interacciones se dividen de esta forma:

- **Primera iteración (1 semana, 14 horas/hombre):**
  En esta semana se llevarán a cabo reuniones con el cliente (en este caso los tutores del TFG) para especificar los requisitos y funcionalidades del proyecto, así como para exponer la plataforma y herramientas que se utilizará para el desarrollo de esta.

- **Segunda iteración (4 semanas, 56 horas/hombre):**
  En este mes se trabajará sobre el aprendizaje de las tecnologías elegidas para el desarrollo de este proyecto, entre las más importantes se encuentran el lenguaje de programación Javascript y la plataforma de desarrollo Node.js. También se procederá a la instalación y ajuste de estas herramientas y a la comprobación de su compatibilidad, así como a la puesta a punto de la base de datos MySQL que se utilizará. Además, se realizarán pequeñas pruebas de funcionamiento y de conexión.

- **Tercera iteración (14 semanas, 196 horas/hombre):**
  Se procederá a la implementación de las primeras funcionalidades de la aplicación. Se comenzará por llamadas a la API más sencillas como el registro o el login del usuario como toma de contacto, y después se pasará a funcionalidades más específicas del proyecto. En esta iteración se pondrá a prueba el aprendizaje anterior y se intentará implementar todo de la forma más eficiente, flexible y ordenada posible a fin de poder realizar mejor la siguiente etapa.
  Se realizarán pruebas funcionales de cada llamada para asegurarse que se puede pasar a la siguiente llamada sin arrastrar errores.

- **Cuarta iteración (4 semanas, 56 horas/hombre):**
  En esta etapa del desarrollo, se efectuarán pruebas exhaustivas del proyecto, así como las correcciones de código pertinentes.

- **Quinta iteración (3 semanas, 42 horas/hombre):**
  Una vez comprobado de que todo funciona correctamente, se procederá al despliegue de la API en un servidor con sus pertinentes pruebas de conexión y funcionamiento. A la vez de esto, se retocarán los últimos apartados de la memoria del proyecto.

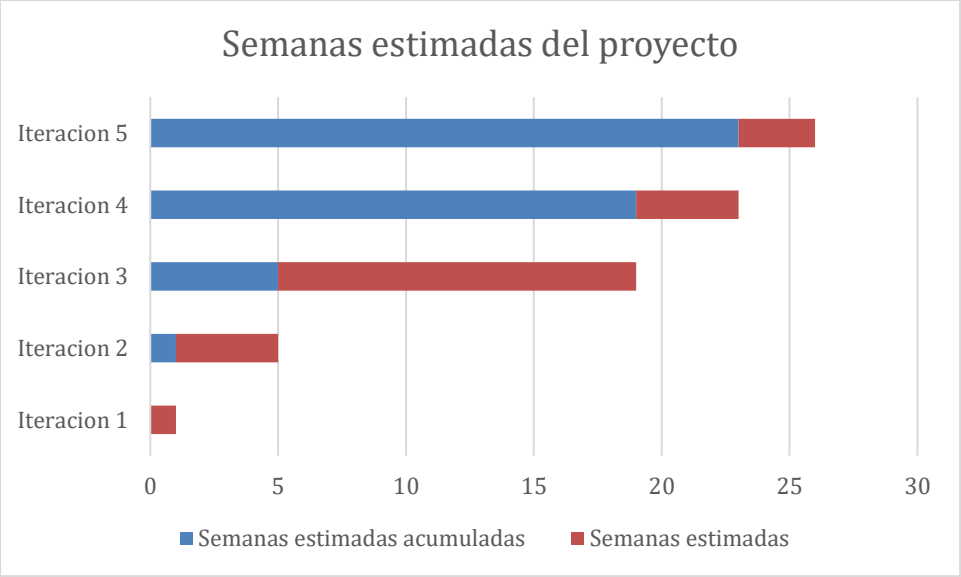

Figura 6. Gráfica del tiempo estimado en semanas

2.2. Entorno tecnológico

2.2.1. Herramientas utilizadas

En este apartado se describen las herramientas utilizadas durante el desarrollo de este proyecto, a su vez que se explica brevemente su función:

- **Telegram** [26]**:** Es una aplicación de mensajería instantánea usada para la comunicación con los tutores.

- **Google Drive** [27]**:** Sistema de guardado de archivos en la nube, usado para compartir documentos que no están relacionados con el código. También se ha utilizado como editor de texto.

- **Skype** [28]**:** Software de comunicación por voz. Fue utilizado para realizar las tutorías en tiempos de la pandemia de la Covid-19.

- **Visual Studio Code** [29]**:** Aplicación para la escritura y edición del código en ordenador.

- **Git for Windows** [30]**:** Conjunto de herramientas que se centra en brindar apoyo para proyectos Git en Windows. En este proyecto, se ha utilizado Git Bash como interfaz para poder utilizar lenguaje de comandos en el sistema operativo.

- **MySQL** [31]**:** Es un sistema de gestión de bases de datos relacional.

- **MySQL WorkBench** [32]**:** Herramienta de administración de bases de datos. Se utiliza como apoyo visual y de edición directa de la base de datos utilizada.

- **Node.js** [13] [14]**:** Es un entorno en tiempo de ejecución multiplataforma basado en el lenguaje de programación JavaScript. Está especialmente orientado a la capa del servidor lo cual lo hace indicado para este proyecto. Una característica de Node.js es su capacidad de añadir módulos a nuestro proyecto de manera sencilla pudiendo incorporar frameworks o librerías que necesitemos. Esto se explica en más detalle en la sección **1.3.4.1.**

- **Npm** [33]**:** Es el sistema de gestión de paquetes por defecto que utiliza Node.js. Es el encargado de gestionar los módulos que se instalan y manejar las dependencias entre estos para que no haya conflictos.

- **Gitlab** [34]**:** Es un servicio web de control de versiones y desarrollo de software basado en Git. Además de un gestor de repositorios el sistema también ofrece alojamiento de wikis y un sistema de seguimiento de errores.

- **Postman** [23]**:** Es una herramienta que se encarga de mandar peticiones a la API desde un programa de escritorio, facilitando mucho la tarea de las pruebas. Para más información, consultar la sección **1.3.4.2.**

A continuación, describiremos brevemente los módulos que hemos utilizado en este proyecto.

- **Express.js** [35]**:** Es un framework que proporciona una infraestructura para aplicaciones web en node.js. Ofrece características muy importantes entre las cuales destacan:
    - Manejador de peticiones HTTP y de rutas URL.
    - Generación de respuestas mediante introducción de datos en plantillas.
    - Opciones de ajustes de la aplicación web como definir el puerto que se utilizara.

- **Sequelize** [36]**:** Es un framework basado en promesas que nos permite hacer transacciones con la base de datos de MySQL. Crea migraciones que sería el equivalente a tener el script de una base de datos y también crea modelos, que servirá para unir la aplicación con la base de datos.

- **Router** [37]**:** Es una librería que permite simplificar las llamadas API que ofrece Express.js

- **Mysql** [38]**:** Framework que nos permite hacer transacciones con la base de datos. Se utiliza como apoyo a Sequelize en alguna función.

- **JWT** [24]**:** Librería que permite utilizar el estándar de Json Web Token como forma para autenticarse de los usuarios. Para más información consultar la sección **1.3.4.3 JWT .**

- **Bcrypt** [39]**:** Librería que permite codificar contraseñas.

- **Is-base64** [40]**:** Librería que nos permite ver si un elemento en base-64 es correcto.

- **Body-parser** [41]**:** Analiza gramaticalmente los parámetros de entrada de una llamada y permite su uso en la API.

- **Validator** [42]**:** Librería que permite detectar errores en parámetros de entrada.

2.2.2. Entorno de trabajo

En la Tabla 1 y Tabla 2 se expondrán las especificaciones del hardware utilizado en este proyecto:

Ordenador portátil	
MSI CX62 6QD	
Hardware	
Procesador	Intel Core i7-6700HQ CPU at 2.60GHz
Memoria RAM	8 GB DDR4
Disco duro	1 TB
Tarjeta gráfica	Nvidia 940 MX 2 GB DDR3 at 1,8GHz
Potencia	120 W
Software	
Sistema operativo	Windows 10
Arquitectura del sistema	64 bits

Tabla 1. Características del ordenador utilizado.

Monitor	
Tipo de pantalla	LCD
Relación de aspecto	16/9
Resolución	1920x1080

Tabla 2. Características del monitor utilizado.

2.3. Estimación de riesgos

En esta sección se describen los posibles riesgos que podrían derivar en retrasos en el proyecto en el caso de que se cumpliesen o incluso cancelación de funcionalidades planeadas. Se muestran estos riesgos en la Tabla 3 en los que se proveerá de una breve descripción, la probabilidad estimada de que suceda con un valor entre 0 y 1 (siendo 0 que no existe posibilidad y 1 que es seguro que ocurra) y un tiempo estimado del retraso en caso de que ocurriese. Una vez estimados los riesgos, se estimará un posible plan de acción en caso de que estos ocurran, a fin de poder minimizar los costes de tiempo que estos pueden conllevar.

Estos planes de acción se describen en la Tabla 4 y cada uno está asociado a su riesgo pertinente. Estos se muestran con una breve descripción de estos acompañados de su exposición al riesgo, que es una estimación del tiempo promedio que se perdería debido al riesgo, calculado como la probabilidad de que ocurra el riesgo por el número de días perdidos estimados.

IDRiesgo	Riesgo	Probabilidad	Descripción	Retraso
R01	Suspenso de la asignatura “Ingeniería del conocimiento”	0.2	El desarrollador tendría que preparar la convocatoria extraordinaria	10
R02	Enfermedad	0.1	El desarrollador debe permanecer en cama por enfermedad	3
R03	Instalación de todas las herramientas de desarrollo	0.5	Dificultad en la instalación y su correcto funcionamiento de las herramientas necesarias	3
R04	Complejidad de las herramientas de desarrollo	0.4	Retraso en el aprendizaje y falta de conocimiento en las herramientas utilizadas	15
R05	Planificación inadecuada	0.8	Mala consideración y gestión de riesgos	20
R06	Problemas con el equipo informático	0.1	Retraso debido a malfuncionamiento del equipo	5
R07	Modificación de los requisitos	0.3	Modificación de los requisitos que provoca modificaciones en las funcionalidades ya hechas	4

Tabla 3. Tabla de riesgos

IDRiesgo	Riesgo	Exposición al riesgo	Plan de acción
R01	Suspenso de la asignatura “Ingeniería del conocimiento”	2 - Crítico	Recuperar horas de trabajo una vez pase el examen
R02	Enfermedad	0.3 - Marginal	Recuperar las horas perdidas una vez que se haya recuperado
R03	Instalación de todas las herramientas de desarrollo	1.5 - Marginal	Buscar soluciones alternativas a los posibles problemas de compatibilidad
R04	Complejidad de las herramientas de desarrollo	6 - Crítico	Estudiar los posibles errores y leer documentación sobre el funcionamiento de las herramientas
R05	Planificación inadecuada	16 - Crítico	Replantear planificación y recuperar horas perdidas
R06	Problemas con el equipo informático	0.5 - Marginal	Llevar a reparación el equipo
R07	Modificación de los requisitos	1.2 - Marginal	Hacer el código más flexible, que permita cambios

Tabla 4. Tabla de posibles soluciones para los riesgos estimados.

2.4. Estimación de costes

En este apartado se detalla la estimación de los costes iniciales de este proyecto y se incluirán costes tanto humanos como técnicos. Esta previsión se realizará contabilizando el IVA y con el salario bruto de un trabajador.

2.4.1 Salario del trabajador

El coste estimado del sueldo del desarrollador se calculará teniendo en cuenta el salario medio de un informático previsto para este tipo de tarea. Según un estudio de remuneración de Michael Page [43] realizado en 2017 indica que el sueldo de un programador con poca experiencia rondaría entre los 18000 y 21000 euros anuales, así que se tomará el valor de 19500 euros como media por lo que

resultaría en aproximadamente 10.83 euros la hora entre las 1800 horas anuales. En este caso, la cuota trabajada sería de 364 horas por lo que obtendremos 3942 euros.

Se tiene que suponer que el proyecto se efectúa a manos de un desarrollador autónomo, por lo tanto, este tendrá que pagar una cuota fija de autónomos de 50 euros al mes según el BOE (Artículo 31 bis).

Por lo que el salario final quedaría en unos 3942 + 50*7 = 4292 euros.

2.4.2 Costes indirectos aplicados al espacio de trabajo

El proyecto ha sido realizado en su totalidad en el domicilio personal del desarrollador, por lo que se tendrán en cuenta los gastos de abastecimiento de la vivienda en vista de las facturas recibidas.

Costes indirectos relacionados con el espacio de trabajo				
Servicio	Coste mensual	Nº de meses	Jornada	Coste total
Luz y gas	200 €	6.5	2 horas/día	101.11 €
Telecomunicaciones	50 €	6.5	2 horas/día	25, 28 €
Total:				126,39 €

Tabla 5. Costes indirectos del espacio de trabajo.

2.4.3 Costes del hardware

En esta sección se debe tener en cuenta el coste del material físico que se ha necesitado para el desarrollo de este proyecto. Se tienen en cuenta el coste del dispositivo, así como las horas totales que tiene de tiempo desde su compra. Para sacar el cálculo del coste aproximado del proyecto en este ámbito, se realizará una tara entre las horas totales del producto y las realizadas exclusivamente en el proyecto.

Costes del hardware				
Dispositivo	Coste de compra	Horas del dispositivo	Horas utilizadas	Coste total
Ordenador Windows	700 €	39000	364	6,53 €

Tabla 6. Costes del hardware.

2.4.3 Costes de software y servicios

La API requiere ser alojada en un servidor en la nube donde se aloja toda la aplicación y base de datos. Para ello se utilizará un servidor localizado en el grupo de investigación reconocido de la Universidad de Valladolid llamado ECA-SIMM. Pero para estimar los costes de uno, utilizaremos un servicio común de hosting como es FireBase el cual cuesta 25 dólares al mes, que serían al cambio 22.15 euros.

Para realizar los diferentes esquemas realizados en el documento del proyecto, se utiliza el programa ASTAH UML el cual es gratuito para el uso de los alumnos de la UVa, pero de no ser este el caso, su coste sería de 4.99 dólares al mes que convertido a euros serían 4.42 euros.

El repositorio utilizado para guardar el proyecto es GitLab el cual da un servicio gratuito a los alumnos de la UVa, pero si no fuera este el caso, tendría un coste de 4 euros al mes.

Costes de software y servicios			
Servicio	Coste mensual	Nº de meses	Coste total
Servidor	22.15 €	6.5	132,9 €
Astah	4.42 €	6.5	26,52 €
GitLab	4 €	6.5	24 €
Total:			183,42 €

Tabla 7. Costes del software y servicios.

2.4.4 Costes totales

En la Tabla 8 se recopilan los costes de todos los aspectos que hemos estimado y se muestra en un cómputo total estimado del proyecto.

Costes totales	
Nombre	Coste
Salario del trabajador	4292 €
Costes indirectos aplicados al espacio de trabajo	126,39 €
Costes del hardware	6,53 €
Costes de software y servicios	183,42 €
Total:	4608,34 €

Tabla 8. Costes totales.

2.5. Planificación final

El resultado final del desarrollo del proyecto no resultó como se estimó. Lo primero anomalía que surgió fue la pandemia del Coronavirus o COVID-19 que no pudo ser prevista. Esto provocó un retraso en la tercera iteración debido a la rotura de la rutina y problemas psicológicos derivados del confinamiento que duró casi 2 meses. Esto provocó un retraso de unas aproximadamente 2 semanas que, aunque no resultaron de forma seguida, se estima que el total de tiempo perdido fue de esta cantidad, por lo que la duración final de la tercera iteración aumentó a unas 210 horas.

Uno de los riesgos considerados fue el posible mal funcionamiento del equipo (R06), lo que conllevó llevarlo a reparar y después recuperar los datos del proyecto, que, si bien estaban guardados en git, se tuvo que rehacer todas las conexiones y configuración del proyecto. Esto sucedió durante la segunda iteración siendo el tiempo de la reparación de 3 días, pero la recuperación del proyecto se amplió otros 2 días más, por lo que el tiempo total de la segunda iteración aumenta a 66 horas.

Otro problema importante que se tuvo durante el desarrollo fue la complejidad de las herramientas de desarrollo (R04) que se tuvo en la segunda y tercera iteración debido a que fueron problemas en dos sitios diferentes.

- En la segunda iteración sucedió por no comprender del todo el funcionamiento de los tokens de acceso a datos del usuario y los tokens de refresco lo que aumentó el tiempo de investigación en una aproximadamente una semana, lo que provocó otro aumento en esta etapa que asciende a 80 horas.
- Por otra parte, en la tercera iteración hubo complicaciones con el entendimiento y manipulación de la sincronización que tiene JavaScript lo que provocó un retraso grave de unas 2 semanas debido a que no encontraba soluciones con las herramientas que estaba utilizando y se tuvo

que recurrir a añadir una herramienta nueva para poder enfrentar el problema de forma efectiva ya que las funcionalidades implicadas no funcionaban de forma fiable. Esto aumenta el número de horas de la tercera iteración a 224 horas.

También se efectuaron algunos cambios menores en los requisitos (R07) debido a cambios en ese momento, o por problemas de entendimiento en la primera especificación de estos, lo que resultó en una serie de cambios menores en la cuarta iteración que se pudieron mitigar a un retraso de 2 días por lo que esta iteración aumenta a 60 horas.

Durante todo el proyecto, el desarrollador tuvo problemas psicológicos debidos a la ansiedad que retrasaron el desarrollo de la aplicación que provocaron unas 10 horas de retraso en cada iteración exceptuando la primera, lo que aumentó aún más el tiempo que se tardó en completar el proyecto. Por lo que los tiempos finales quedarían como:

- **1ª Iteración: 1 semana**
- **2ª Iteración: 5,42 semanas**
- **3ª Iteración: 16,71 semanas**
- **4ª Iteración: 5 semanas**
- **5ª Iteración: 3,71 semanas**

La Figura 7 muestra una gráfica con el tiempo real que duró el proyecto.

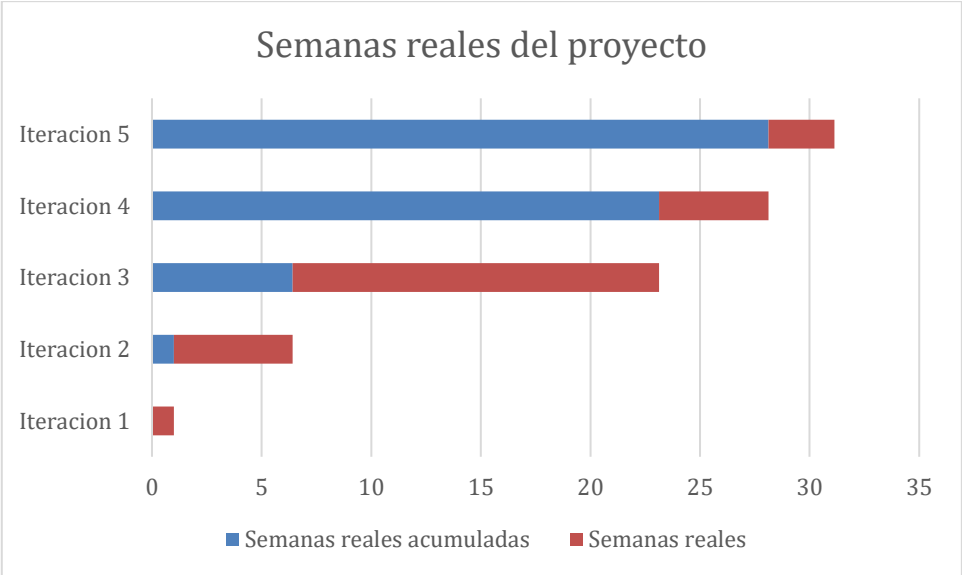

Figura 7. Gráfico con las horas reales del proyecto.

La Figura 8 muestra una comparativa entre las semanas estimadas y reales:

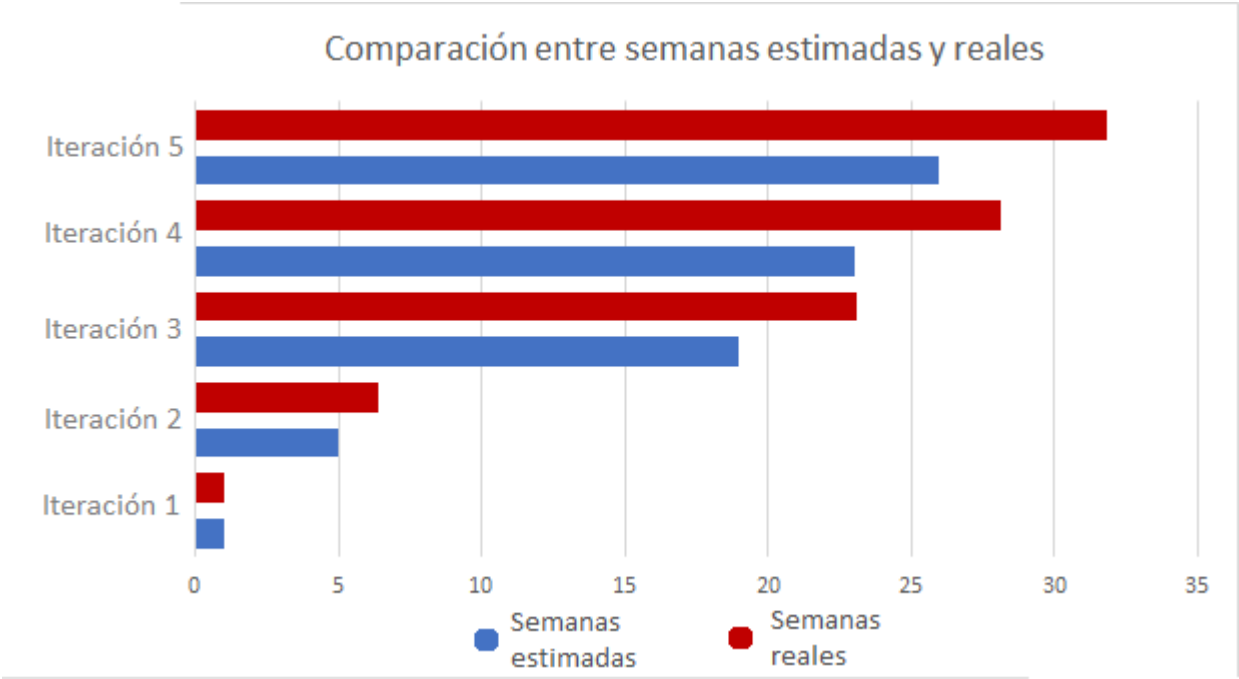

Figura 8. Gráfico con la comparativa entre semanas estimadas y reales

Capítulo 3

3. Análisis del sistema-Descripción de las iteraciones

3.1 Iteración 1

En el comienzo del proyecto, el desarrollador se reúne con el “cliente” que en este caso son los dos tutores del TFG, con la intención de conocer las funcionalidades del proyecto. Esto sirvió como primera toma de contacto con el proyecto para ir identificando los requisitos e historias del usuario necesarias para el desarrollo posterior.

