Universidad de **Valladolid**

Escuela de Ingeniería Informática

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática
Mención en Tecnologías de la Información

Propuesta de matchmaking para aplicaciones educativas multijugador online

Autor:
Mario Panadero Palazuelos

Universidad de **Valladolid**

Escuela de Ingeniería Informática

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática

Mención en Tecnologías de la Información

Propuesta de matchmaking para aplicaciones educativas multijugador online

Autor:

Mario Panadero Palazuelos

Tutores:

Mario Corrales Astorgano

Cristian Tejedor García

A mi familia y amigos por ser parte también de esta gran etapa

Agradecimientos

En primer lugar, agradecer a los dos tutores de este proyecto, Cristian Tejedor García y Mario Corrales Astorgano, por su apoyo, ayuda y dedicación ya que gracias a ellos este TFG ha podido seguir una línea adecuada y se ha podido terminar de una forma muy gratificante.

En segundo lugar, agradecer a mi familia porque sin su apoyo, confianza y cariño hubiera sido imposible llegar hasta el final de esta etapa universitaria.

Por último, agradecer también a todas las personas que han formado parte de estos 4 años y a mis amigos de toda la vida porque también son partícipes de que haya llegado hasta aquí.

Resumen

En la actualidad, los juegos y aplicaciones de dispositivos móviles han adquirido una gran relevancia estando presentes en nuestras vidas en casi todo momento. El hecho de que estas aplicaciones móviles perduren en el tiempo y mantengan nuestra atención y ganas de seguir jugando tiene mucho que ver con la forma que tiene este de emparejarnos con otros jugadores. Un sistema de emparejamiento (matchmaking) es un mecanismo que incorpora un juego o aplicación multijugador online mediante el cual se emparejan jugadores en función de una serie de factores. En el presente, los sistemas de matchmaking se antojan críticos en los ciclos de vida de los videojuegos y deben estar diseñados de tal forma que hagan de la experiencia del usuario que juega la más satisfactoria posible y que por ello esta se transforme en una continuidad. El cometido de este TFG es el estudio, la propuesta y el desarrollo de un sistema de matchmaking genérico y adaptable a aplicaciones de carácter didáctico, que tenga como característica principal el equilibrio entre el nivel de los usuarios que juegan.

Abstract

Nowadays, mobile games and applications have acquired a big relevance being present in our lifes most of the time. The fact that makes these mobile apps survive along time and keep our attention and desire to continue playing is related to the way players are matched to each others. A matchmaking system is a mechanism incorporated in an online multiplayer game or application by which players are matched following multiple factors. Currently (in 2022), matchmaking systems are a critical element in the life cycle of videogames and they must be designed to make the player's experience as satisfying as possible and, for this reason, she or he continues playing. The purpose of this bachelor's thesis is the study, proposal and development of a generic matchmaking system that could be adapted to educational applications, which aims at providing a balanced group of players in terms of similar skill level.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

1.1. Contexto

El origen de este proyecto es el año 2016, cuando uno de los tutores de este proyecto, Cristian Tejedor García, desarrolló una aplicación educativa (TipTopTalk!) para la mejora de la pronunciación extranjera en su TFM [1]. Dicha aplicación incluía elementos de gamificación, de forma que los usuarios mediante una serie de ejercicios individuales pudieran entrenar su pronunciación de una forma más entretenida y amena.

En 2017 se extendió la funcionalidad de este proyecto añadiendo una modalidad multijugador, en la que los usuarios podían jugar los unos contra los otros, para motivar aún más el uso de la aplicación, dotándola de una mejora significativa en términos de utilización y, por consiguiente, de aprendizaje entre los usuarios. Este sistema multijugador permitía la creación de partidas entre jugadores dónde se competía intentando conseguir el mayor número de puntos posibles mediante la correcta pronunciación de la palabra solicitada. Toda la aplicación estuvo sostenida mediante la API de Google Play Games y pasó a llamarse Clash of Pronunciations (COP) [2].

Posteriormente, en el año 2020, COP sufrió un gran inconveniente con la cancelación por parte de Google de toda su API multijugador [3], lo que dejó a la aplicación sin funcionamiento. Este problema fue solventado por el TFG de Andrés de la Maza [4], que buscó replicar de la mejor forma el funcionamiento perdido mediante una API propia. Pero no solo se tuvo que cambiar el backend, sino que para la parte de frontend, era necesario migrar y actualizar el código obsoleto para adaptarlo a la nueva API de COP. De este proceso se encargó el TFG de Juan García [5] en el año 2021.