El proyecto parte de la base del proyecto CoP del alumno Rafael Sillero Navajas en el que ahora se busca hacer un servidor API REST para poder alojar el juego. Por esta razón se parte de unos requisitos previos que necesitan satisfacer lo que necesita el proyecto CoP.

Planificación

Una vez que se identificaron los requisitos y el concepto general del proyecto, se dispone a realizar una estimación del tiempo que se empleará en el proyecto, así como explicar la plataforma y las herramientas que se utilizaran en este. Se realiza un boceto de cómo se distribuyen las entidades del proyecto y cómo se relacionan entre sí con el fin de empezar a conocer la estructura que tomará y el procedimiento que se abordará para el comienzo del desarrollo.

Descripción de requisitos

Requisitos funcionales

En esta sección se describirán los requisitos funcionales del proyecto. Estos definen una función que tiene el sistema software dados un conjunto de entradas, comportamientos y salidas. Estos requisitos se muestran en la Tabla 9 junto con un identificador, nombre, descripción y prioridad respecto a las demás (pudiendo ser alta, media o baja).

ID	Nombre	Descripción	Prioridad
RF01	Registro del usuario	El sistema permitirá crear un nuevo usuario.	Alta
RF02	Identificación del usuario	El sistema permitirá que un usuario se identifique.	Alta
RF03	Mostrar datos del usuario	El sistema mostrará los datos personales de un usuario.	Media
RF04	Modificación de perfil	El sistema permitirá que un usuario modifique su perfil.	Alta
RF05	Mostrar clasificación	El sistema mostrará la clasificación por puntos de todos los jugadores.	Alta
RF06	Añadir logro	El sistema permitirá añadir un logro a un usuario	Media
RF07	Mostrar logros	El sistema mostrará los logros ya obtenidos de un jugador.	Media
RF08	Buscar oponentes	El sistema buscará oponentes adecuados a la puntuación de un usuario.	Alta
RF09	Crear partida	El sistema creará una partida.	Alta
RF10	Mostrar información de una partida	El sistema mostrará la información completa acerca de una partida.	Alta
RF11	Mostrar información de las partidas de un usuario	El sistema mostrará la información sobre las partidas jugadas por un usuario.	Media
RF12	Comenzar turno	El sistema permitirá a un usuario jugar su turno.	Alta
RF13	Finalizar turno	El sistema permitirá finalizar el turno de un jugador.	Alta

Tabla 9. Tabla de los requisitos funcionales del proyecto

Requisitos no funcionales

Los requisitos no funcionales describen propiedades del sistema, así como definir restricciones y criterios que muestran el cómo se comporta este. Estos requisitos se muestran en la Tabla 10 junto con su identificador, nombre, descripción y relevancia en el sistema (puede ser crítica, deseable o baja).

Id	Nombre	Descripción	Relevancia
RNF01	Notificación de errores	En caso de que alguna entrada de datos u operación por parte del usuario sea incorrecta, el sistema devolverá un mensaje explicando el problema.	Crítica
RNF02	Almacenamiento en base de datos	El sistema deberá tener acceso a una base de datos de MySQL, la cual se conectará con gracias al framework Sequelize de Node.js donde realizará operaciones con esta.	Crítica
RNF03	Formato de palabras	El sistema utilizará un formato JSON para gestionar las palabras.	Crítica
RNF04	Multiplataforma	El sistema podrá ser utilizado desde diferentes sistemas operativos, como Android 5.0+, iOS 10.0 o superior, Windows 7 en adelante o Linux 2.6 o superior.	Deseable
RNF05	Escalabilidad de la base de datos	El sistema deberá permitir que se añadan nuevos lenguajes (idiomas) de forma manual en la base de datos.	Crítica
RNF06	Facilidad de ser instalado	El sistema debe de ser fácil de instalar en cualquier equipo siempre que se siga el manual de instalación adjunto en esta memoria.	Deseable
RNF07	Privacidad de datos sensibles	El sistema debe ser capaz de encriptar datos sensibles de un usuario (contraseñas) utilizando la librería BCrypt de Node.js.	Crítica
RNF08	Seguridad en el acceso a la aplicación	El sistema utilizará un sistema de tokens que sigan el estándar JWT para realizar peticiones y recibir respuestas del servidor de forma personal y segura.	Crítica

Tabla 10. Tabla de los requisitos no funcionales del proyecto

Requisitos de información

Los requisitos de información son aquellos que enumeran propiedades u objetos que son importantes en el sistema. Estos requisitos se muestran en la Tabla 11 junto con su identificador, nombre y descripción.

Id	Nombre	Descripción
RI01	Datos de los usuarios	El sistema almacenara a los usuarios con los siguientes campos: - Identificador único del usuario.

		- Nombre del usuario. - Correo electrónico del usuario. - Contraseña del usuario. - Puntuación del usuario. - Fecha de la última conexión del usuario.
RI02	Datos de las partidas	El sistema almacenara a las partidas con los siguientes campos: - Identificador único de la partida. - Identificador del creador de la partida. - Boolean de si la partida es una tarea. - Modo de juego de la partida. - Turno actual de un jugador de una partida. - Dificultad de una partida. - Nombre de una partida. - Nombre acortado de una partida. - Temporizador de una partida. - Rondas de una partida.
RI03	Datos de las palabras	El sistema almacenara a las palabras con los siguientes campos: - Identificador único de la palabra. - Nombre de la palabra. - Pronunciación fonética de la palabra. - Audio de la palabra. - URL de la palabra.

		- Dificultad de la palabra.
RI04	Datos de los lenguajes	El sistema almacenara a los lenguajes con los siguientes campos: - Identificador único del lenguaje. - Nombre del lenguaje. - Código del lenguaje.
RI05	Datos de los logros	El sistema almacenara a los logros con los siguientes campos: - Identificador único del logro. - Nombre del logro. - Descripción del logro. - Imagen del logro.
RI06	Almacenamiento de los tokens de refresco	El sistema almacenara a los tokens de refresco con los siguientes campos: - Identificador único del token. - Contenido del token. - Tiempo de expiración del token.

Tabla 11. Tabla de los requisitos de información del proyecto

Descripción de API

En este proyecto se va a implementar un servicio API REST en Node.js.

La API funcionará como servidor de un juego móvil enfocado al aprendizaje de palabras de diferentes idiomas (CoP). Se utilizará una base de datos de MySql para guardar toda la información del programa.

Se realizarán 14 llamadas en la API que constituirán estas características que se quieren implementar (se detallaran en la seccion de Anexo **C. API**):

- Registro e identificación de usuarios con un sistema de reconocimiento por token.
- Sistema de logros los cuales podrán ganar los usuarios a medida que cumplan ciertos objetivos.

- Un ranking de todos los usuarios basados en su puntuación.

- Búsqueda de oponentes para las partidas según tu rango actual en el ranking, buscando el enfrentamiento entre usuarios de parecido nivel.

- Partidas las cuales se podrán jugar por turnos o todos los jugadores a la vez.

- Información acerca de todas las partidas, tanto acabadas o como en curso.

- Al final de estas, se otorgará una puntuación a cada jugador, incluyendo un extra si logró sobresalir frente a los demás lo cual le permitirá subir en el ranking.

3.2 Iteración 2

En esta iteración se comienza una toma de contacto con las herramientas que se utilizaran durante el proyecto a fin de tener un mejor entendimiento sobre en lo que se trabaja para posteriores iteraciones.

Implementación de código básica

Como primera toma de contacto del proyecto, se realizan pequeñas implementaciones de código cómo comprobar el envío y recepción de las peticiones, la importación de módulos, o transacciones de información de prueba con la base de datos.

Esto permite ir familiarizándose con el lenguaje utilizado (JavaScript) e ir realizando pequeñas comprobaciones de configuraciones y conexiones del proyecto, a fin de agilizar la codificación posterior.

Despliegue de tecnología: estado de la cuestión de herramientas actuales

En esta parte, se empieza a desplegar la plataforma de trabajo, para esto se crea el proyecto utilizando una plantilla de Express en el que se crean los directorios y ficheros básicos para poder arrancar el programa. Además de esto, se instalan los módulos que se conocen de antemano que serán utilizados para ir probando su funcionalidad.

Se descarga y se instala también otros programas externos que se utilizarán como apoyo del proyecto como MySQL WorkBench o Postman en los que se conectarán con el proyecto y sus respectivas configuraciones. Además de esto se creará un repositorio virtual donde alojar el proyecto para tener una copia de seguridad.

Pruebas de concepto

Estas pruebas se realizan para comprobar si se saben aplicar bien los conceptos que se necesitan a la hora de realizar el proyecto. Algunos ejemplos de conceptos serían la utilización del formato JSON, la no sincronicidad de JavaScript o aplicar correctamente el estándar de JWT.

3.3 Iteración 3

En esta iteración se comienza la programación de la funcionalidad que se busca en este proyecto. Esto engloba las numerosas llamadas que tienen que se requieren y que estas funcionen correctamente. Esta es la iteración más larga del proyecto.

Implementación de código avanzada

La implementación avanzada consiste en comenzar la codificación de las funcionalidades específicas del proyecto. Esto se basa en utilizar el lenguaje de programación, la plataforma y las herramientas disponibles para crear la funcionalidad que satisfaga los requisitos especificados.

Esta iteración fue la más larga y compleja en comparación a las otras, y esto es debido a la gran implementación de código que se hacía en esta fase. Se fueron implementando poco a poco todos los requisitos del proyecto, sin embargo, algunos de ellos al ser más complejos y debido a la inexperiencia del desarrollador resultaron en que se utilizara más tiempo en ello para poder solucionar los problemas.

Los principales problemas que se han hecho frente han sido:

- **Callback hell:** Este nombre viene del inglés y significa “el infierno de las llamadas de retorno” y esto se debe principalmente a que JavaScript es un lenguaje asíncrono, por lo que, si se llama a una función, puede que esta no tenga el resultado de su operación antes de que el código continúe su con su flujo normal. Esto ha provocado que muchas veces que se ha llamado a la base de datos (sobre todo cuando se realizaban búsquedas múltiples) no se tuviera un resultado y por tanto el programa devolviese constantemente un valor indefinido que provocaba errores. Esto se pudo solucionar utilizando promesas adicionales, y utilizando el módulo de MySQL para poder gestionar mejor las búsquedas múltiples que Sequelize no podía hacer bien.

- **Búsquedas múltiples:** Este problema vino de la mano del anterior, ya que llego un momento que se necesitaban hacer búsquedas de varios elementos de un grupo de elemento, por lo que al principio se pensó en hacer bucles con framework de Sequelize, pero sin embargo esto ocasiono los callback hell hablados en el anterior punto, ya que el programa tardaba más de lo planeado y continuaba su flujo. Esto se solucionó utilizando otro modulo distinto que fue el de MySQL, que sí que permitía realizar búsquedas múltiples de una sola vez haciendo que el tiempo de ejecución que se empleaba en las llamadas de la base de datos, se redujese significativamente.

Es importante también detallar la implementación del estándar JWT para gestionar las sesiones de los usuarios registrados en la aplicación. La primera vez que se identifique alguien en la aplicación, el sistema le generará un token de acceso el cual estará firmado con el identificador único del usuario por lo que será propio de él y que nadie más poseerá. Este token tendrá una fecha de expiración y

cuando este pase, dejará de ser válido y se necesitará obtener una nueva clave para poder usar la aplicación usando un token de refresco que será un token que se le dará al usuario a la vez que el de acceso, pero con un tiempo de expiración mayor o volviendo a identificarse en el sistema. El token de acceso es enviado a la aplicación a través de una cabecera especial de HTTP llamada “Authorization” y dentro de esta, se utiliza la palabra clave “Bearer” para referirnos a esta. De esta manera el token solo puede ser accedido de una manera.

Pruebas funcionales

Al mismo tiempo que se implementan las llamadas de la API, se van realizando pruebas funcionales sobre estas, a fin de ir comprobando que funcionan correctamente antes de proseguir con la siguiente llamada.

Esto se hace para ir solucionando errores que surgen durante la codificación y que estos no se acumulen después. Con esto se puede comprobar que la llamada realiza perfectamente su función y se puede pasar a la siguiente sin ningún tipo de problema.

Redacción memoria

A medida que se realiza la implementación del código, se va haciendo paralelamente la redacción de la memoria del proyecto en donde se comienza escribiendo las estimaciones de tiempos y riesgos además de los requisitos ya especificados anteriormente, así como la información general de las herramientas utilizadas. A medida que se va completando el proyecto, se pueden ir añadiendo más y más contenido a la memoria.

3.4 Iteración 4

En esta iteración se busca terminar la parte práctica del TFG y para ello se realizan pruebas y correcciones de última instancia de cosas que no terminan de encajar bien con lo que se busca.

Pruebas

Una vez terminada la implementación total del proyecto, se realizan pruebas completas y cubriendo numerosos casos diferentes de entrada para cubrir la mayor parte de errores que se pueda.

Estas pruebas se hacen con muchos casos diferentes y además, como si se tratara de un usuario nuevo que explora la aplicación por primera vez, lo que permite ver si todas las funcionalidades están bien cohesionadas.

Cambios de código generales tras revisión final

Después de realizar las pruebas, se apuntan donde se ocasionaron errores o donde el código no funcionó de la manera esperada. Posteriormente a esto, se procede a realizar una revisión completa del código de la llamada donde se encontró el error para solucionarlo.

Además de esto, también se realiza un análisis de la funcionalidad y si es necesario, se corrigen algunos matices de los requisitos para adecuar aún más el proyecto a lo que se busca. Al cambiar estos detalles, se tiene que hacer una corrección en el código para mostrar estos cambios nuevos.

Redacción memoria

Durante esta iteración se continúa con la redacción de la memoria, añadiendo más información a medida que se van completando los pasos. En este punto ya pueden realizar algunos diagramas, completando el documento de la API, las historias de usuario… entre otras secciones.

3.5 Iteración 5

En esta iteración final, se procede a completar finalmente el documento de la memoria del TFG y el despliegue de la aplicación.

Redacción de la memoria

En este punto se escriben los últimos detalles de la memoria y se completan secciones que no se podían haber realizado anteriormente como por ejemplo los diagramas de actividad, el tiempo real dedicado al proyecto y las pruebas finales realizadas. Se da también una revisión completa a la memoria y se corrigen los fallos que hubieran.

Despliegue

Para el despliegue, se hizo una reunión con los tutores del proyecto para poder subir el proyecto a un servidor del grupo universitario ECA-SIMM. En esta reunión se siguieron los pasos mostrados en la sección C. Manual de despliegue.

Capítulo 4

4. Estado final de la aplicación

4.1 Diagramas de análisis

4.1.1 Historias de usuario

En esta sección se describirán las diferentes historias de usuario del proyecto. Como podemos ver en la Figura 9 todas las historias las realiza un cliente API REST en los que no interactúa ningún actor más y en las que todas son independientes entre sí. Es importante aclarar que, aunque las historias no dependen unas de otras, la mayoría de ellas necesitan que el usuario esté identificado en el sistema para poder realizar la funcionalidad principal de la llamada, exceptuando “Register” y “Login” en los que no hace falta esto.

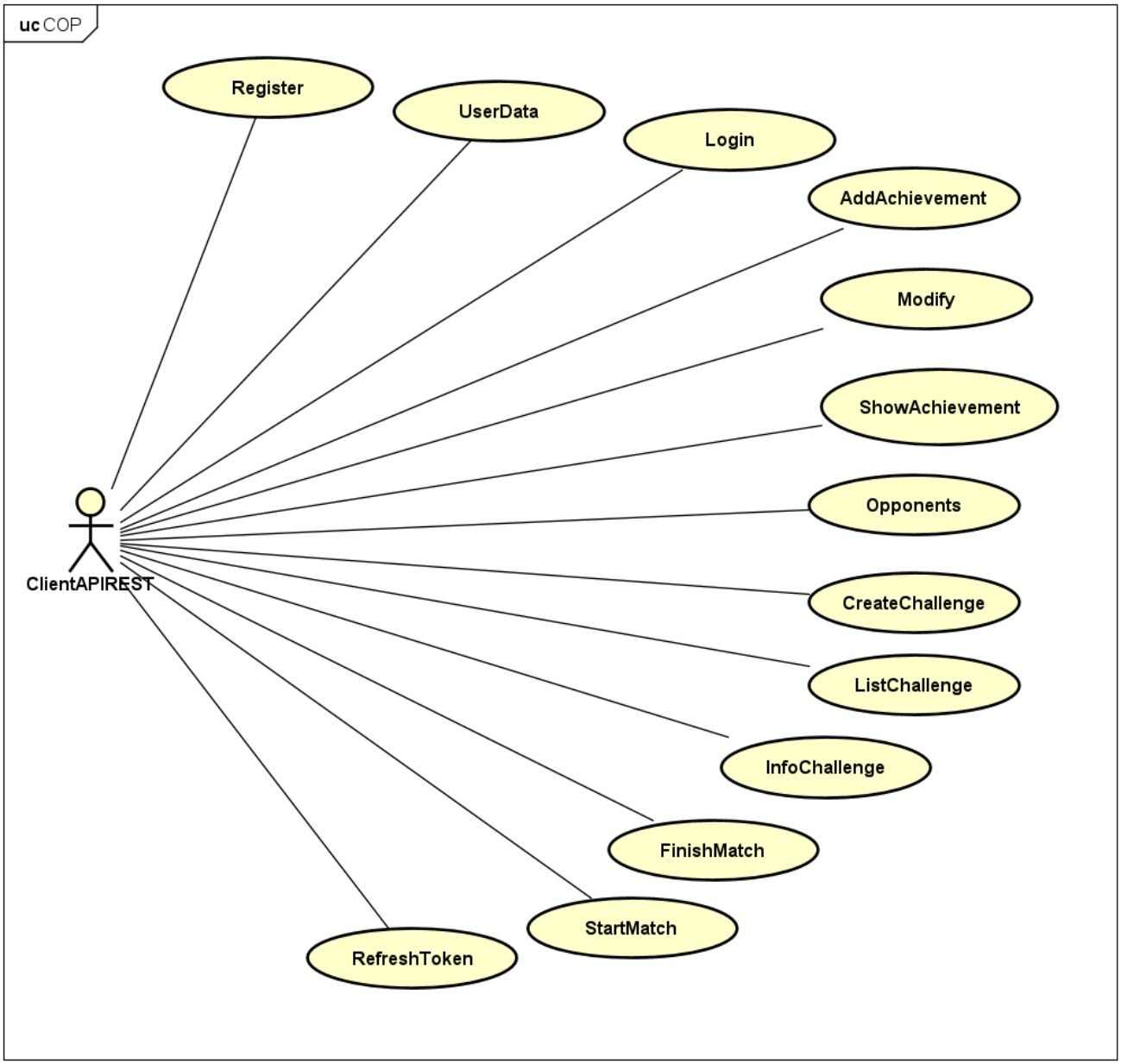

Figura 9. Diagrama de los casos de uso

A continuación, se expondrán todas las historias de usuario. Se detallarán en forma de tabla como se muestra en Tabla 12:

4. Estado final de la aplicación

Historia de usuario	
Id	
Nombre	
Prioridad	
Riesgo	
Descripción	
Validación	

Tabla 12. Plantilla usada para describir las historias de usuario.

- **Id:** Identificador único de cada historia de usuario. Su nombre se dará por las letras “HU” seguidas del número correspondiente.
- **Nombre:** Nombre corto para mostrar brevemente la función de la historia.
- **Prioridad:** Preferencia del desarrollo de la historia comparadas con el resto. Pueden ser “alta”, “media” y “baja”.
- **Riesgo:** Estimación del peligro que podría acarrear un fallo en la historia respecto al proyecto total. También se califica como “alto”, “medio” y “bajo”.
- **Descripción:** Información resumida de la funcionalidad de la historia de usuario.
- **Validación:** Condiciones que deben completarse para que la historia se considere como finalizada.

Historia de usuario	
Id	HU01
Nombre	Registrar un usuario (Register).
Prioridad	Alta
Riesgo	Alta
Descripción	Como usuario, quiero registrarme en el sistema.
Validación	- Quiero poder registrarme en el sistema con un nombre de usuario, email y contraseña. - Quiero que tanto el nombre como el email sean únicos. - Quiero poder registrar una imagen como avatar propio.

Tabla 13. HU01

Historia de usuario	
Id	HU02
Nombre	Identificar a un usuario (Login).
Prioridad	Alta
Riesgo	Alta
Descripción	Como usuario, quiero identificarme y entrar con mis credenciales en el sistema.
Validación	- Quiero poder identificarme de manera segura en el sistema. - Quiero ser notificado en caso de error en alguno de los campos. - Quiero que se quede constancia de quién soy para próximas operaciones.

Tabla 14. HU02

Historia de usuario	
Id	HU03
Nombre	Devolver datos del usuario (UserData).
Prioridad	Media
Riesgo	Media
Descripción	Como usuario, quiero ver mis propios datos
Validación	- Quiero poder recibir todos mis datos personales. - Quiero poder conocer la puntuación que tengo en el juego. - En caso de tener una imagen, también quisiera poder verla.

Tabla 15. HU03

Historia de usuario	
Id	HU04
Nombre	Modificación de perfil (Modify).
Prioridad	Alta
Riesgo	Medio
Descripción	Como usuario, quiero cambiar mis datos personales.
Validación	- Quiero poder cambiar mis datos personales de forma segura. - Quiero poder cambiar la imagen del avatar si quiero por otra.

Tabla 16. HU04

Historia de usuario	
Id	HU05
Nombre	Mostrar el ranking de jugadores (Ranking).
Prioridad	Alta
Riesgo	Media
Descripción	Como usuario, quiero ver la clasificación por puntos de los jugadores
Validación	- Quiero poder ver la clasificación por puntos de los jugadores.

Tabla 17. HU05

Historia de usuario	
Id	HU06
Nombre	Mostrar los logros de un jugador (ShowAchievements).
Prioridad	Media
Riesgo	Media
Descripción	Como usuario, quiero ver los logros que he obtenido durante el tiempo.
Validación	- Si tengo logros, quiero ver una lista de ellos con su información. - Si no tengo logros, quiero saber que no tengo ninguno aún.

Tabla 18. HU06

Historia de usuario	
Id	HU07
Nombre	Añadir logro (AddAchievements).
Prioridad	Media
Riesgo	Media
Descripción	Como usuario quiero poder añadirme logros.
Validación	- Quiero poder añadirme un logro existente a mi cuenta. - Si no existe el logro, quiero ser informado de ello.

Tabla 19. HU07

Historia de usuario	
Id	HU08
Nombre	Buscar oponentes (Opponents).
Prioridad	Alta
Riesgo	Alta
Descripción	Como usuario quiero buscar oponentes contra los que jugar y que tengan un nivel parecido al mío.
Validación	- Quiero que me emparejen con otros usuarios activos para poder jugar una partida. - Quiero que me emparejen con otros usuarios activos que tengan un nivel parecido al mío. - En caso de que no haya suficientes jugadores disponibles, quiero que se me informe.

Tabla 20. HU08

Historia de usuario	
Id	HU09
Nombre	Crear partida (CreateChallenge).
Prioridad	Alta
Riesgo	Alta
Descripción	Como usuario, quiero crear una partida contra otros jugadores.
Validación	- Quiero crear una partida con todas las especificaciones que quiero. - Quiero que se registre una lista de jugadores que jugarán en esa partida. - Quiero registrar las palabras y lenguajes que se utilizaran en la partida. - En caso de error, quiero que no se guarde nada.

Tabla 21. HU09

Historia de usuario	
Id	HU10
Nombre	Mostrar información de la partida (InfoChallenge).
Prioridad	Alta
Riesgo	Media
Descripción	Como usuario, quiero poder consultar la información acerca de una partida.
Validación	- Quiero recibir información sobre la partida. - Quiero recibir información sobre los jugadores de la partida. - Quiero recibir información acerca de las palabras usadas en la partida y su emparejamiento. - Quiero conocer de quien es el turno en la partida en caso de no estar acabada.

Tabla 22. HU10

Historia de usuario	
Id	HU11
Nombre	Mostrar información de las partidas de un jugador (ListChallenges).
Prioridad	Media
Riesgo	Media
Descripción	Como usuario, quiero consultar el historial de partidas en las que jugué.
Validación	- Quiero recibir información sobre todas las partidas en las que participe - Quiero recibir información sobre los jugadores que participaron en esas partidas. - Quiero poder filtrar las partidas entre las acabadas e inacabadas. - Quiero poder filtrar las partidas por fecha.

Tabla 23. HU11

Historia de usuario	
Id	HU12
Nombre	Comenzar el turno de juego (StartMatch).
Prioridad	Alta
Riesgo	Alta
Descripción	Como usuario quiero comenzar mi turno de una partida.
Validación	- Quiero poder ejecutar mi turno si es que de verdad me toca. - Si no me toca, no debo ser capaz de ejecutar mi turno.

Tabla 24. HU12

Historia de usuario	
Id	HU13
Nombre	Finalizar turno de una partida (FinishMatch).
Prioridad	Alta
Riesgo	Alta
Descripción	Como usuario, quiero terminar mi turno de la partida.
Validación	- Quiero poder finalizar mi turno de una partida. - Quiero que se me sumen los puntos que he ganado. - Si no es mi turno, no debo ser capaz de finalizar el turno. - En cualquier caso, debo saber el resultado de la operación. - Si es el final de la partida, quiero ser informado de ello.

Tabla 25. HU13

Historia de usuario	
Id	HU14
Nombre	Refrescar el token de acceso (RefreshToken).
Prioridad	Alta
Riesgo	Alta
Descripción	Como usuario, quiero refrescar mi token de acceso en caso de que se caduque.
Validación	- Si mi token está caducado, quiero que se me actualice por uno nuevo.

Tabla 26. HU14

4.1.2 Modelo de dominio

El modelo de dominio de la Figura 10 nos muestra las relaciones y multiplicidades entre las diferentes entidades que componen el proyecto.

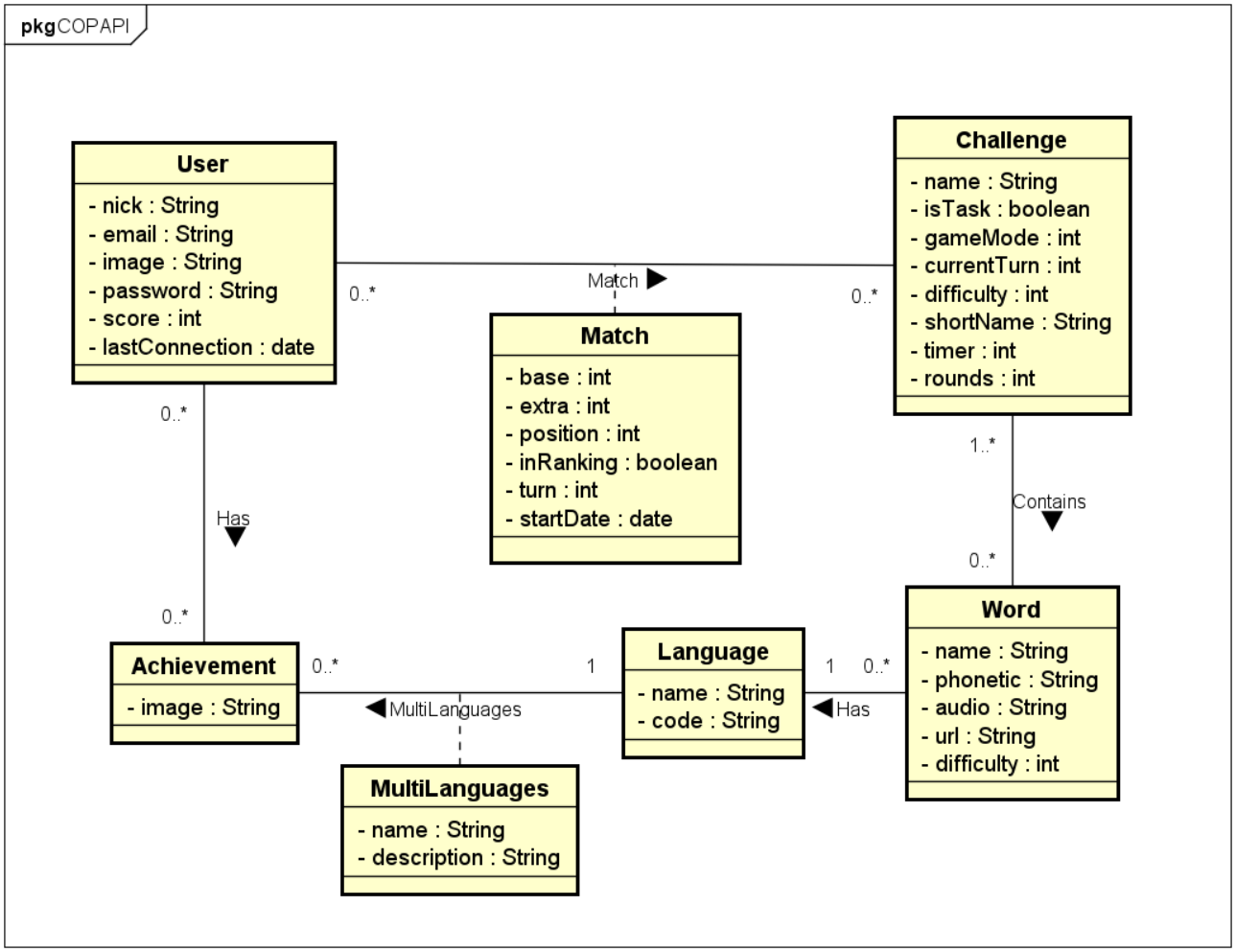

Figura 10. Diagrama del modelo de dominio

A continuación, se describen los elementos mostrados en el modelo de dominio.

Clase: User

- **Descripción:** Clase que modela los atributos de un usuario registrado.

- **Responsabilidades:** Modelar un usuario junto con sus datos personales que permitirá identificarlos en todas las operaciones.

- **Atributos:**
  - nick: Nombre con el que el usuario se dará a conocer en la aplicación.
  - email: Correo electrónico del usuario.
  - image: Imagen de avatar que el usuario tiene la posibilidad de poner.
  - password: Contraseña encriptada del usuario.
  - score: Puntuación total ganada por un jugador
  - lastConnection: Última conexión del jugador, útil para saber si es activo o inactivo.

Clase: Challenge

- **Descripción:** Clase que modela todas las partidas creadas.

- **Responsabilidades:** Modelar una partida con todos los atributos necesarios para poder jugar correctamente.

- **Atributos:**
    - name: Nombre de la partida.
    - isTask: Señala si esta partida es una tarea.
    - gameMode: Modo de juego de la partida.
    - currentTurn: Señala a qué jugador le toca jugar.
    - difficulty: Dificultad de la partida.
    - shortName: Nombre acortado de la partida.
    - timer: Tiempo que dura una partida.
    - rounds: Rondas que dura una partida.

Clase: Match

- **Descripción:** Clase transitiva que modela las partidas individuales de un usuario.

- **Responsabilidades:** Modelar una partida individual de un jugador, diferente de todas las demás

- **Atributos:**
    - base: Puntuación ganada por un jugador en una partida.
    - extra: Puntuación extra que puede ganar un jugador si se cumplen ciertos requisitos.
    - position: Posición del jugador en el ranking justo al crear la partida.
    - inRanking: Si se ha actualizado ya su puntuación, esto impide que pueda volver a actualizarse.
    - turn: turno del jugador en esa partida.
    - startDate: Fecha de comienzo del turno de un jugador en la partida.

Clase: Word

- **Descripción:** Clase que modela las diferentes palabras que se registran.

- **Responsabilidades:** Modelar las palabras que se utilizaran en las partidas.

- **Atributos:**
    - name: Nombre de la palabra.
    - phonetic: Pronunciación fonética escrita de la palabra.
    - audio: Localización del archivo de audio de la palabra.
    - url: Localización de la imagen de la palabra.
    - difficulty: Dificultad de la palabra.

Clase: Language

- **Descripción:** Clase que modela los distintos lenguajes de las palabras con las que se juegan.
- **Responsabilidades:** Modelar la información sobre los lenguajes que se utilizan en la aplicación.
- **Atributos:**
  - name: Nombre del lenguaje
  - code: Código BCP47 del lenguaje

Clase: Achievement

- **Descripción:** Clase que modela la información de los logros.
- **Responsabilidades:** Modelar la información propia de un logro.
- **Atributos:**
  - image: Localización de la imagen de un logro.

Clase: Multilanguages

- **Descripción:** Clase transitiva que modela la información del lenguaje de un logro.
- **Responsabilidades:** Modelar la información que tiene un logro de un lenguaje.
- **Atributos:**
  - name: Nombre del logro en un idioma.
  - description: Descripción del logro en un idioma.

4.1.3 Diagramas de actividad

En este apartado se muestran todos los diagramas de actividad de todas las llamadas del proyecto. En todos se describen un camino de acciones en el que se efectúa un caso correcto de la funcionalidad de la llamada, aunque también se muestran caminos error derivados de una sentencia condicional.

Algo que comparten todos los diagramas de actividad es que se necesita que tanto la entrada de datos como el token para identificar al usuario sean correctos (y en caso del token que no esté caducado tampoco) ya que esto es necesario para que realizar la funcionalidad tal cual se planeó sin problemas durante su ejecución.

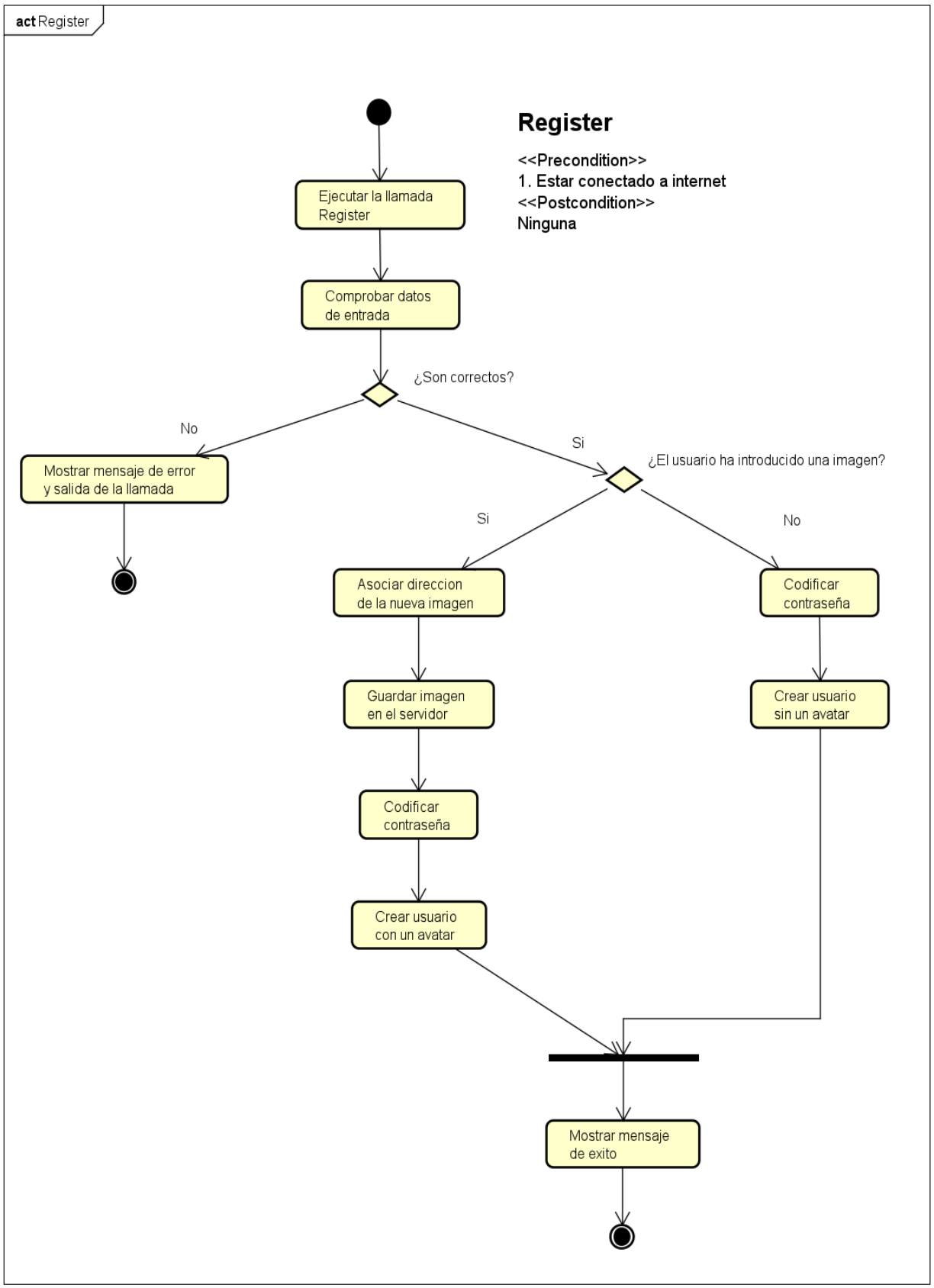

Figura 11. Diagrama de actividad del registro

Una vez que se comprueben que los datos son correctos se procede a registrar al usuario en el sistema, para ello, es importante conocer si el usuario ha introducido una imagen. Si no ha introducido ninguna imagen se procede a simplemente codificar su contraseña e introducirla en la base de datos. Si es el caso contrario, se asigna una dirección única a la nueva imagen y se decodifica para poder guardarla en el directorio de imágenes. Después de esto se codifica la contraseña del usuario y se le mete en el sistema.

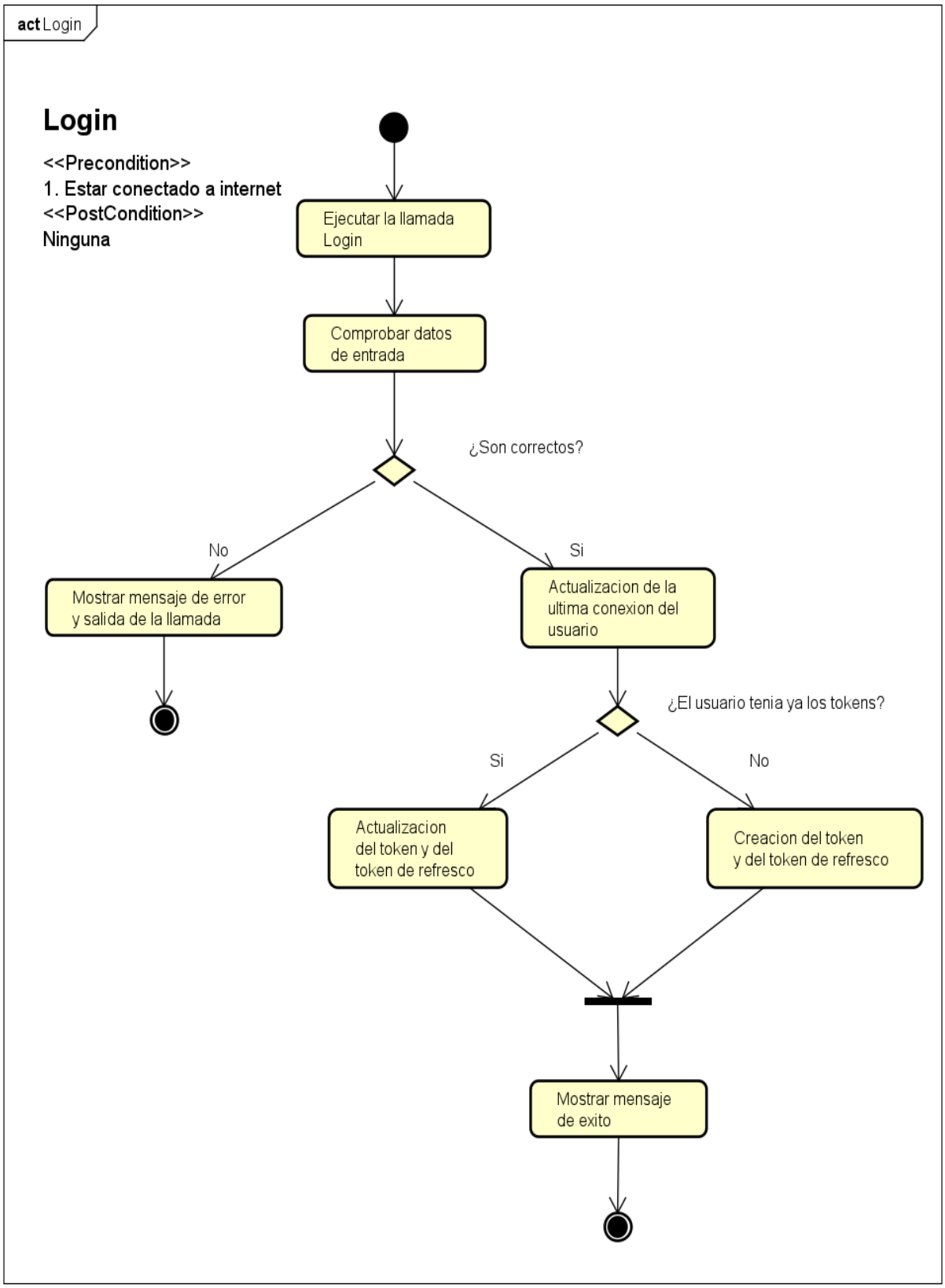

Figura 12. Diagrama de actividad de la autentificación

Una vez que se comprueban que los datos de entrada pertenecen a un usuario y son correctos, se procede a ver si este ya tenía el token de acceso y de refresco anteriormente. Si este es el caso, se actualizarán con una nueva fecha de expiración. En caso contrario, se crearán los dos tokens mencionados antes y se le asignan a su usuario.