Partiremos del matchmaking que nos ofrece COP, que consiste en un ranking por puntuaciones dónde un jugador puede escoger a los jugadores que estén 5 puestos por encima y por debajo de él. El objetivo del presente TFG será crear un nuevo sistema de matchmaking que permita a aplicaciones educativas, entre ellas COP, emparejar jugadores de nivel similar que dote de equilibrio a las partidas entre jugadores, permitiendo aumentar la competitividad y satisfacción de los usuarios [6]. Para ello, se deberá utilizar un sistema de clasificación que relacione a cada jugador con una puntuación equivalente a su nivel, y que irá variando en función de los resultados que dicho jugador obtenga en las partidas que

dispute [7]. Este matchmaking se desarrollará utilizando como referencia COP, pero el principal objetivo es que este sea escalable y adaptable a otras aplicaciones educativas similares.

1.2. Motivación

Los sistemas de emparejamiento (matchmaking) son una de las características principales en la experiencia de usuario en juegos online [8]. En la actualidad se ha convertido en un elemento imprescindible en el ciclo de vida de cualquier tipo de aplicación o videojuego que incorpore uno de ellos. El diseño y la implementación de un sistema de matchmaking es primordial ya que es el que puede hacer que los usuarios que han comenzado a jugar continúen y atraigan más jugadores, con los consiguientes ingresos y repercusión que supone, o lo dejen prontamente, derivando en pérdidas y en convertirse en una aplicación sin prestigio [6]. Si un jugador pierde la mayoría de partidas que disputa porque se enfrenta a rivales más habilidosos, este se frustrará y muy posiblemente deje de jugar, o si experimenta un tiempo de espera excesivo a la hora de encontrar una partida. De la misma forma que si un jugador gana siempre se cansará pronto. Para ello, todo proceso de elección y desarrollo del sistema de emparejamiento debe ser metódico y cuidadoso, adaptándolo a las características del juego en el que se quiere implementar y buscando el balance adecuado entre el nivel de los jugadores emparejados y el tiempo que el propio sistema tarda en unirlos. En definitiva, un buen sistema de matchmaking debe hacer que la satisfacción de todos los usuarios que utilicen el juego sea la máxima posible y que eso se traduzca en una permanencia y una atracción de más jugadores que garanticen el éxito de la aplicación.

Este Trabajo Fin de Grado pretende proponer y desarrollar un nuevo sistema de emparejamiento (matchmaking) enfocado para aplicaciones educativas y adaptable a cualquier tipo de aplicación que tenga conexión a internet e implemente un sistema multijugador, buscando un emparejamiento adecuado teniendo en cuenta el nivel de habilidad de los jugadores que se enfrenten, con el objetivo de que la satisfacción de los usuarios al usarla sea la mayor posible. Para ello, se tomará como referencia la aplicación COP, que tiene un sistema de matchmaking más básico, aunque el objetivo es generalizar el sistema de emparejamiento para poder ser utilizado en otras aplicaciones educativas similares.

1.3. Objetivos

El objetivo principal del proyecto es realizar un servidor de matchmaking genérico para aplicaciones educativas que empareje a los diferentes jugadores en función de una serie de criterios que permitan que las partidas sean equilibradas, y con ello, se aumente el nivel de satisfacción de los jugadores que la utilicen. Este objetivo general se puede dividir en:

- Objetivo 1: Realizar un estudio de los diferentes métodos de matchmaking existentes para escoger los criterios adecuados para este TFG.
- Objetivo 2: Crear un servidor web de matchmaking que se comunique con los clientes mediante una API necesaria para el correcto funcionamiento de la aplicación.

1.4. Metodología

Para el desarrollo del proyecto se ha decidido utilizar una metodología iterativa e incremental [9]. Esto se justifica debido a ser un único desarrollador para el trabajo completo y ser una metodología flexible ante posibles cambios en la planificación y los requisitos, que son probables a darse debido a la inexperiencia y ser un proyecto nuevo. Las metodologías ágiles han sido descartadas debido a que son más apropiadas para grupos de trabajo de varias personas y se adaptan más a proyectos cuyos requisitos son vulnerables a cambiar drásticamente [10].

Utilizando un método iterativo es más sencillo realizar un seguimiento por parte de los clientes, que pueden ver de manera cercana cómo es el último avance, si cumple las especificaciones requeridas y si existe alguna corrección en los requisitos. Todo ello, permite realizar una revisión antes de avanzar a la siguiente iteración y continuar a la siguiente fase del desarrollo, lo que reduce el riesgo de que existan retrocesos que perjudiquen el progreso del proyecto.

Un esquema de esta metodología se puede ver en la Figura 1.1. Además, en la sección 4 se detallará lo realizado en cada una de estas iteraciones.