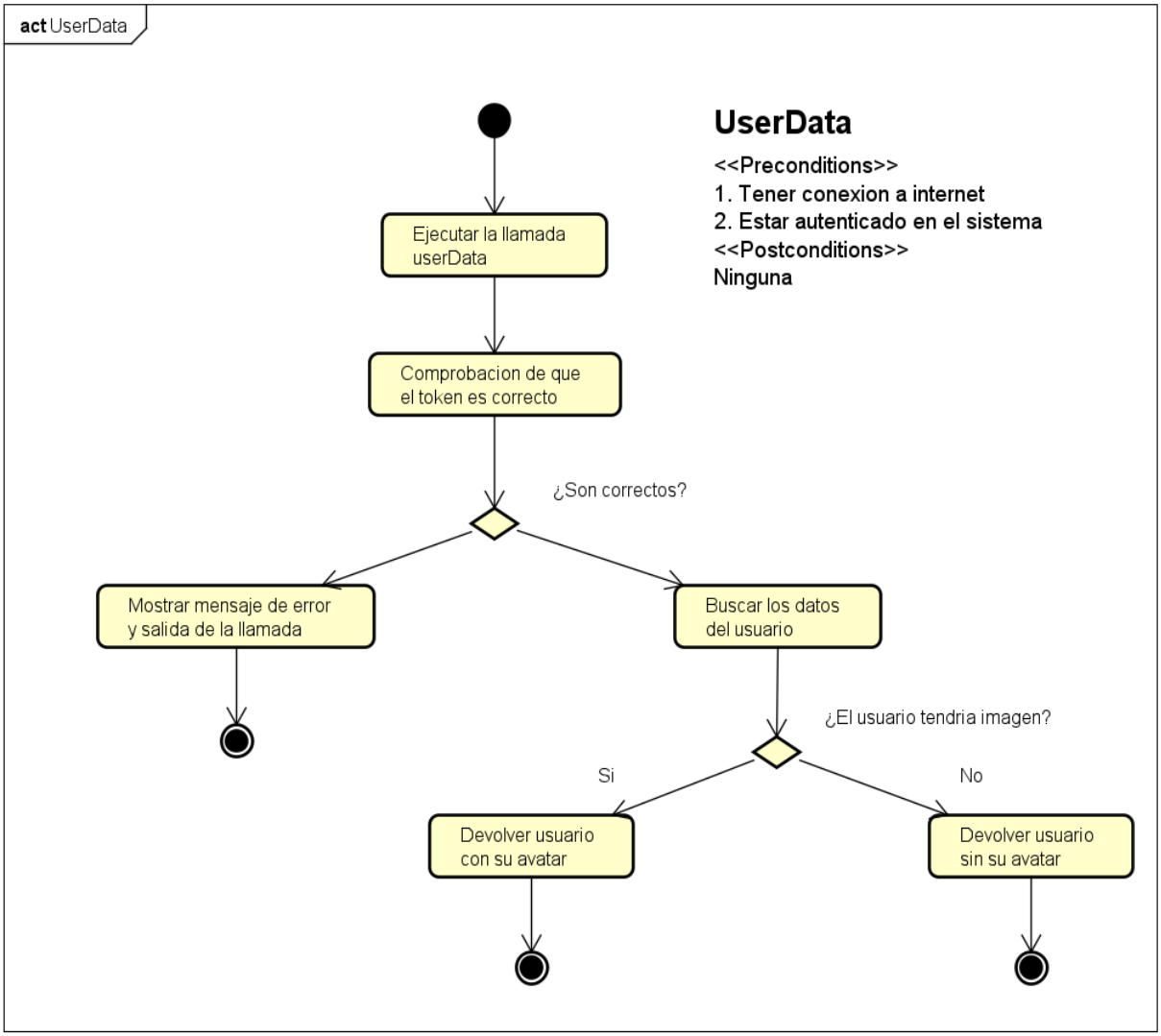

Figura 13. Diagrama de actividad de mostrar datos del usuario

En este diagrama, una vez que se ha comprobado que el token es correcto, solo hay que hacer una diferenciación en si el usuario tiene una imagen o no que devolver.

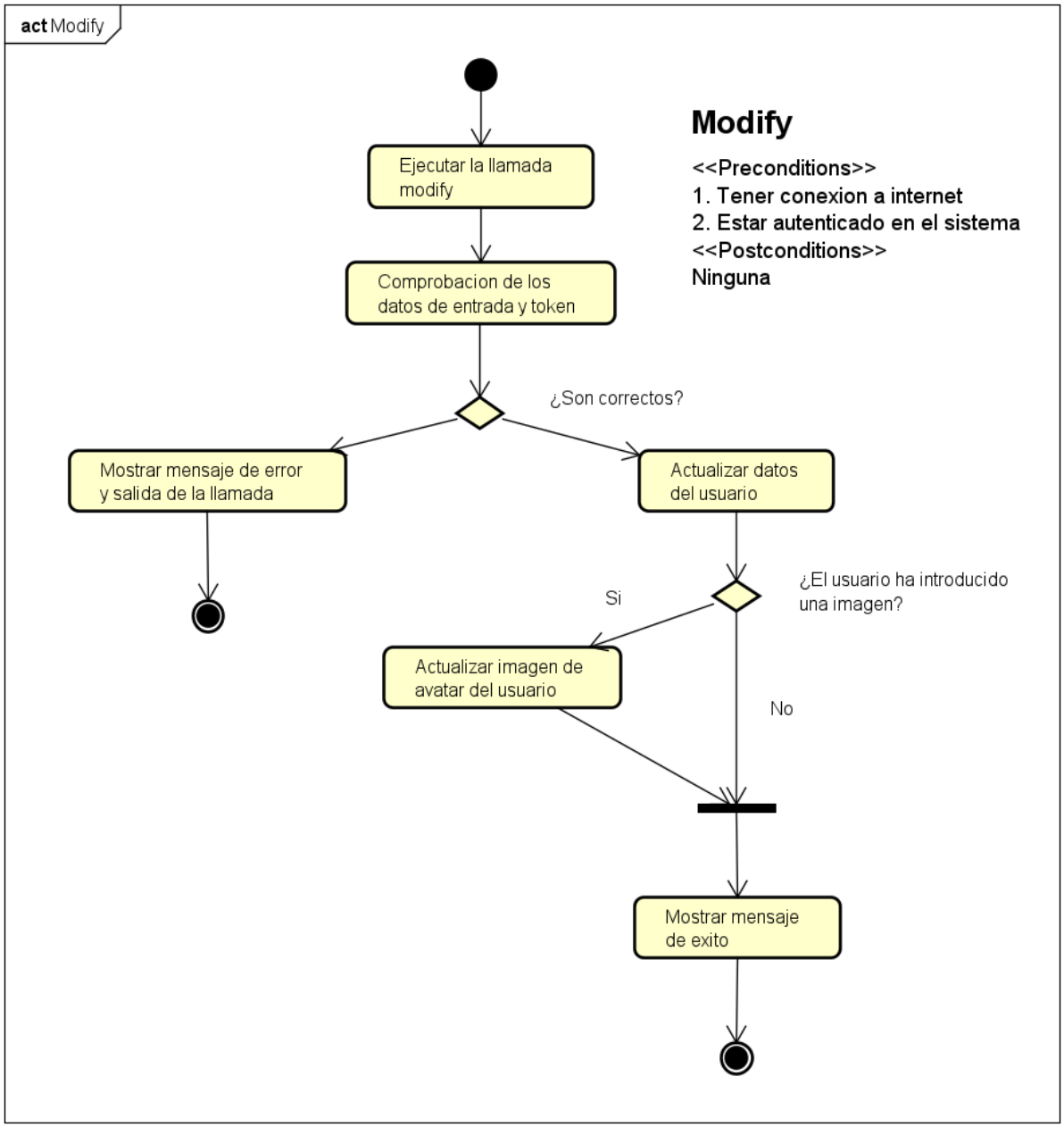

Figura 14. Diagrama de actividad de modificar perfil

Este caso es muy parecido al de Register. En caso de que el usuario haya introducido una imagen, se procede a decodificarla y actualizarla. En caso contrario no hay que realizar este paso.

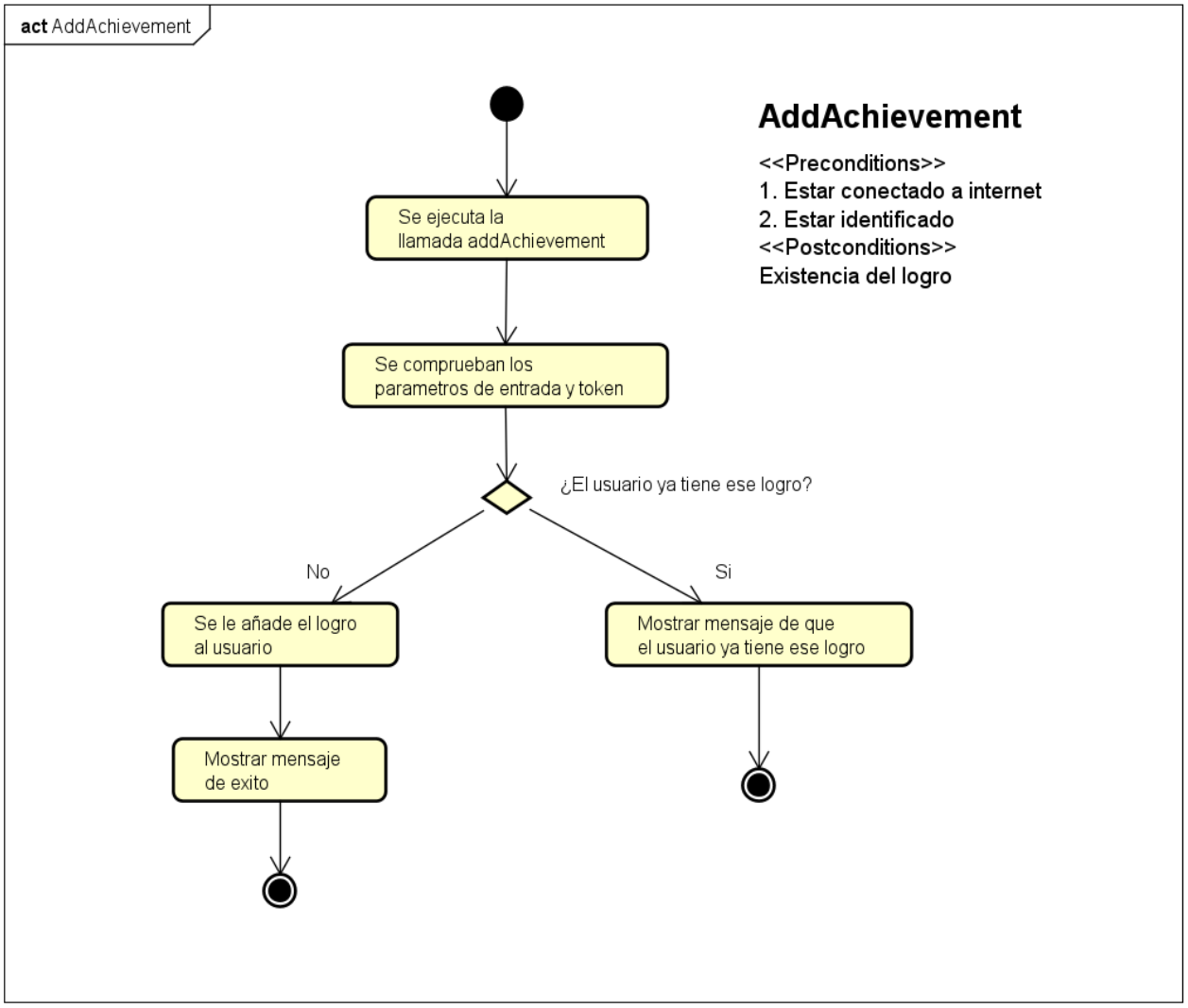

Figura 15. Diagrama de actividad de añadir logro

Para añadir un logro, hay que tener en cuenta en si el usuario ya tenía ese determinado logro o no. En caso de que lo tenga se le añadirá, pero en caso de que no, se mostrará un mensaje de que ya lo tenía.

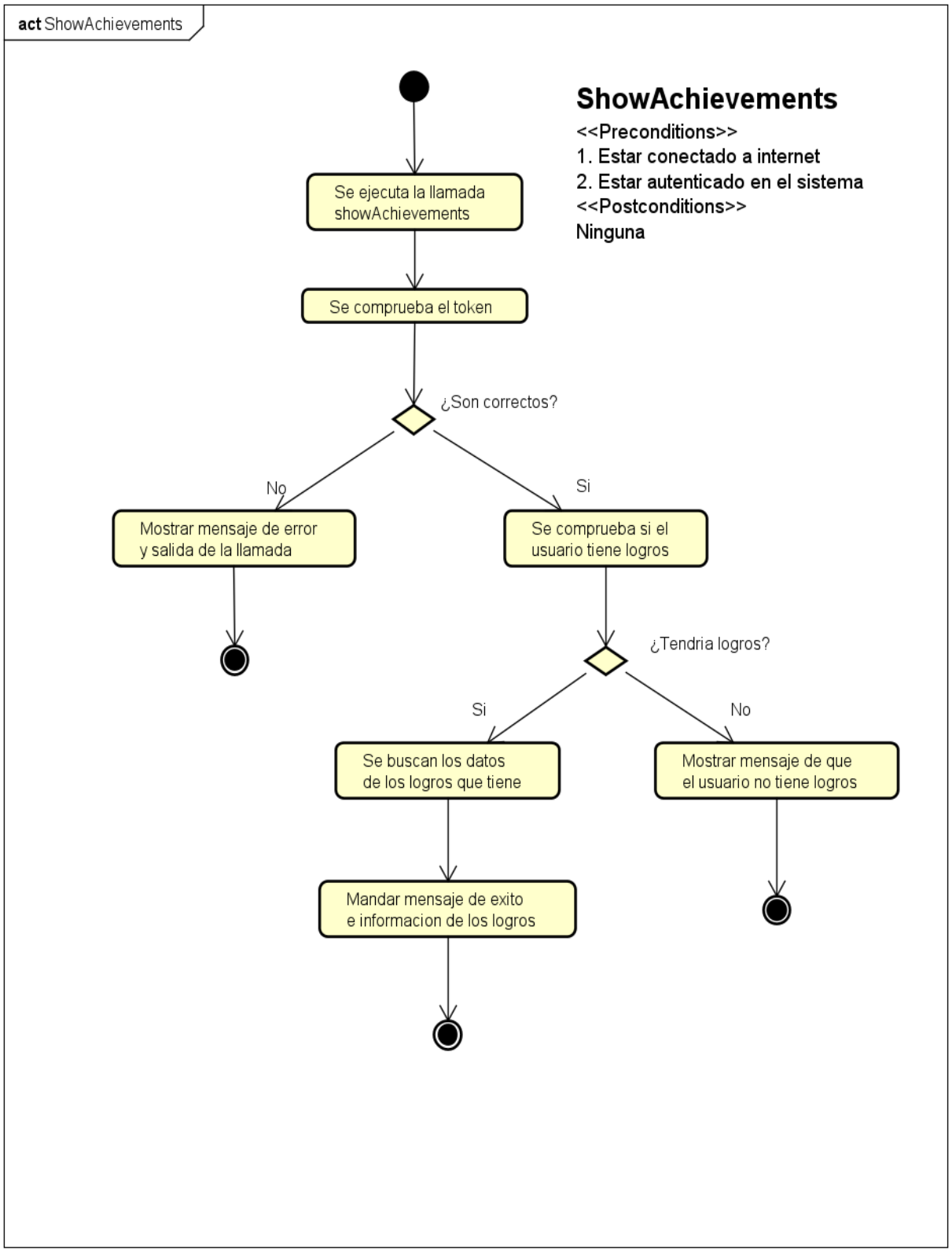

Figura 16. Diagrama de actividad de mostrar logros del usuario

Una vez que se compruebe si el token es correcto, se deberá observar si el usuario tiene algún logro ganado. Si no lo tiene, se mostrará un mensaje de que no tiene ningún logro. En caso contrario se hará una búsqueda de los logros que tenga y se mostrará su información.

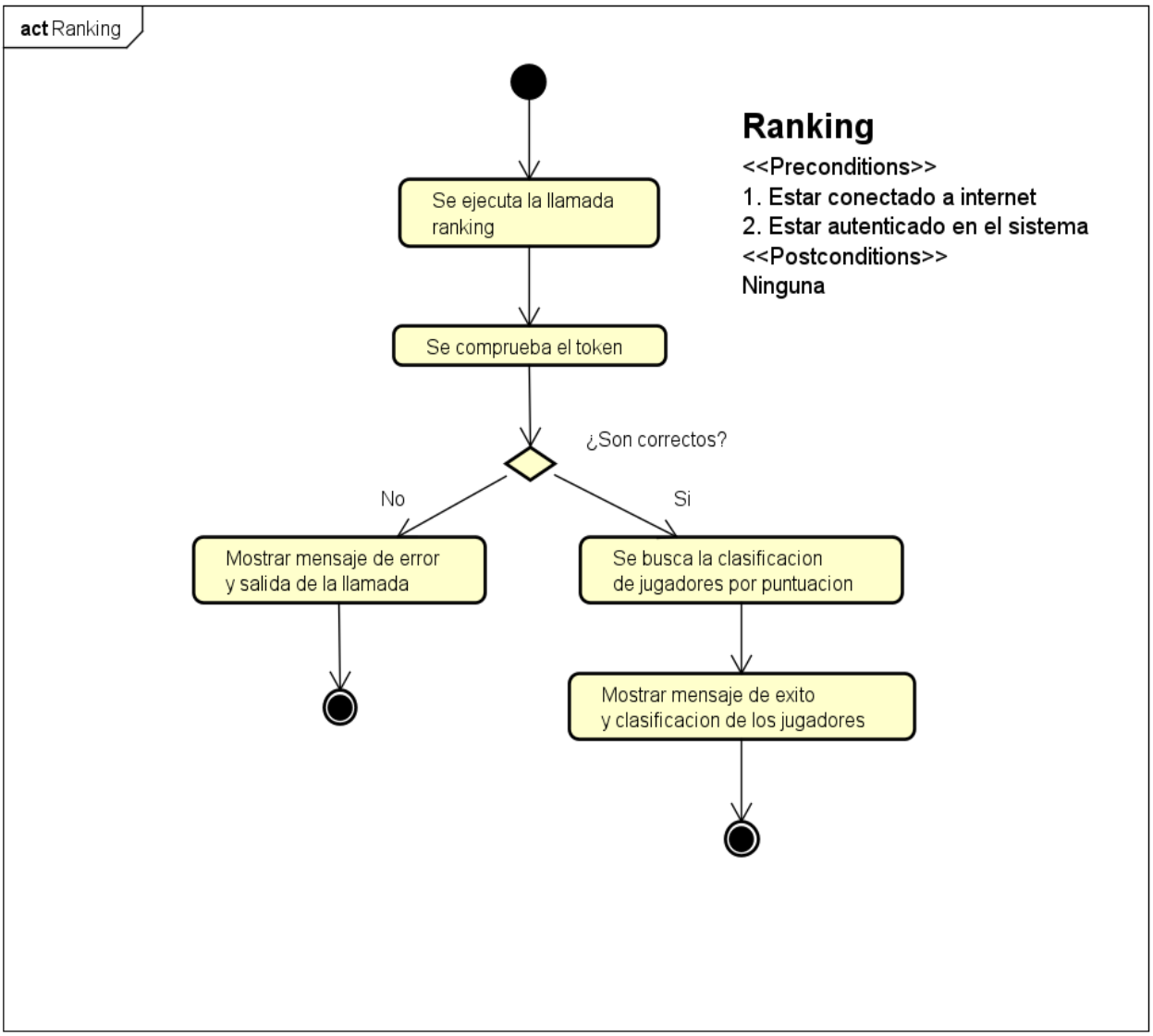

Figura 17. Diagrama de actividad del ranking

En caso de que los datos sean correctos, se mostrará la clasificación general de los jugadores.

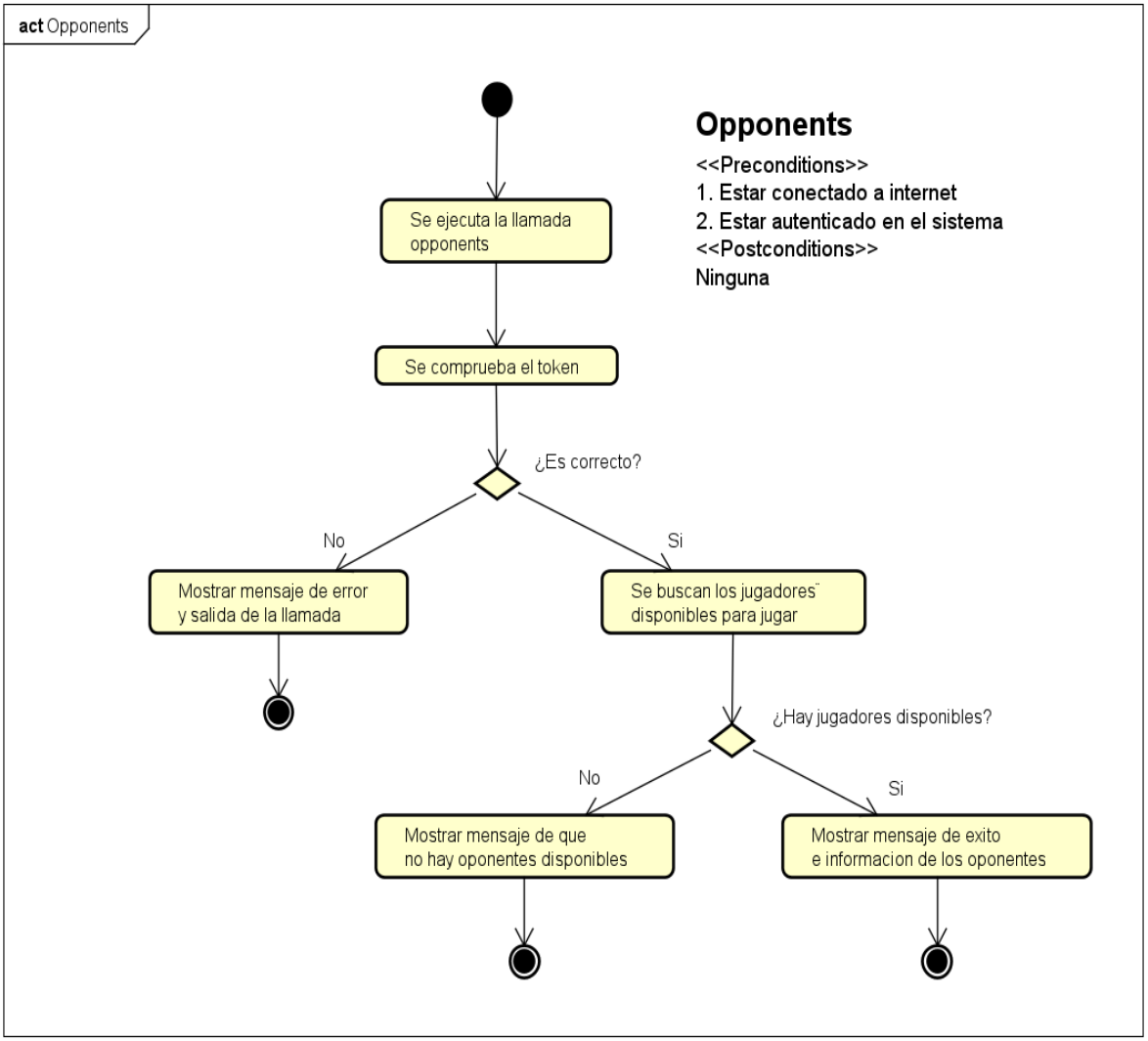

Figura 18. Diagrama de actividad de la búsqueda de oponentes

Una vez que se haya comprobado el token de entrada del usuario, se realiza una búsqueda de los usuarios disponibles para posteriormente crear una partida nueva. Para buscar los usuarios se busca primero a jugadores activos (que se hayan conectado en una cierta cantidad de tiempo) y después por proximidad en puntos respecto al usuario que ha invocado esta llamada. En el caso de que no haya suficientes jugadores que cumplan los requisitos como para formar una partida, se notificará de que no hay oponentes disponibles. En caso contrario se devolverá la información de estos adversarios.

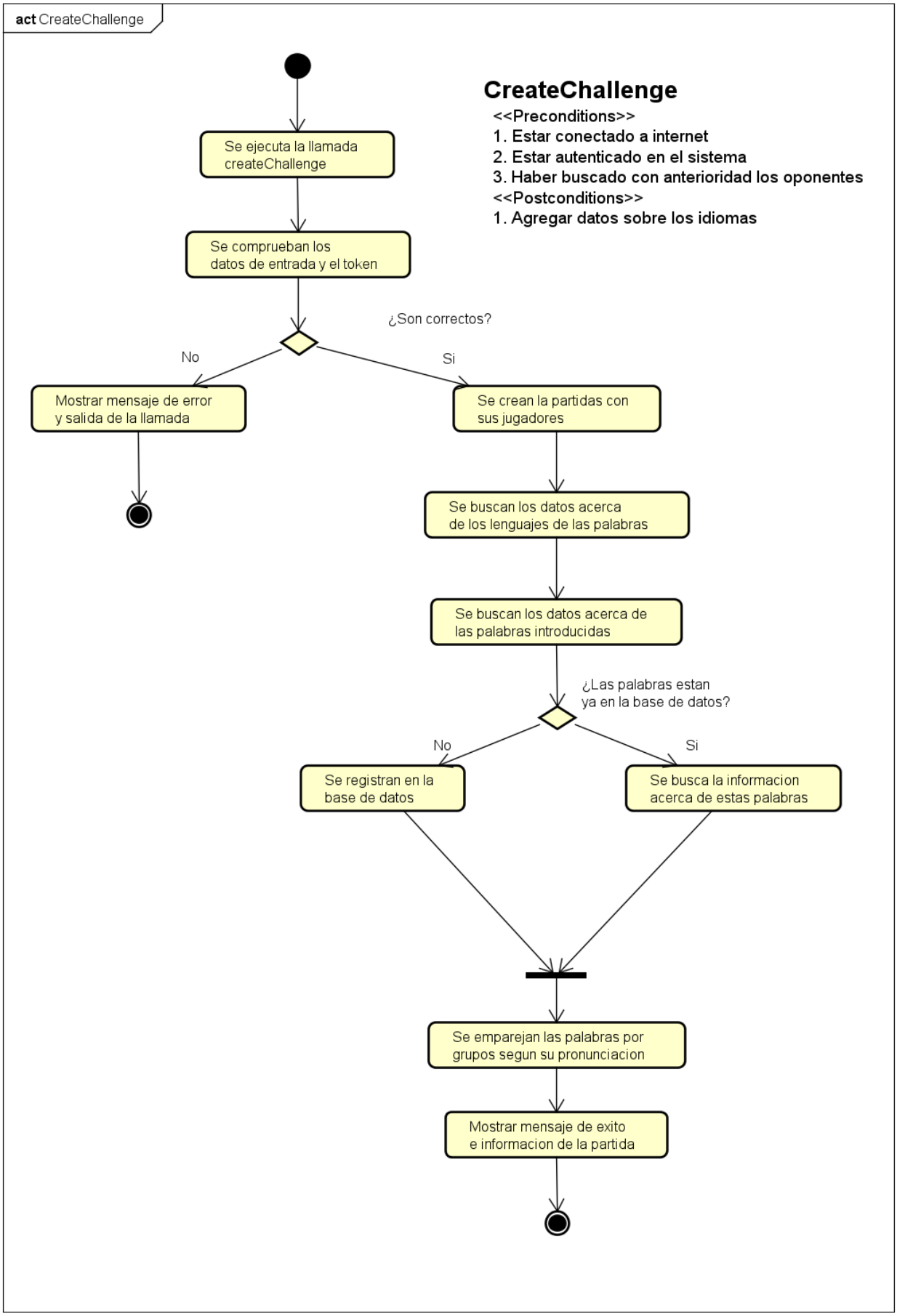

Figura 19. Diagrama de actividad de crear una partida

A la hora de crear una partida, primero se comprueban los numerosos datos de entrada que hay que introducir, después se crea la partida junto con los oponentes encontrados en la llamada anterior, luego se comprueban los datos acerca de los lenguajes de las palabras introducidas y por último se comprueba si estas palabras ya se encontraban en el sistema, si no lo estaban, se meten en este gracias a que se pasa la lista de palabras como entrada y si ya estaban, se realiza una búsqueda a partir de los datos de entrada.

Por último, se agrupan las palabras según su pronunciación, esto viene dado en los datos de entrada también y si todo es correcto, la partida se creará con éxito.

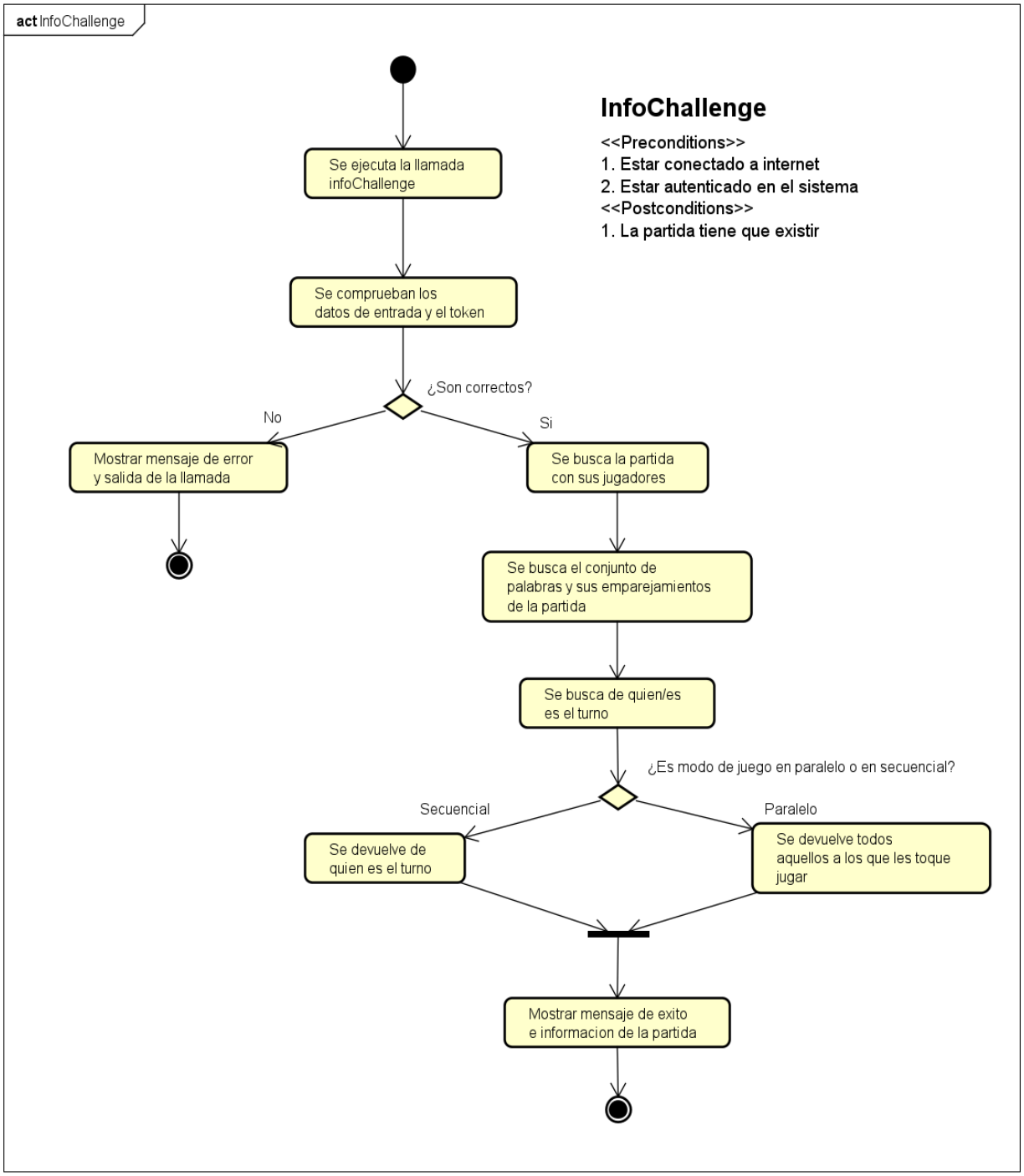

Figura 20. Diagrama de actividad de mostrar información de una partida

En esta llamada, una vez se comprobaron los datos de entrada y token, se realiza una búsqueda completa de toda la información de una partida. Primero se busca la partida y a sus jugadores, después el conjunto de palabras y sus emparejamientos y, por último, de que jugador es el turno en caso de que la partida siga activa. Si es un modo de juego secuencial, se pasará una única identificación del jugador al que le toque. En caso de ser un modo en paralelo, se mostrará la identificación de todos los jugadores a los que les toque.

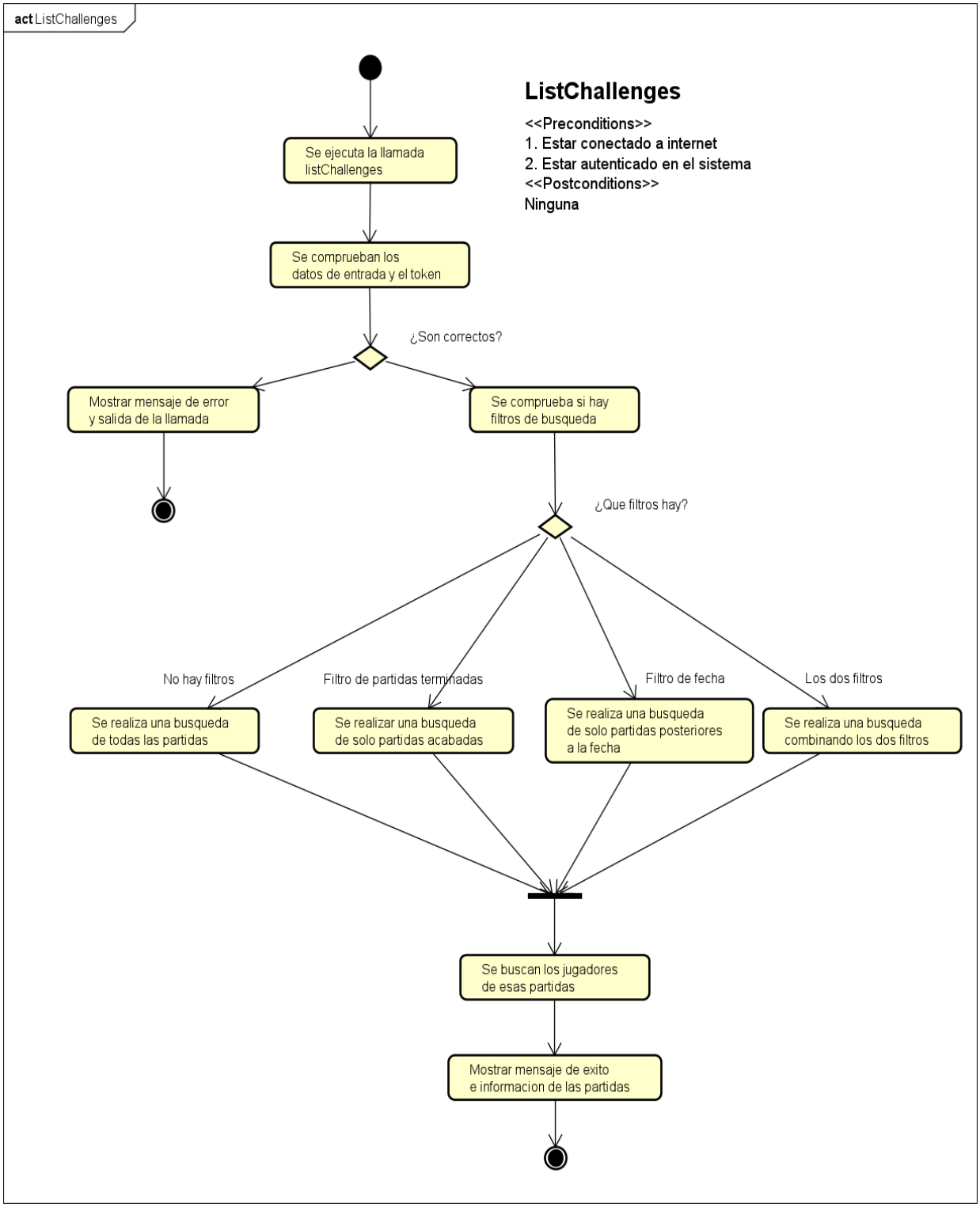

Figura 21. Diagrama de actividad de mostrar las partidas de un usuario

Una vez que se han hecho las comprobaciones pertinentes, se mira si se han seleccionado algún método de filtraje de las partidas. Por defecto se devolverán todas las partidas tanto terminadas como sin terminar, y sin ningún tipo de fecha límite. En caso de activar uno de estos filtros, se devolverán las partidas que cumplan con esa clase de requisitos. En las dos opciones se devuelve tanto la información de las partidas y los jugadores que participan.

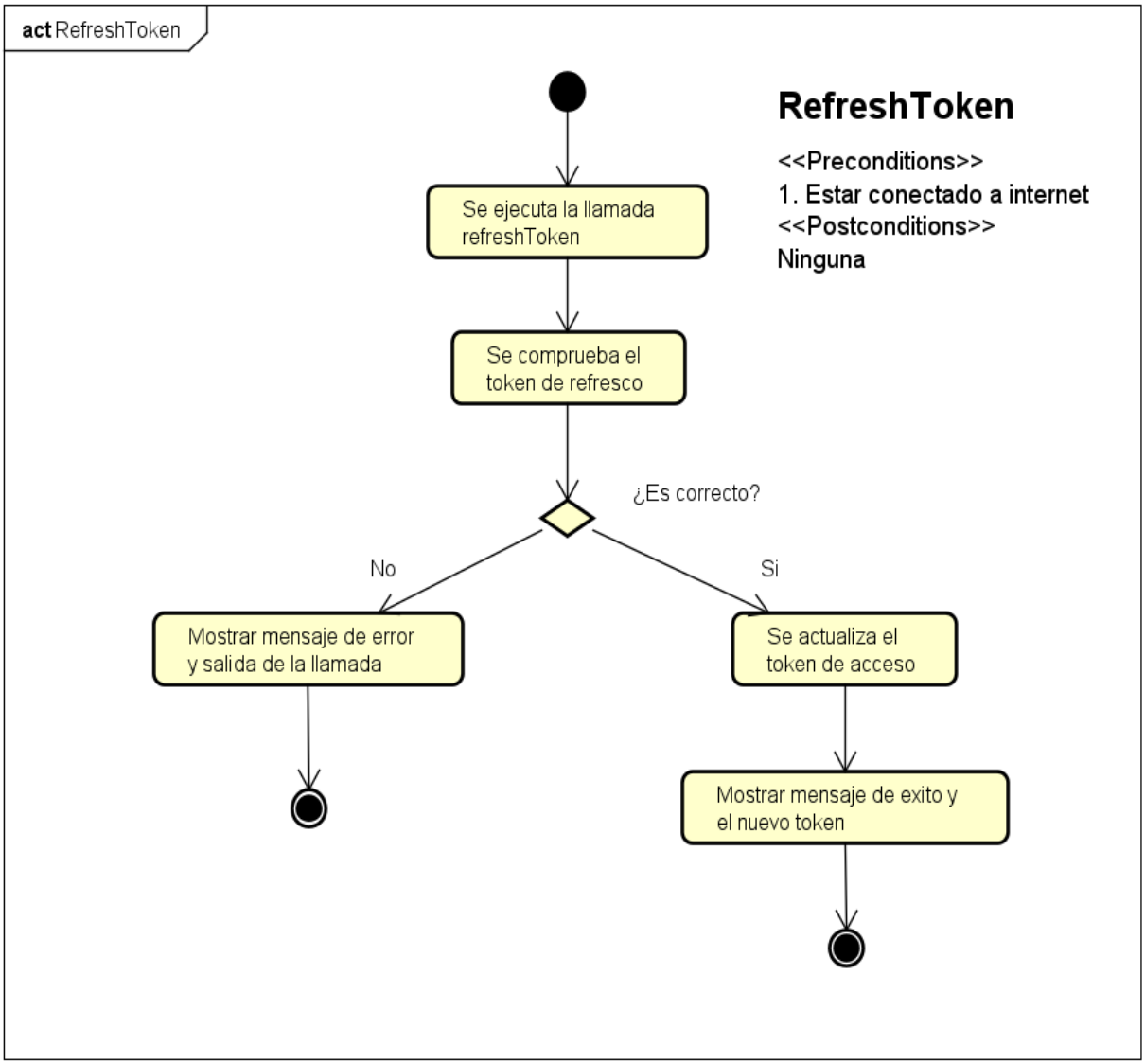

Figura 22. Diagrama de actividad de refrescar el token de acceso

Una vez que se comprueba que el usuario tiene un token correcto y que no está caducado su token de refresco, se le actualiza su token de acceso con una nueva fecha de expiración.

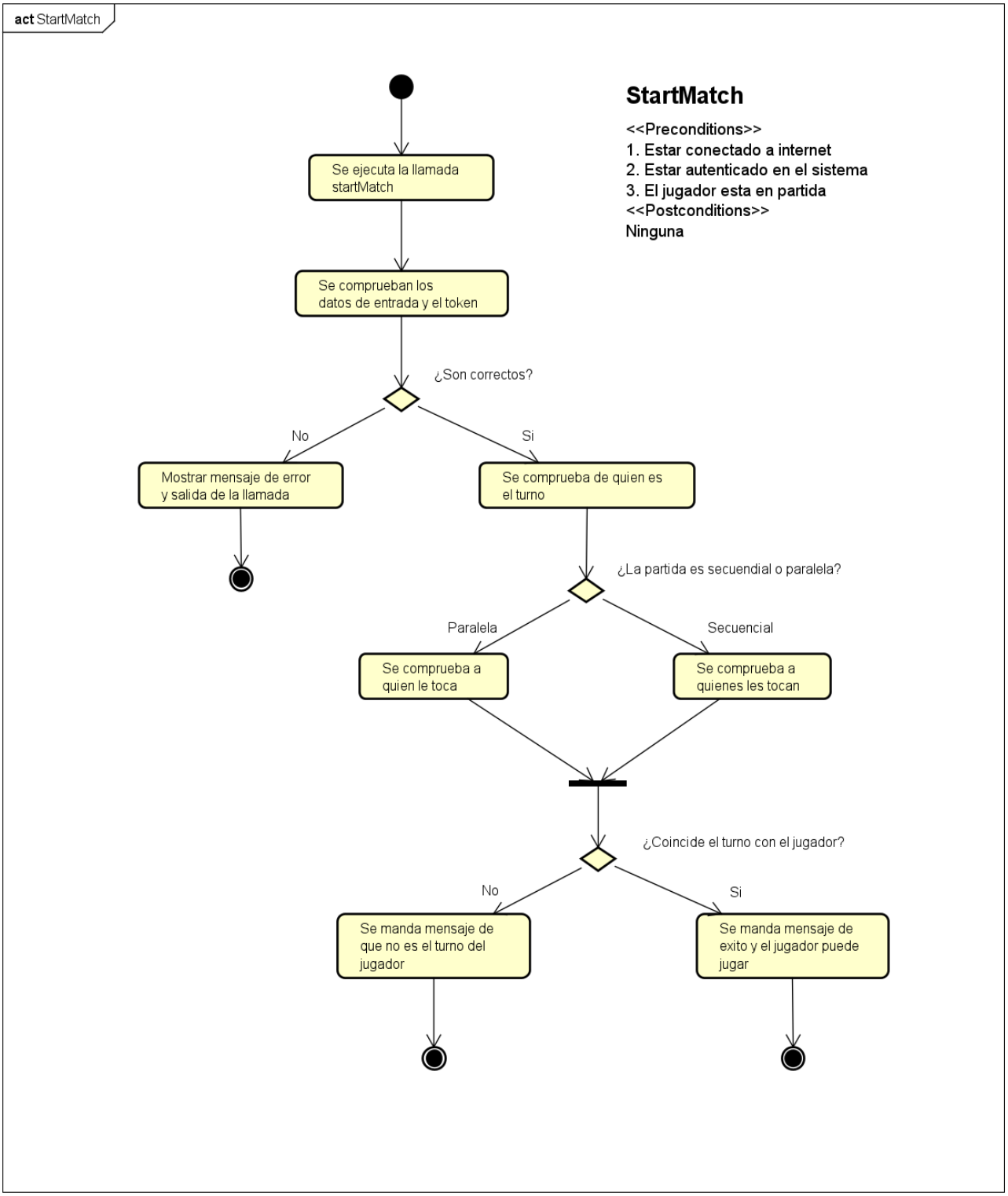

Figura 23. Diagrama de actividad de comenzar turno

Para este caso, después de comprobar los datos de entrada, se comprueba que modo de juego se está jugando en la partida, si es secuencial se devuelve una lista de todos a los que le toquen y se comprueba si el usuario está entre ellos. En cambio, si es secuencial, tiene que coincidir el identificador del usuario con el de quien le toca. Una vez se ha comprobado, si es turno del jugador se notifica que puede jugar y en caso contrario, no podrá hacerlo.

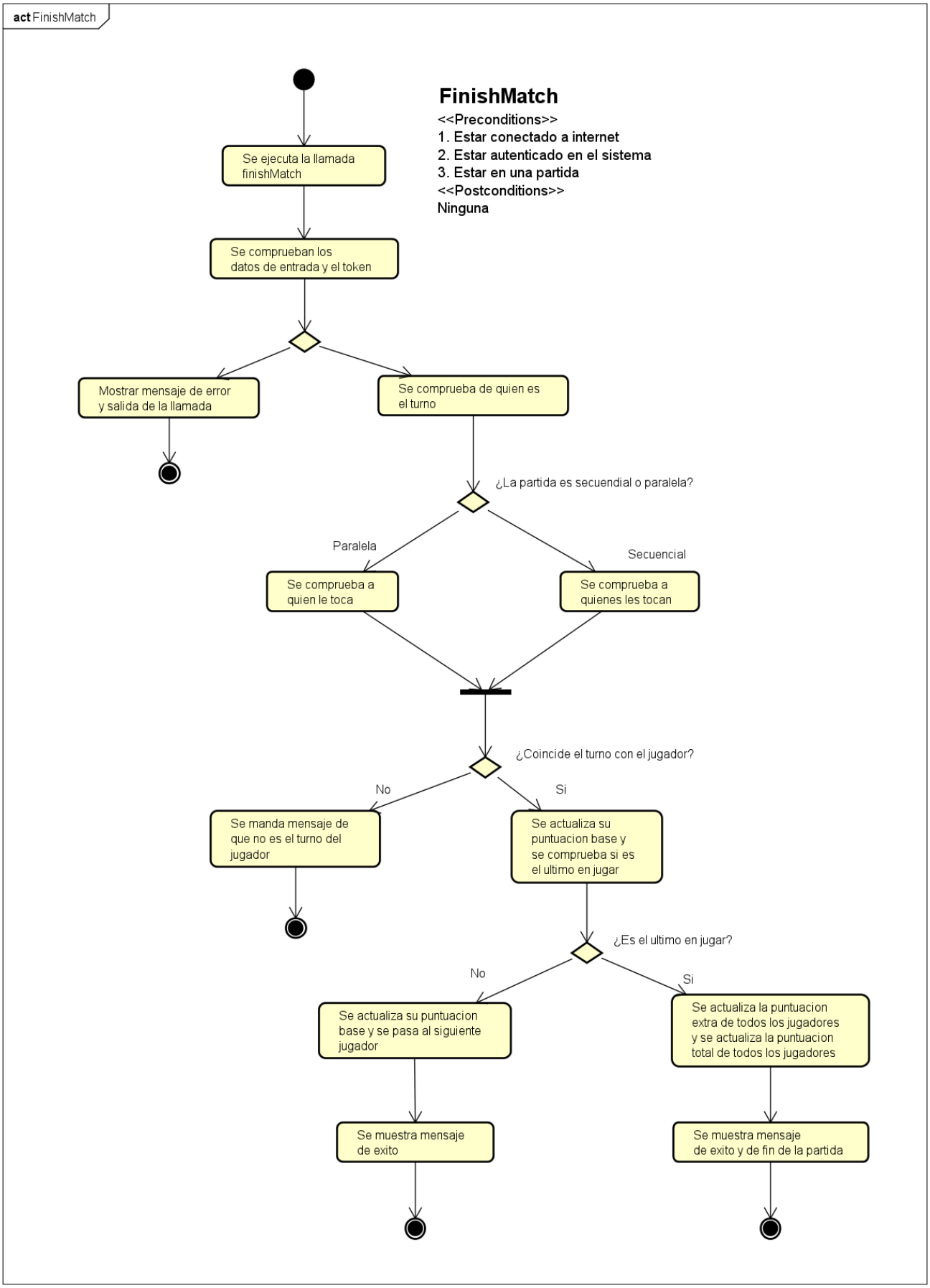

Figura 24. Diagrama de actividad de finalizar turno

El caso para finalizar el turno comienza de forma similar al de comenzar el turno, se comprueba primero si la modalidad es en paralelo o secuencial y después se valida con el identificador del usuario. Una vez pasado este punto, se actualiza la puntuación base del jugador y por último se comprueba si después de esta jugada se termina la partida, si es así, se actualizarán todos los puntos extra y puntuación final de todos los jugadores a la vez. Si este no es el caso, esta acción no se realizará.

La manera en la que se añadirán puntos extra a los jugadores viene dada de la manera en la que, si un jugador en una posición más baja del ranking gana a uno o varios de los que están más arriba que el en la clasificación, este será recompensado con una cantidad de puntos extra para reflejar que ha sobrepasado a jugadores por encima de él y que jugaba con una desventaja de nivel. Al igual que si un jugador con mejor clasificación pierde ante otros con puntuación menor será penalizado debido a que se considera que parte con más ventaja que los demás jugadores.

Condition			Extra points	
IsCreator	MaxBaseScore	BetterRank	$n = 2$	$n \in \{3, 4, 5\}$
T	T	T	0	0
T	T	F	$rank1 - rank2$	$3 * (n - 1)$
T	F	T	$-(rank1 - rank2)$	$-(n - 1)$
T	F	F	0	0
F	T	T	0	0
F	T	F	$rank1 - rank2$	$(n - 1)$
F	F	T	0	0
F	F	F	0	0

Figura 25. Tabla de puntos extra en los desafíos, adaptada de [2].

4.2 Diagramas de diseño

4.2.1 Estructura de un proyecto Node (explicación adaptada a este proyecto)

La estructura de un proyecto en Node.js puede variar, ya que se pueden utilizar diferentes frameworks como plantilla. Esto permite dar distintos enfoques a los proyectos ya que cada frameworks puede ofrecer diferentes funcionalidades y comodidades y por tanto tener algo más de flexibilidad en el tipo de proyecto que se quiera realizar.

En este proyecto, se ha utilizado el framework Express como base por tanto se utilizó una plantilla que viene dada por el complemento “Express Generator”. Por tanto, todos los proyectos de este tipo tendrán una estructura por directorios igual o muy parecida a la que a continuación se va a explicar.

El proyecto se divide en estos directorios mostrados en la Figura 26:

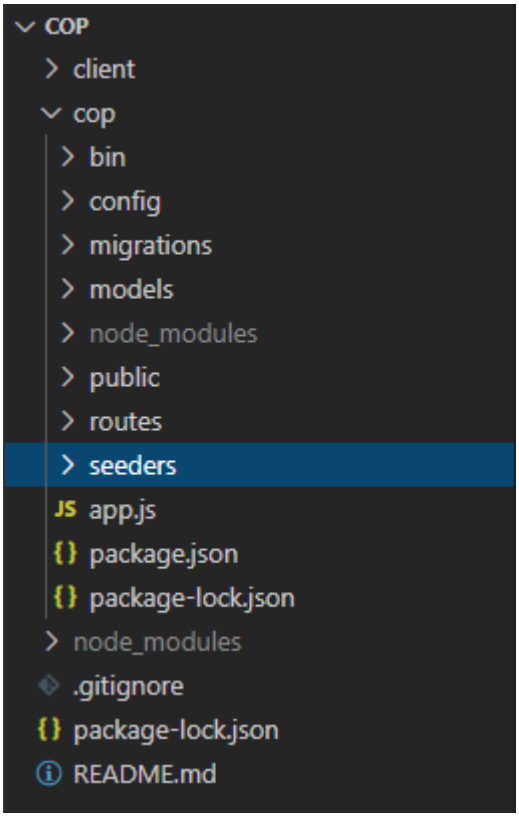

Figura 26. Clasificación por directorios del proyecto Node.js

- **Client:** En esta carpeta se encuentran los ficheros del cliente de prueba utilizados en las primeras pruebas del proyecto, esto va aparte de nuestra API.

- **Bin:** Contiene el fichero “www” el cual es el primero que se ejecuta cuando se arranca el servidor y ofrece ciertas configuraciones como el número del puerto del servidor.

- **Config:** Contiene el fichero “config” el cual recoge las configuraciones que conectan la API con la base de datos de MySQL.

- **Migrations:** Esta carpeta contiene todas las migraciones del framework “Sequelize”. Esto se podría considerar como los scripts de la base de datos.

- **Models:** En este directorio se encuentran la capa de modelo de nuestra aplicación. Cuando se invocan estos modelos en el código, permite que se realicen transacciones directas con la base de datos.

- **Node_modules:** Aquí se encuentran guardados todos los módulos (en los que se engloban frameworks y bibliotecas) que utiliza Node.js y Express. Hay que destacar que aquí se encuentran los que son nativos de Node.js y los que se añaden a mayores al proyecto.

- **Public:** Aquí se encuentra el directorio donde se guardarán todas las imágenes de los usuarios y de los logros y las cuales serán accedidas mediante una dirección.

- **Routes: Contiene la lógica de negocio del sistema (controlador).** Este es el directorio donde se concentra el trabajo del proyecto. Aquí es donde se alojan las llamadas de la API las cuales serán llamadas por un cliente externo para realizar determinadas acciones. También se guardan aquí el fichero de constantes y el fichero de funciones que sirven de apoyo a las llamadas.

- **Seeders:** Este directorio contiene los ficheros que una vez se ejecuten, permiten poblar la base de datos de forma automática. Esto es especialmente útil en producción cuando se necesitan hacer varias migraciones y entre ellas se pierden los datos guardados en la base de datos.

- **App.js:** Este fichero contiene la llamada a algunos módulos necesarios para el funcionamiento del servidor, además que aquí se declaran los nombres de las llamadas que luego utilizará el cliente para enviar peticiones.

- **Package.json:** En este fichero se encuentran el nombre y versión de los distintos módulos que el desarrollador ha incluido a mayores de los que venían. Este fichero también sirve para reinstalar todos estos módulos de forma automática en caso de ser necesario.

- **Package-lock.json:** Información completa acerca de todos los módulos que incorpora Node.js y Express además de los añadidos por el desarrollador.

- **Gitignore:** Fichero que añade Git para no guardar los módulos en el repositorio.

- **Readme.md:** Pequeño fichero de texto que resume información sobre el proyecto.

4.2.2 Modelo de la base de datos

En la Figura 27 se muestra el diagrama del modelo de la base de datos MySQL de la aplicación. Este modelo determina la estructura lógica de una base de datos y muestra de forma concentrada la manera en la que se almacenará y se relacionará la información de esta. El diagrama que se muestra es un entidad-relación en el que las diferentes tablas de la base de datos se tratan como si fueran entidades y a estas se las completa añadiendo sus respectivos atributos y relacionándolas con otras entidades creando un esquema que aporta una gran cantidad de información de forma muy resumida.

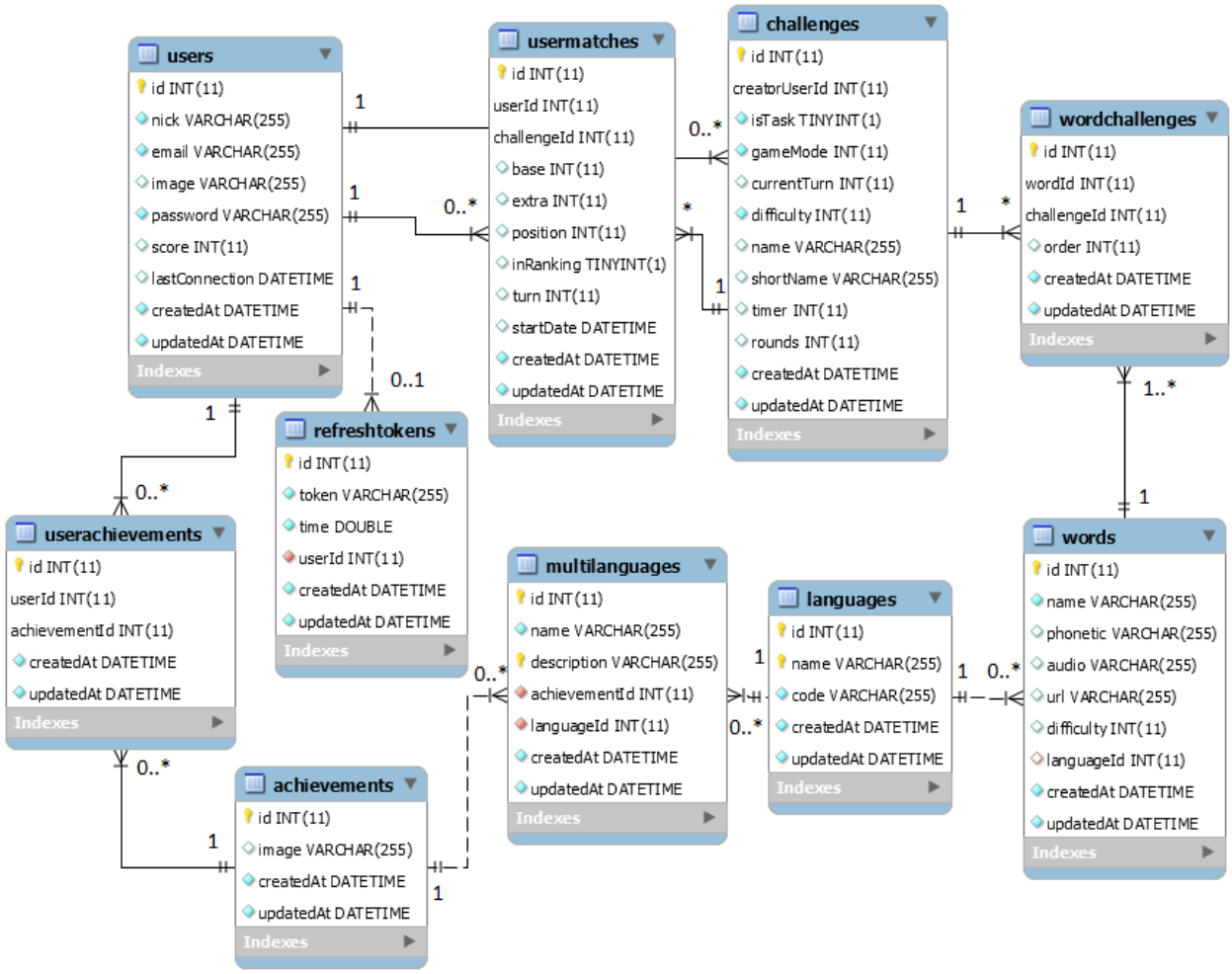

Figura 27. Modelo de la base de datos.

En el diagrama de la Figura 27 mostrado se ven las relaciones entre las tablas. La tabla ***users*** (usuarios) es la más importante del modelo, ya que almacena la información de los usuarios de la aplicación. La tabla ***refreshToken*** (token de refresco) dependen de ***users*** ya que uno de estos token tiene que estar asignado a un usuario. También salen las tablas intermedias de ***userAchievements*** (logros de un usuario) y ***userMatches*** (partida de un usuario) relacionadas con ***achievements*** (logros) y ***challenges*** (partidas) ya que se necesita de estas tablas intermedias con identificadores de ambas partes, para poder representar que un determinado usuario tiene un determinado logro o partida. La tabla ***userMatches*** también tiene atributos propios de una partida individual, por lo que se colocan ahí.

La tabla ***achievements*** está unida con la tabla intermedia ***multilanguages*** (multilenguajes) que se relaciona con la tabla ***languages*** (lenguajes) ya que se necesita identificar un determinado logro con un determinado lenguaje, de forma parecida al caso anterior, además de añadir atributos propios que no pueden ser colocados en las otras tablas.

La tabla ***challenges*** contiene la información de las partidas y está relacionada tanto con ***users*** como con ***userMatches***, debido a que se necesita la información de quién es el creador de la partida, así como la de los jugadores que participan, por lo que el atributo del creador al ser único se pone en la tabla ***challenges***, y los demás atributos propios de la partida y sus jugadores, se ponen en la tabla

intermedia. También se relaciona con la tabla ***words*** (palabras) y se hace una tabla intermedia (palabras de una partida) que contiene las palabras que serán utilizadas en la partida y por tanto para identificarlas se realizar esto.

Por último, la tabla ***words*** también se relaciona con la tabla ***languages*** para asignar un idioma determinado a todas las palabras. Como es una relación de 1 a 0 a muchos, no se necesita de una tabla intermedia en este caso.

4.2.3 Diagrama de paquetes

En la Figura 28 se muestra el diagrama de paquetes del proyecto:

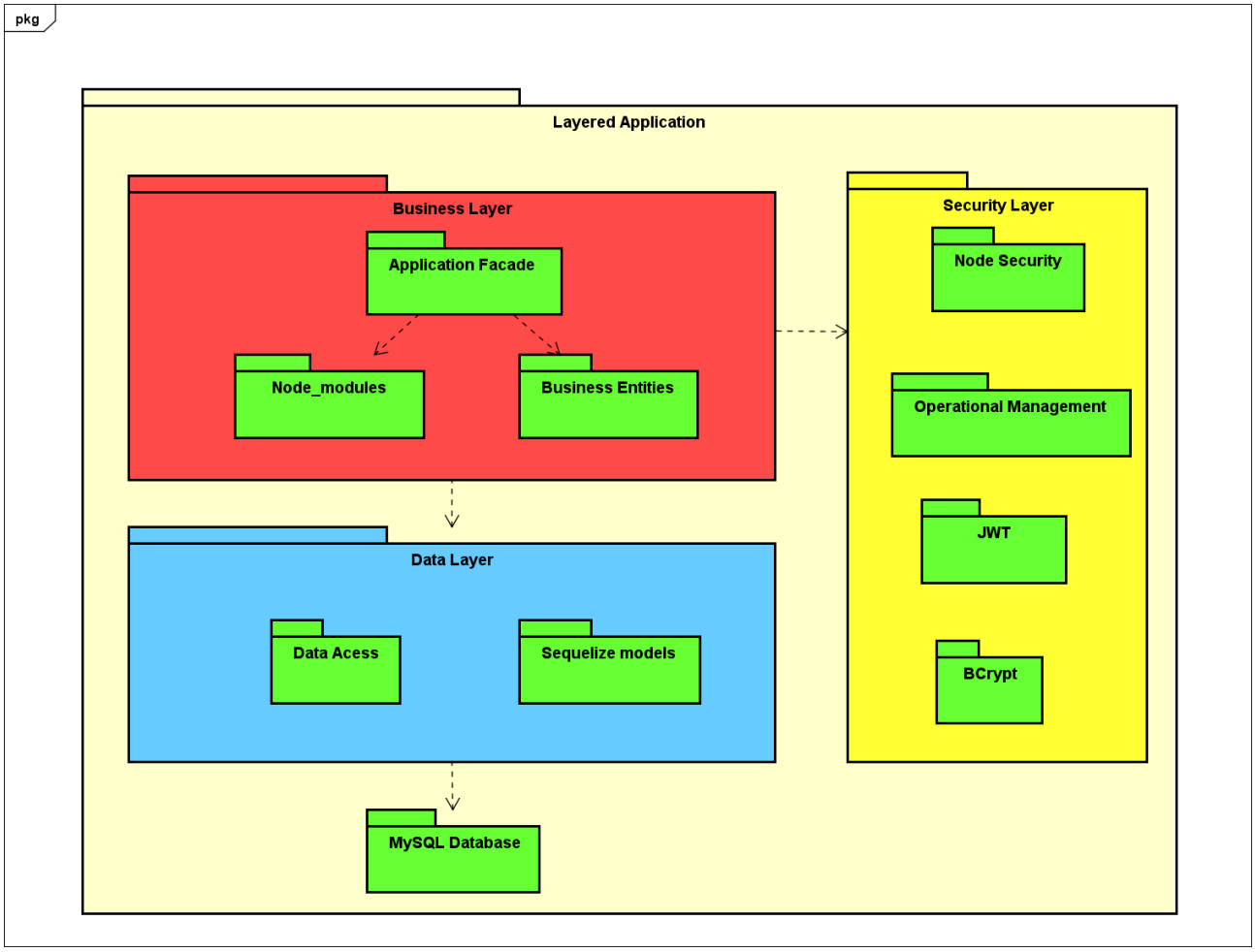

Figura 28. Diagrama de paquetes.

El proyecto se divide en tres partes principales las cuales son:

- **Business layer:** Esta es la capa de negocio o la capa del controlador. Aquí es donde se reciben las peticiones del cliente y se realizan operaciones que realizan las funcionalidades que se buscan en el proyecto a partir de los datos recibidos. Esta capa se compone a su vez:

  - **Application facade:** Es la fachada de la aplicación. Aquí es donde llegan las peticiones de los clientes y los datos de estas son redireccionados a la entidad correspondiente que se encargara de resolverlas.

  - **Node_modules:** Son los módulos que se añaden a Node.js para expandir sus funcionalidades o simplificar el código que se tiene que hacer para realizar una determinada tarea. Depende de la *Application Facade.*

  - **Business entities:** Estas son las entidades que controlan los datos recibidos y realizan operaciones con ellos a fin de poder resolver una tarea determinada y devolver una respuesta al usuario. Dependen de la *Application Facade.*

- **Data layer:** Esta es la capa de modelo o la capa de datos. Esta parte se encarga de conectar la capa de controlador con la base de datos en caso de que haya que realizar transacciones con esta. Esta capa se compone de dos elementos:

  - **Data access:** Aquí se encuentra la configuración necesaria para conectar el proyecto con la base de datos, como, por ejemplo, la dirección, el nombre, el usuario y su contraseña entre otros.

  - **Sequelize models:** Estos modelos que provienen del framework de Sequelize son como el puente intermedio entre las tuplas de la base de datos y el código. Gracias a esto, la información de la base de datos puede ser utilizada en el controlador y viceversa.

- **Security Layer:** También conocida como capa de seguridad. En esta capa se encuentran todos los elementos utilizados para mantener la mayor seguridad posible dentro del proyecto. Se compone de estos elementos:

  - **Node Security:** Es la seguridad que provee la propia plataforma de Node.js.

  - **Operational Management:** Este es el controlador que utiliza Node.js para atender a las peticiones de diferentes usuarios. Cuando llega una petición se queda en espera hasta que es resuelta, entonces se notifica al controlador de Node.js para que pueda ser respondida al cliente.

  - **JWT:** Es el estándar de firma digital que se utiliza en este proyecto para que un usuario pueda acceder a sus datos de forma individual y segura.

  - **BCrypt:** Librería de Node.js que permite codificar contraseñas y datos sensibles para garantizar su seguridad.

- **MySQL database:** Aquí se encuentra la base de datos en la que se guardan todos los datos del proyecto.

4.2.4 Diagrama de despliegue

El cliente mediante la aplicación móvil del juego se conectará al servidor de Node.js mediante una petición HTTP y por donde accede a la API CoP que está programado en JavaScript. El servidor Node.js a su vez está conectado con una base de datos MySQL gracias a JDBC.

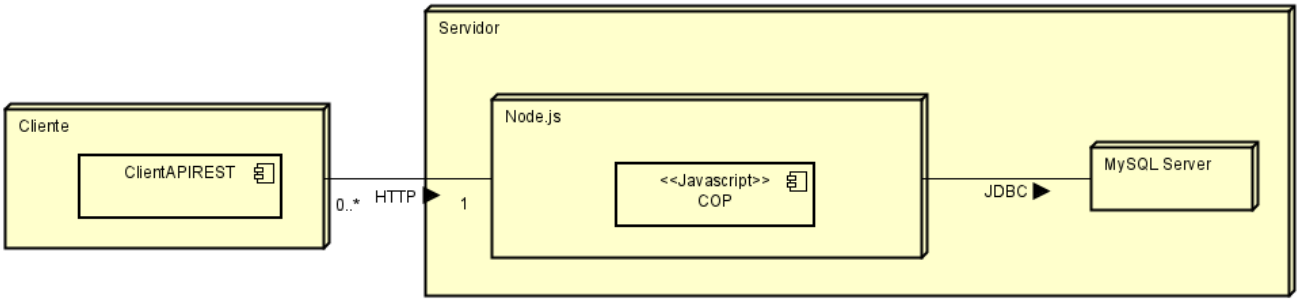

Figura 29. Diagrama de despliegue

Capítulo 5

5. Pruebas

5.1 Pruebas de caja negra

En esta sección se detallarán las pruebas de funcionamiento sobre la aplicación. Estas pruebas han sido realizadas durante todo el transcurso del proyecto y estas abordan todas las llamadas del programa.

Las pruebas realizadas siguen el modelo de pruebas de “caja negra”, en las cuales se pretende probar el cumplimiento de los requisitos definidos, sin embargo, no se espera profundizar en los detalles técnicos que hay detrás de estos, sino más bien, en realizar unas acciones y en función de ellas esperar obtener un resultado especifico. Una prueba se considera como “correcta” si recibimos el resultado esperado, pero se considerará “fallida” en el caso contrario.

Todas las pruebas se han realizado con el programa Postman, aunque al comienzo del proyecto se utilizó un cliente de prueba de Node.js utilizando el módulo React. Esto se desechó posteriormente y se utilizó Postman por la mayor comodidad y flexibilidad que ofrecía para las pruebas, así como un guardado de las distintas llamadas y no tener que volverlas a meter a mano.

En la parte superior central seleccionamos el tipo de llamada que realizaremos (en nuestro caso, una llamada POST) junto con la dirección donde se ubica la llamada en nuestro servidor. En el lado izquierdo tenemos un directorio que recopila las pruebas guardadas. En la pestaña authorization se escribe el token si es necesario mandarlo, para que se mande dentro de la cabecera siguiendo un patrón ya establecido. La pestaña que se muestra es la de body que es el json de datos de entrada que enviamos a la API y debajo de esto tenemos la respuesta que envía el servidor a la petición realizada.

En la Figura 30 se muestra la interfaz del programa Postman para ilustrar mejor cómo se han realizado estas pruebas.

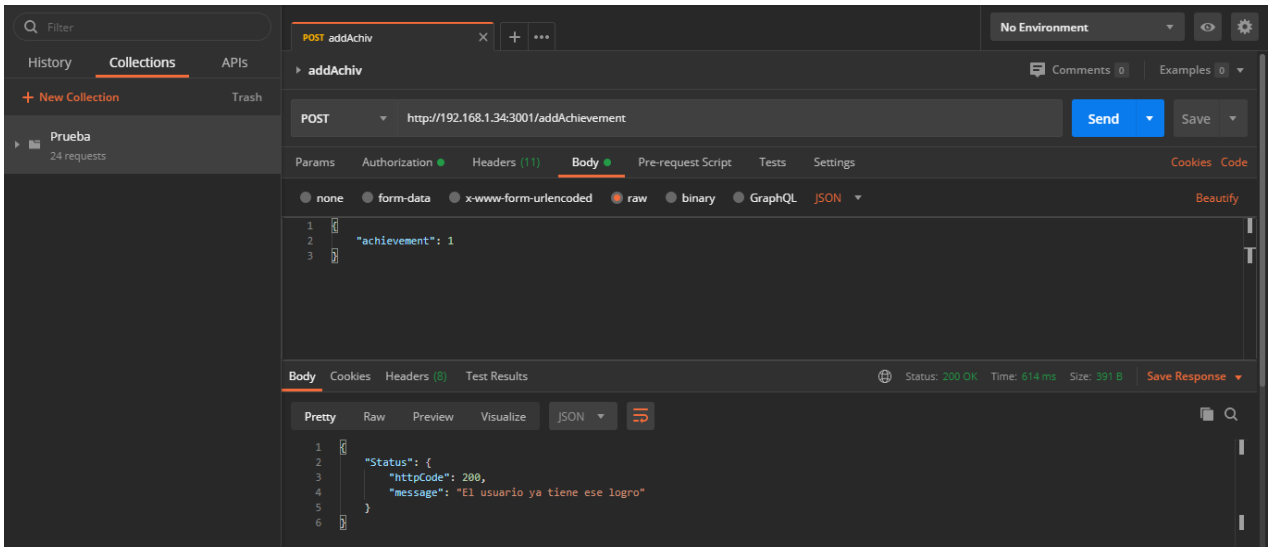

Figura 30. Imagen ilustrativa de una prueba en Postman

Las pruebas se identificarán con la letra “P” seguido por un número según el orden en el que se han realizado. Se pondrá el nombre de la llamada a la que se manda la prueba, una breve descripción, el resultado esperado y el resultado final que nos devuelve. Es importante destacar también se han hecho pruebas de los casos con respuesta incorrecta pero esperada, sin embargo no han sido incluidas en la memoria.

Identificador	P01
Nombre	Register
Descripción	Introducción de un nuevo usuario en el sistema.
Resultado esperado	Una vez introducidos un nick y username no repetidos junto con una contraseña correcta, el usuario será introducido en el sistema.
Resultado final	Correcto

Tabla 27. Descripción de la prueba P01

Identificador	P02
Nombre	Register2
Descripción	Introducción de un nuevo usuario con una imagen.
Resultado esperado	Mismo caso que en el P01, pero introduciendo una imagen codificada en base64 la cual se decodifican y se guardará.
Resultado final	Correcto

Tabla 28. Descripción de la prueba P02

Identificador	P03
Nombre	Login
Descripción	Identificación de un usuario mediante nombre y contraseña.
Resultado esperado	Identificación del usuario y respuesta con los tokens de acceso a datos y de refresco propios del usuario.
Resultado final	Correcto

Tabla 29. Descripción de la prueba P03

Identificador	P04
Nombre	UserData
Descripción	Obtención de datos del usuario.
Resultado esperado	Se devolverá una lista con todos los datos públicos del usuario.
Resultado final	Correcto

Tabla 30. Descripción de la prueba P04

Identificador	P05
Nombre	ShowAchievements
Descripción	Obtención de logros de un usuario.
Resultado esperado	Se devolverá una lista con los logros y su descripción obtenidos por un usuario.
Resultado final	Correcto

Tabla 31. Descripción de la prueba P05

Identificador	P06
Nombre	AddAchievement
Descripción	Añadir un logro a un usuario.
Resultado esperado	Se introducirá un logro existente a un usuario.
Resultado final	Correcto

Tabla 32. Descripción de la prueba P06

Identificador	P07
Nombre	Modify
Descripción	Modificación de datos de un usuario.
Resultado esperado	Se devolverá un mensaje de cambio realizado con éxito. Estos cambios se guardarán en el sistema.
Resultado final	Correcto

Tabla 33. Descripción de la prueba P07

Identificador	P08
Nombre	Modify2
Descripción	Modificación de datos de un usuario con una imagen.
Resultado esperado	Igual que en el caso P07 pero esta vez con una imagen codificada en base64.
Resultado final	Correcto

Tabla 34. Descripción de la prueba P08

Identificador	P09
Nombre	Ranking
Descripción	Clasificación por puntos de los jugadores.
Resultado esperado	Se devuelve la clasificación de los usuarios en orden descendente de puntos.
Resultado final	Correcto

Tabla 35. Descripción de la prueba P09

Identificador	P10
Nombre	Opponents
Descripción	Oponentes de un jugador.
Resultado esperado	Se buscan jugadores según los puntos del usuario que lo llama.
Resultado final	Correcto

Tabla 36. Descripción de la prueba P10

Identificador	P11
Nombre	CreateChallenge
Descripción	Creación de una partida.
Resultado esperado	Se crea una partida a partir de los oponentes buscados antes y de una lista de palabras.
Resultado final	Correcto

Tabla 37. Descripción de la prueba P11

Identificador	P12
Nombre	CreateChallenge2
Descripción	Creación de una partida con un lenguaje inexistente.
Resultado esperado	Error en la creación, salida de la llamada sin guardar nada.
Resultado final	Fallido

Tabla 38. Descripción de la prueba P12

Identificador	P13
Nombre	InfoChallenge
Descripción	Devuelve la información completa de una partida.
Resultado esperado	Información completa de una partida, incluyendo datos de esta, jugadores, palabras y turno.
Resultado final	Correcto

Tabla 39. Descripción de la prueba P13

Identificador	P14
Nombre	ListChallenge
Descripción	Devuelve todas las partidas de un usuario según fecha y si están terminadas.
Resultado esperado	Lista con todas las partidas que cumplen los requisitos.
Resultado final	Correcto

Tabla 40. Descripción de la prueba P14

Identificador	P15
Nombre	ListChallenge2
Descripción	Mismo caso que en el P14 pero introducimos la fecha mal.
Resultado esperado	Captura del error y salida del programa.
Resultado final	Fallido

Tabla 41. Descripción de la prueba P15

Identificador	P16
Nombre	RefreshToken
Descripción	Actualización del token de acceso para el usuario en caso de que se haya caducado.
Resultado esperado	Devuelve un nuevo token de acceso para el usuario.
Resultado final	Correcto

Tabla 42. Descripción de la prueba P16

Identificador	P17
Nombre	startMatch
Descripción	Un usuario comienza su turno.
Resultado esperado	Si es turno del jugador, se informará que es su turno y podrá jugar.
Resultado final	Correcto

Tabla 43. Descripción de la prueba P17

Identificador	P18
Nombre	finishMatch
Descripción	El usuario juega su turno
Resultado esperado	Si es turno del jugador, este lo jugará y se devolverá información sobre la partida.
Resultado final	Correcto

Tabla 44. Descripción de la prueba P18

Identificador	P19
Nombre	finishMatch2
Descripción	El usuario juega su turno y es el último
Resultado esperado	Si es turno del jugador, este lo jugará y se informará de que la partida terminó y se actualizarán todas las puntuaciones de todos los jugadores.
Resultado final	Correcto

Tabla 45. Descripción de la prueba P19

Capítulo 6

6. Conclusiones

En este capítulo se recogen las conclusiones a partir del desarrollo de este proyecto, así como ideas de cara al futuro que han surgido una vez finalizado el mismo. Lo que primero se ha de resaltar del proyecto es que es factible realizar una API REST que funcione como un servidor. Todo esto realizado por un solo desarrollador novato como TFG en el que el peso del proyecto no es excesivo y de una complejidad asequible, así como a la vez cumpliendo los objetivos y requisitos propuestos para que el juego pueda funcionar correctamente. Este proyecto se encuentra actualmente pendiente de subir a producción en el grupo de investigación reconocido ECA-SIMM.

En segundo lugar, a nivel personal del estudiante a cargo del desarrollo se ha podido sumergir en un proyecto real que le acerca a tener una experiencia más real muy parecida a lo que se hace en el mundo laboral, aportando así una toma de contacto antes de comenzar a trabajar en una empresa. Además, que ha servido para ganar experiencia y conocimientos sobre cómo crear una API REST y con las herramientas mayormente utilizadas como son Node.js, JavaScript y Postman. Gracias a este estudio se ha podido realizar un proyecto muy completo que aporta la funcionalidad necesaria al juego CoP que antes proporcionaba Google Play Games.

Por último, el proyecto también ha servido para aprender como comprender y seguir una planificación para llevar a cabo un proyecto de longitud media. Además, se ha aprendido también a estimar tiempos, riesgos y costes, así como planificar una elaboración de pruebas. Todo ello ha aportado experiencia que será útil en proyectos personales o laborales en el futuro.

6.1 Trabajo futuro

A raíz de las pruebas realizadas durante el proyecto y de sugerencias que se han ido aportando, se podría añadir algunas mejoras adicionales al proyecto para mejorar su funcionamiento o aportar una mayor calidad al usuario. Algunas de estas posibles mejoras son:

- **Mayor seguridad:** Se podría invertir más tiempo en aumentar la seguridad del proyecto, protegiendo sobre todo las contraseñas de los usuarios y de los tokens de acceso y refresco, además de bloquear posibles accesos externos en caso de que alguien robe información del sistema además de prevenir ataques de fuerza bruta a la autenticación.

- **Aumentar las opciones del jugador:** Por ejemplo, se podría invertir tiempo en crear nuevas funcionalidades para el usuario, como poder registrar amigos, poder jugar con ellos libremente, jugar partidas libres que no modifiquen la puntuación o incluso crear un sistema de agrupación

por clanes en el que un grupo de usuarios compitan por obtener una mayor puntuación global que el resto de los clanes habiendo una clasificación propia para esta modalidad.

- **Mejorar la eficiencia y escalabilidad del proyecto:** En este caso, debido a la inexperiencia del desarrollador en proyectos de este tipo, algunos detalles se pudieron haber implementado mejor y con más claridad. En un futuro en el que se cuente con más experiencia, se podría mejorar el proyecto en estos ámbitos secundarios:

    - Mejorar la sintaxis del código. Por ejemplo: utilizar *async await* en vez de promesas que las llaves provocan que el código.
    - Reducir el número de bucles necesarios para llevar a cabo una acción concreta.
    - Buscar si hay módulos que puedan realizar ciertas funcionalidades de manera más sencilla a la presentada.
    - Reordenar el código para que sea más fácil poder leerlo.

Apéndices

A. Acrónimos

- **API:** Application Programming Interface.
- **CD:** Compact Disc.
- **CoP:** Clash of Pronunciation.
- **CPU:** Central Processing Unit.
- **ECA-SIMM:** Grupo de investigación de entornos de computación.
- **HU:** Historia de usuario.
- **JSON:** JavaScript Object Notation.
- **JWT:** Json Web Token.
- **RAM:** Random Access Memory.
- **RF:** Requisito Funcional.
- **RFN:** Requisito No Funcional.
- **REST:** REpresentational State Transfer.
- **TFG:** Trabajo de fin de grado.
- **UVa:** Universidad de Valladolid.

B. Manual de instalación

En este apartado de la memoria se explicarán los requerimientos y procedimientos necesarios para instalar el proyecto en una computadora. En este manual se explicarán los pasos en el caso de utilizar un sistema operativo Windows 10, algunos pasos podrían variar para otros sistemas.

- **Instalación de la terminal de trabajo y clonación del repositorio:**
    - Se instala git desde el link *https://git-scm.com/download/win*
    - Se aceptan las opciones que nos vienen por defecto.
    - Situarse en el directorio en el que queremos clonar el repositorio del proyecto.
    - Abrir Git Bash con click derecho en un espacio vacío.
    - Se pone el comando *git clone https://gitlab.inf.uva.es/anddela/cop.git* el cual copiará el repositorio en el directorio elegido. Esta dirección es donde está alojado el repositorio en la nube.
- **Instalación de node y módulos necesarios:**
    - Se descarga y se instala Node.js desde esta dirección *https://nodejs.org/es/download/*
    - Se aceptan los términos e instalamos con las opciones por defecto.
    - Se comprueba que se ha instalado correctamente poniendo el comando node -v y npm -v y si nos devuelve la versión es que está correctamente instalado.
    - En el terminal donde está el proyecto, se instalan todos los módulos necesarios para el funcionamiento del proyecto utilizando los comando *npm i* y *npm install -g sequelize-cli* . Esto hará que se instalen todos los módulos de forma automática gracias al fichero package.json.
- **Instalación y configuración de la base de datos:**
    - Se descarga e instala el programa MySQL Installer Community de esta página *https://dev.mysql.com/downloads/installer/* . También se pueden utilizar otros gestores de bases de datos como Wamp o Xamp pero las configuraciones de estas serían las mismas que la primera mencionada.
    - Una vez en el instalador, se instala la versión de desarrollador, se deja que se instale todo y nos aseguramos de crear el usuario “root” con contraseña “1234”.
    - Cuando termine todo de instalarse, en el MySQL Workbench, se crea una conexión de esta forma:

- Se selecciona el “+” para añadir una nueva conexión y se pone de nombre base de datos como “mydb”.
- El hostname tiene que ser 127.0.0.1.
- Puerto 3306.
- Usuario “root”.
- Y contraseña “1234”.

- Una vez añadida la conexión, hay que dirigirse a la consola de MySQL WorkBench y se añade este comando *ALTER USER 'root'@'localhost' IDENTIFIED WITH mysql_native_password BY '1234' .*

- **Migración de la base de datos:**
  - Se vuelve a la terminal en el directorio donde se encuentra el proyecto y se introduce este comando *sequelize db:create .* Con esto se creará la estructura para las migraciones y el modelo del proyecto.
  - Después se ejecuta este comando *sequelize db:migrate* y se migrarán todas las tablas a la base de datos.

- **Arranque del servidor:**
  - Para arrancar el servidor, se tiene que situar en el directorio donde se aloje este, abrir la terminal allí e introducir este comando *npm start.* Si todo está bien, el servidor debería arrancar con este mensaje de la Figura 31.

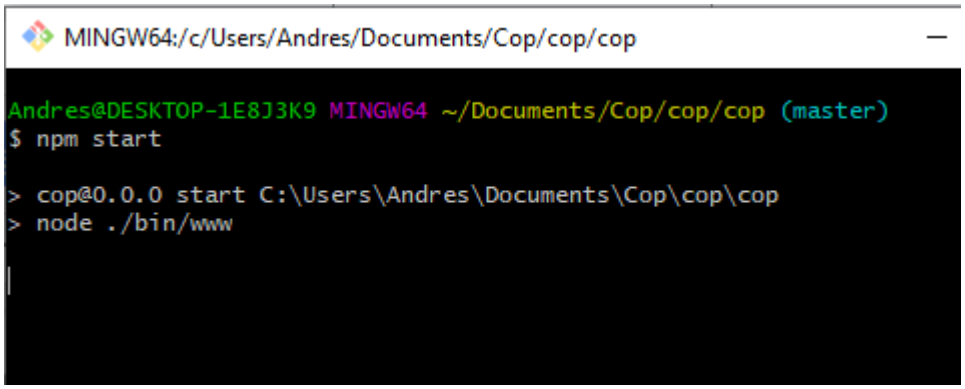

Figura 31. Imagen del arranque del servidor.

- **Instalación del programa de pruebas:**
  - Se descarga el programa Postman de la página https://www.postman.com/downloads/ e iniciamos sesión.

- Para preparar la prueba, se necesita ver primero nuestra IP, para esto vamos se da clic en el símbolo de red de windows y se entra a las propiedades de la red que estemos utilizando.

- Una vez allí, la IP que se necesita es la que viene en el campo IPv4.

- Después se preparan las pruebas, un ejemplo de una seria *http://192.168.1.35:3001/createChallenge.*

C. Manual de despliegue

En esta sección se explicarán los pasos efectuados a la hora de desplegar el proyecto en un servidor. Se considerará que es una maquina virtual con un sistema operativo Linux y que por tanto tendremos a disposición desde el principio de una terminal de trabajo. En caso de que el servidor tenga un sistema operativo que utilice Windows, se efectuará de la misma manera, salvo en que se necesita descargar la herramienta “Git for Windows” para poder utilizar una terminal de trabajo.

- **Instalación de Node.js:**
    - Al estar utilizando una terminal, lo que necesitaremos hacer es descargar el instalador utilizando el comando *cp https://nodejs.org/dist/v12.18.3/node-v12.18.3-linux-x64.tar.xz.*
    - Una vez realizado esto, extraemos los archivos del instalador utilizando el comando *tar https://nodejs.org/dist/v12.18.3/node-v12.18.3-linux-x64.tar.xz.*
    - Una vez extraídos los ficheros, procedemos a entrar en el directorio resultante y escribimos el comando *sudo make install* para instalar Node.js.
    - Una vez instalado, procedemos a revisar que se instalo correctamente usando el comando *node -v* y *npm-v* y si devuelve el numero de la versión, es que esta correctamente instalado.
    - Procedemos a instalar todos los módulos tal cual viene especificado en el manual de instalación usando los comandos *npm i* y *npm install -g sequelize-cli.*

- **Clonando el repositorio de Git:**
    - Al igual que en el manual de instalación, necesitamos traer nuestro proyecto al servidor, por lo que primero instalamos Git para poder clonar nuestro repositorio, por eso usamos el comando *sudo apt-get install git.*
    - Una vez hecho esto, se clona el repositorio en el directorio que queramos utilizando este comando *git clone https://gitlab.inf.uva.es/anddela/cop.git.* De esta manera tendremos el proyecto en el servidor.

- **Preparar la aplicación para despliegue:**
    - Una vez que tengamos lo anterior, es necesario configurar el proyecto al estado de producción, para eso tenemos que situarnos dentro del proyecto y utilizar este comando *export NODE_ENV=production.*

- **Instalar y preparar la base de datos:**

    - Lo que primero se tiene que hacer es descargar e instalar un servidor de MySQL, para ello se utiliza este comando para poder instalarlo directamente *sudo apt-get install mysql-server.*

    - Una vez instalado, se inicializa utilizando el comando *sudo systemctl start mysql.*

    - En principio se utilizará la dirección por defecto que viene dada en la instalación (127.0.0.1) por lo que no habrá que configurar nada en este aspecto. Lo que, si se tiene que configurar es el nombre de usuario y la contraseña, que se hará usando este comando *UPDATE mysql.user SET authentication_string = PASSWORD('password') WHERE User = 'user';* en el cual sustituiremos la palabra “user” y “password” por el usuario y contraseña que se quiera.

    - Creamos una nueva base de datos poniendo este comando *CREATE DATABASE mydb;* ya que tiene que coincidir con el mismo nombre que el usado en el proyecto.

    - Una vez creada y estando dentro de la nueva base de datos, copiaremos el script del proyecto para generar las tablas que necesitamos.

- **Arranque del servidor:**

    - Una vez realizado con éxito todo lo anterior, solo queda arrancar el servidor con el comando *npm start* y ya estará el proyecto desplegado y listo para recibir peticiones.

D. API

Documentación de la API

1. Register

Se crea un nuevo usuario en el sistema.

URL

POST http://IP:3001/register

JSON

{

username: String

email: String

pass: String

image: String

formatImage: String (.jpg,.jpeg,png)

}

Campos

username: El nombre del usuario en el juego.

email: Correo electrónico registrado del usuario.

pass: Contraseña del usuario.

image: Imagen codificada en base64 del usuario.

formatImage: Formato de la imagen. Tres posibles opciones jpg, jpeg, png

Respuesta

{

Status: JSON{

httpCode: int

message: String

}}

Campos

Status JSON: JSON con todos los datos referidos al resultado de la operación
httpCode: Código numérico del http
message: Mensaje que detalla el resultado de la operación.

2. Login

Permite al usuario identificarse con su cuenta en el juego.

URL

POST http://IP:3001/login

JSON

{

usernameLogin: String

passwordLogin: String

}

Campos

usernameLogin: Nombre del usuario en el sistema con el que se identifica.

passwordLogin: Contraseña del usuario necesaria para identificarse.

Respuesta

{

Data: JSON Array

Authorization: Bearer + token(String)

refreshAuthorization: Bearer + token(String)

expirationToken: int

expirationRefreshToken: int

Status: JSON

httpCode: int

message: String

}

Campos

Status JSON: JSON con todos los datos referidos al resultado de la operación

httpCode: Código numérico del http

message: Mensaje que detalla el resultado de la operación.

token: Cadena de caracteres encriptada que permite al sistema tener constancia del usuario conectado.

refreshToken: Token de refresco propio del usuario que el sistema utilizará para comprobar si la sesión sigue activa

expirationToken: Tiempo en horas en el que el token estará activo

expirationRefreshToken: Tiempo en horas en el que el token de refresco estará activo

3. UserData

El sistema devuelve los datos del perfil de un usuario identificado.

URL

POST **http:///IP:3001/userData**

Headers

{

Authorization: Bearer + token (String)

}

Campos:

Authorization: Token de acceso del usuario

JSON

{

}

Respuesta

{

Data: JSON

nick: String

email: String

image: String

Status: JSON

httpCode: int

message: String

}

Campos

Status JSON: JSON con todos los datos referidos al resultado de la operación

httpCode: Código numérico del http

message: Mensaje que detalla el resultado de la operación.

Data:

nick: Nombre del usuario en el sistema.

email: Correo electrónico del usuario en el sistema.

image: Dirección absoluta al alojamiento de la imagen del usuario.

4. Modify

El sistema permite modificar los parámetros del perfil de un usuario.

URL

POST http://IP:3001/modify

Headers

{
Authorization: Bearer + token(String)
}

Campos:

Authorization: Token de acceso del usuario

JSON

{

username: String

email: String

pass: String

image: String

formatImage: String (.jpg,.jpeg,png)

}

Campos

username: El nombre del usuario en el juego.

email: Correo electrónico registrado del usuario.

pass: Contraseña del usuario.

image: Imagen codificada en base64 del usuario.

formatImage: Formato de la imagen. Tres posibles opciones jpg, jpeg, png

Respuesta

{

Status: JSON

httpCode: int

message: String

}

Campos

Status JSON: JSON con todos los datos referidos al resultado de la operación

httpCode: Código numérico del http

message: Mensaje que detalla el resultado de la operación.

5. Ranking

El sistema permite ver al jugador una clasificación por puntos de los jugadores

URL

POST http://IP:3001/ranking

Headers:

{

Authorization: Bearer + token (String)

}

Campos:

Authorization: Token de acceso del usuario

JSON

{

}

Respuesta

{

Data: JSON Array

nick: String

score: int

Status JSON

httpCode: int

message: String
}

Campos

nick: Nombre de un usuario registrado

score: Puntuación del jugador

Status JSON: JSON con todos los datos referidos al resultado de la operación

httpCode: Código numérico del http

message: Mensaje que detalla el resultado de la operación.

6. ShowAchievements

El sistema muestra los logros personales del jugador.

URL

POST http://IP:3001/showAchievements

Headers

{

Authorization: Bearer + token (String)

}

Campos:

Authorization: Token de acceso del usuario

JSON

{

}

Respuesta

{

Data: JSON array

achivementId: int

name: String

description: String

image: String

Status JSON

httpCode: int

message: String

}

Campos

achivementId: Identificador del logro

name: Nombre del logro

description: Texto descriptivo sobre el logro

image: Texto en base64 que sirve para reconstruir

Status JSON: JSON array con todos los datos referidos al resultado de la operación

httpCode: Código numérico del http

message: Mensaje que detalla el resultado de la operación.

7. AddAchievement

El sistema asocia un logro a un usuario

URL

POST http://IP:3001/addAchievement

Headers

{
Authorization: Bearer + token (String)
}

Campos:

Authorization: Token de acceso del usuario

JSON

{
achievement: int
}

Campos

achievement: Número del logro a asignar al usuario

Respuesta

{
Status: JSON

httpCode: int

message: String
}

Campos

Status JSON: JSON con todos los datos referidos al resultado de la operación

httpCode: Código numérico del http

message: Mensaje que detalla el resultado de la operación.

8. RefreshToken

El sistema renueva el token del usuario a partir de un token de refresco otorgado

URL

POST http://IP:3001/refreshToken

JSON

{

refreshToken: String

}

Campos

refreshToken: Token de refresco propio del usuario que el sistema utilizará para comprobar si la sesión sigue activa

Respuesta

{

Status: JSON

httpCode: int

message: String

Data: JSON

refreshToken: refreshToken

}

Campos

refreshToken: Nuevo token de acceso que el usuario usará a partir de ahora.

Status JSON: JSON con todos los datos referidos al resultado de la operación

httpCode: Código numérico del http

message: Mensaje que detalla el resultado de la operación.

9. Opponents

El sistema busca oponentes para el usuario, basado en sus puntuaciones y su actividad. De manera ascendente a la puntuación por encima, y de manera descendente para posiciones por debajo.

URL

POST http://IP:3001/opponents

Headers

{
Authorization: Bearer + token (String)
}

Campos

Authorization: Token de acceso del usuario

JSON

{

}

Respuesta

{

Status: JSON

httpCode: int

message: String

Lista de oponentes: JSON array

id: string

nick: String

image: String

score: int

Number: int

}

Campos

Status JSON: JSON con todos los datos referidos al resultado de la operación

httpCode: Código numérico del http

message: Mensaje que detalla el resultado de la operación.

Lista de oponentes: Lista de los próximos oponentes del usuario

id: Identificador propio del usuario

nick: Nombre del jugador con el que se registró en el sistema

image: Ruta donde está alojada la imagen de perfil del usuario

score: Puntuación actual del usuario en ese momento

Number: Número de oponentes de la sala

Reason: Explicación del número de jugadores de la sala

10. CreateChallenge

A partir de la lista de oponentes y del propio usuario, el sistema creará una nueva partida.

URL

POST http://IP:3001/createChallenge

Headers

{
Authorization: Bearer + token (String)
}

Campos

Authorization: Token de acceso del usuario

JSON

{

isTask: boolean (int en la base de datos, 0 o 1)

gameMode: int

difficulty: int

currentTurn: int (0 paralelo o 1 secuencial)

opponents: JSON Array

id: int

score: int

words:

id: String

name: String

short_name: String

timer: int

rounds: int

pairs: JSON

words: JSON Array

word: String

url: String

trans: String

lang: String

}

Campos

isTask: Indica si es una tarea o no (Boolean 0 o 1 en la base de datos).

gameMode: Modo de juego de la partida.

difficulty: Dificultad que tiene la partida.

currentTurn: Muestra de que jugador es el turno, o de todos si se juega en paralelo (0 o 1)

opponents: Lista de jugadores que tendrá la partida (JSON ARRAY).

id: Identificador del jugador

score: Puntuación actual del jugador

words: Lista con todos las agrupaciones y palabras (JSON ARRAY)

pairs: Agrupaciones de palabras

word: Nombre de la palabra

url: dirección de la imagen de la palabra

trans: Traducción fonética de la palabra

lang: Código BCP-47 Lenguaje en el que está la palabra

estos códigos se encuentran en https://tools.ietf.org/html/bcp47

Respuesta

{

Status: JSON

httpCode: int

message: String

Data:

challengeId: int

}

Campos

Status: JSON array con todos los datos referidos al resultado de la operación

httpCode: Código numérico del http

message: Mensaje que detalla el resultado de la operación.

challengeId: Identificador de la partida que se ha creado.

11. InfoChallenge

Muestra información sobre un reto determinado y sus partidas.

URL

POST http://IP:3001/infoChallenge

Headers

{
Authorization: Bearer + token (String)
}

Campos

Authorization: Token de acceso del usuario

JSON

{
challengeId: int
}

Campos

challengeId: Identificador de la partida

Respuesta

{

Status: JSON

httpCode: int

message: String

Challenge: JSON Array

id: int

isTask: int

gameMode: int

currentTurn: int

difficulty: int

creatorUserId: int

UserMatch: JSON Array

id: int

base: int

extra: int

position: int

turn: int

challengeId: int

userId: int

words: JSON Array

id: int

name: String

phonetic: String

audio: String

url: String

difficulty: int

language: String

Turn: int [ ]

}

Campos

Status JSON: JSON con todos los datos referidos al resultado de la operación

httpCode: Código numérico del http

message: Mensaje que detalla el resultado de la operación.

challenge: Toda la información acerca de la partida

id: Identificador de la partida

isTask:

gameMode: Modo de juego de la partida

currentTurn: Turno actual de un jugador

difficulty: Dificultad de la partida

creatorUserId: Identificador del creador de la partida

userMatch: Toda la información acerca de los jugadores de una partida

id: Identificador de jugador en esa partida

base: Puntuación obtenida en esa partida por ese jugador

extra: Puntuación adicional que puede ganar un jugador

position: Posición en el ranking de un jugador al momento de crear la partida

turn: El número que indica el orden en el que juega en esa partida

challengeId: Identificador de la partida

userId: Identificador del jugador

words: Toda la información sobre las palabras de una partida

id: Identificador de la palabra

name: Nombre de la palabra

phonetic: Pronunciación fonética de la palabra

audio: Dirección donde se encuentra el audio con la pronunciación

url: Dirección donde se encuentra la imagen de la palabra

difficulty: Dificultad de la palabra

language: Idioma en el que está la palabra

order: Indica el emparejamiento de las palabras

Turn: indica de quién o quiénes es el turno (id).

12. ListChallenges

Muestra la información de una lista de desafíos y las partidas de cada jugador.

URL

POST http://IP:3001/listChallenges

Headers

{
Authorization: Bearer + token (String)
}

Campos:

Authorization: Token de acceso del usuario

JSON

{

ended: Boolean

date: Date ()

}

Campos

ended: Variable para diferenciar si se quieren partidas terminadas o todas

date: Fecha opcional, se seleccionarán partidas desde esta fecha en adelante
Formato: ‘yyyy-mm-dd’. Poner null si no se necesita

Respuesta

{

Status JSON: JSON con todos los datos referidos al resultado de la operación.

httpCode: Código numérico del http.

message: Mensaje que detalla el resultado de la operación.

Challenge: JSON Array

id: int

isTask: int

gameMode: int

currentTurn: int

difficulty: int

creatorUserId: int

UserMatch: JSON Array

id: int

base: int

extra: int

position: int

turn: int

challengeId: int

userId: int

}

Campos

Status: JSON con todos los datos referidos al resultado de la operación

httpCode: Código numérico del http

message: Mensaje que detalla el resultado de la operación.

challenge: Toda la información acerca de la partida

id: Identificador de la partida

isTask:

gameMode: Modo de juego de la partida

currentTurn: Turno actual de un jugador

difficulty: Dificultad de la partida

creatorUserId: Identificador del creador de la partida

userMatch: Toda la información acerca de los jugadores de una partida

id: Identificador de jugador en esa partida

base: Puntuación obtenida en esa partida por ese jugador

extra: Puntuación adicional que puede ganar un jugador

position: Posición en el ranking de un jugador al momento de crear la partida

turn: El número que indica el orden en el que juega en esa partida

challengeId: Identificador de la partida

userId: Identificador del jugador

13. StartMatch

El sistema permite comenzar la partida de un desafío a un jugador

URL

POST http://IP:3001/startMatch

Headers

{

Authorization: Bearer + token (String)

}

Campos:

Authorization: Token de acceso del usuario

JSON

{

challengeId: int

}

Campos

challengeId: Identificador de la partida

Respuesta

{

Status: JSON con todos los datos referidos al resultado de la operación

httpCode: Código numérico del http

message: Mensaje que detalla el resultado de la operación.

play: Boolean

}

Campos

Status: JSON con todos los datos referidos al resultado de la operación

httpCode: Código numérico del http

message: Mensaje que detalla el resultado de la operación.

play: Muestra si el jugador puede jugar su turno o no

14. FinishMatch

El sistema añade la puntuación del jugador y termina su turno.

URL

POST http://IP:3001/finishMatch

Headers

{

Authorization: Bearer + token (String)

}

Campos:

Authorization: Token de acceso del usuario

JSON

{

challengeId: int

base: int

}

Campos

challengeId: Identificador de la partida

base: Puntuación obtenida por el jugador

Respuesta

{

play: Boolean

reason (opcional): String

info (opcional): String

}

Campos

play: Indica si es turno del jugador o no

reason (opcional): Motivo por el cual el jugador no puede jugar (no es su turno)

info (opcional): Información sobre la actualización de los puntos

E. Contenido del fichero adjunto

El contenido del fichero adjunto contiene estos elementos:

Directorio raíz

MemoriaTFGAndresDeLaMazaValles.pdf: Documento con la memoria del proyecto en formato PDF

cop.zip: Incluye el código de la API REST desarrollado durante el proyecto descrito en esta memoria

pruebas.postman_collection: Fichero con la colección de pruebas realizadas.

Bibliografía

[1] Rafael Sillero Navajas. Desarrollo de la modalidad multijugador para la aplicación Clash of Pronunciation. Julio 2017. Trabajo Fin de Grado, Departamento de Informática, Universidad de Valladolid. Último acceso: 29/07/2020. [Online]. Disponible: https://uvadoc.uva.es/handle/10324/27645

[2] 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," in IEEE Access, vol. 8, pp. 74250-74266, 2020, doi: 10.1109/ACCESS.2020.2988406

[3] Servidor de Google Play Games deja inoperativo el multijugador. Último acceso: 29/07/2020. [Online]. Disponible: https://9to5google.com/2019/09/16/google-play-games-services-shutting-down-multiplayer-api/

[4] REST API Tutorial. Último acceso: 29/07/2020. [Online]. Disponible: https://restfulapi.net/

[5] API. Último acceso: 29/07/2020. [Online]. Disponible: https://www.redhat.com/es/topics/api/what-are-application-programming-interfaces

[6] API REST squema. Último acceso 29/07/2020. [Online]. Disponible: https://www.webempresa.com/blog/rest-api-de-wordpress.html

[7] REST Último acceso: 29/07/2020. [Online]. Disponible: https://desarrolloweb.com/articulos/que-es-rest-caracteristicas-sistemas.html

[8] API REST. Último acceso: 29/07/2020. [Online]. Disponible: https://bbvaopen4u.com/es/actualidad/api-rest-que-es-y-cuales-son-sus-ventajas-en-el-desarrollo-de-proyectos

[9] Google Play Games. Último acceso: 29/07/2020. [Online]. Disponible: https://www.xatakandroid.com/juegos-android/google-play-games-a-fondo

[10] Open Match. Último acceso: 29/07/2020. [Online]. Disponible: https://cloud.google.com/blog/products/open-source/open-match-flexible-and-extensible-matchmaking-for-games

[11] Firebase RealTime Database. Último acceso: 29/07/2020. [Online]. Disponible: https://firebase.google.com/docs/database

[12] Steamworks. Último acceso: 29/07/2020. [Online]. Disponible: https://partner.steamgames.com/

[13] Node.js. Último acceso: 29/07/2020. [Online]. Disponible: https://www.tutorialspoint.com/nodejs/nodejs_introduction.htm#:~:text=Node.,that%20run%20across%20distributed%20devices.

[14] Node.js. Último acceso: 29/07/2020. [Online]. Disponible: https://www.toptal.com/nodejs/why-the-hell-would-i-use-node-js

[15] Node.js imagen. Último acceso: 29/07/2020. [Online]. Disponible: https://bbvaopen4u.com/en/actualidad/mean-full-stack-javascript-development-quartet-web-apps

[16] Demanda de Node.js. Último acceso: 29/07/2020. [Online]. Disponible: https://www.monterail.com/blog/nodejs-developers-in-demand

[17] Frameworks, libraries and tools. Último acceso: 29/07/2020. [Online]. Disponible: https://www.bls.gov/ooh/computer-and-information-technology/web-developers.htm

[18] Grandes compañías utilizando Node.js. Último acceso: 29/07/2020. [Online]. Disponible: https://www.tothenew.com/blog/how-are-10-global-companies-using-node-js-in-production/#:~:text=As%20Node.,online%20games%20and%20collaboration%20tools.

[19] Netflix. Último acceso: 29/07/2020. [Online]. Disponible: https://www.netflix.com/es/

[20] Linkedin. Último acceso: 29/07/2020. [Online]. Disponible: https://es.linkedin.com/?trk=guest_homepage-basic_nav-header-logo

[21] Trello. Último acceso: 29/07/2020. [Online]. Disponible: https://trello.com/es

[22] PayPal. Último acceso: 29/07/2020. [Online]. Disponible: https://www.paypal.com/es/home

[23] Postman. Último acceso: 29/07/2020. [Online]. Disponible: https://www.postman.com/api-platform/?utm_source=www&utm_medium=home_hero&utm_campaign=button

[24] Json Web Token. Último acceso: 29/07/2020. [Online]. Disponible: https://jwt.io/introduction/

[25] Imagen de metodología iterativa e incremental. Último acceso: 29/07/2020. [Online]. Disponible: http://www.binaryspark.com/classes/studio20_Class_4.pdf

[26] Telegram. Último acceso: 29/07/2020. [Online]. Disponible: https://telegram.org/

[27] Google Drive. Último acceso: 29/07/2020. [Online]. Disponible: https://www.google.com/intl/es_ALL/drive/

[28] Skype. Último acceso: 29/07/2020. [Online]. Disponible: https://www.skype.com/es/

[29] Visual Studio Code. Último acceso: 29/07/2020. [Online]. Disponible: https://code.visualstudio.com/docs

[30] Git for Windows. Último acceso: 29/07/2020. [Online]. Disponible: https://github.com/git-for-windows/git/wiki/FAQ

[31] MySQL. Último acceso: 29/07/2020. [Online]. Disponible: https://dev.mysql.com/doc/

[32] MySQL Workbench. Último acceso: 29/07/2020. [Online]. Disponible: https://www.mysql.com/products/workbench/

[33] Npm. Último acceso: 29/07/2020. [Online]. Disponible: https://docs.npmjs.com/about-npm/

[34] Gitlab. Último acceso: 29/07/2020. [Online]. Disponible: https://about.gitlab.com/

[35] Express. Último acceso: 29/07/2020. [Online]. Disponible: https://expressjs.com/es/

[36] Sequelize. Último acceso: 29/07/2020. [Online]. Disponible: https://sequelize.org/master/index.html

[37] Router. Último acceso: 29/07/2020. [Online]. Disponible: https://www.npmjs.com/package/router

[38] MySQL (módulo). Último acceso: 29/07/2020. [Online]. Disponible: https://www.npmjs.com/package/mysql

[39] Bcrypt. Último acceso: 29/07/2020. [Online]. Disponible:.https://www.npmjs.com/package/bcrypt

[40] Is-Base-64. Último acceso: 29/07/2020. [Online]. Disponible: https://www.npmjs.com/package/is-base64

[41] Body-Parser. Último acceso: 29/07/2020. [Online]. Disponible: https://www.npmjs.com/package/body-parser

[42] Validator. Último acceso: 29/07/2020. [Online]. Disponible: https://www.npmjs.com/package/validator

[43] Estudio de remuneración de Michael Page. Último acceso: 29/07/2020. [Online]. Disponible: https://www.michaelpage.es/sites/michaelpage.es/files/MP_SPA_ON_ER_IT_03052017.pdf