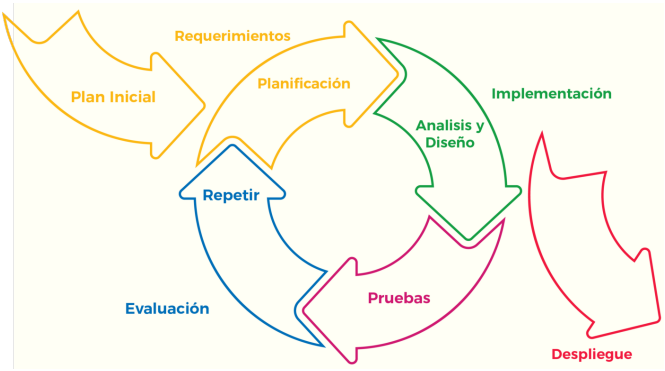

Figura 1.1: Metodología iterativa e incremental [9]

1.5. Estructura de la memoria

- **Introducción**: En esta parte se expone el proyecto de manera general, comentando el contexto, la motivación, la metodología a emplear, qué objetivos se pretenden alcanzar y la estructura que va a presentar la memoria.
- **Estado de la cuestión**: Capítulo en el que se detallan características del estado del proyecto como de dónde nace, partes teóricas utilizadas y tecnologías empleadas.
- **Planificación**: Estimación y descripción de las iteraciones, así como su tiempo de ejecución, coste, riesgos.
- **Descripción de las iteraciones**: Descripción del trabajo realizado en cada una de las iteraciones hasta finalizar el proyecto.

- **Estado final de la aplicación**: Descripción del análisis y diseño de la aplicación, con todas sus figuras y diagramas.
- **Pruebas**: En esta sección se agrupan todas las pruebas realizadas y los resultados de cada una.
- **Conclusiones y trabajo futuro**: Conclusiones obtenidas del desarrollo del proyecto y listado de mejoras propuestas de cara al futuro.
- **Apéndices**: Documentos adicionales que incluye una lista de acrónimos utilizados en la memoria, los manuales de uso y de instalación del sistema, tablas con las pruebas del sistema realizadas y las tablas comparativas del matchmaking de las diferentes aplicaciones educativas utilizadas.
- **Bibliografía**: Listado de referencias utilizadas para el desarrollo del proyecto.

Capítulo 2

Estado de la cuestión

2.1. TipTopTalk! y COP

Como se ha explicado en la sección 1.1, TipTopTalk! fue la aplicación pionera. Fue desarrollada por Cristian Tejedor García durante el curso 2015-2016 [1] y fue creada con el fin de hacer del aprendizaje y la práctica de la pronunciación de diferentes idiomas una actividad divertida, didáctica y más entretenida que los métodos convencionales. Se trata de una aplicación para móviles Android que integra ejercicios que permiten a un jugador mejorar la pronunciación de varios idiomas (español, inglés y chino) de manera individual [11] [12]. Para ello, TipTopTalk! incorpora diferentes juegos a elección del usuario y de facetas distintas en función de cual se quiera entrenar (exposición, discriminación y pronunciación). Según el grado de acierto de sus respuestas, los jugadores obtienen puntos, logros y trofeos, de manera que pueden ver representado su nivel en un ranking y aumentar su competitividad.

Esta aplicación evoluciona posteriormente, mediante el Trabajo de Fin de Grado de Rafael Sillero Navajas [13], añadiéndole funcionalidad extra con respecto a TipTopTalk! mediante un sistema multijugador con el que los jugadores podían enfrentarse entre ellos y obtener puntuaciones en base a sus aciertos y errores. De esta forma, COP se convirtió en una versión más amplia y con más registros y posibilidades que su predecesora y, en definitiva, cumplía su función didáctica de una manera más eficaz. Esta versión multijugador implementa un matchmaking básico. Este se basa en una clasificación o ranking mediante el cual los jugadores son situados según sus puntuaciones. Para cada jugador, según dicha puntuación, le aparecen los 5 jugadores por encima y por debajo en el ranking y este puede elegir a cualquiera de ellos para disputar la partida (entre 1 y 4) [14]. Una vez disputada la partida, según los resultados se reordena la clasificación de forma que los candidatos posteriores serán distintos. En la figura 2.2 se puede ver la interfaz del sistema multijugador que implementa COP.

Los TFG posteriores [4] [5] mejoraron aún más la aplicación de COP y la adaptaron para su uso después de que la API de Google Play Games suprimiera sus servicios y quedara inutilizable. En este TFG se continúa con esta última versión, en la que hay una API propia para el servidor.

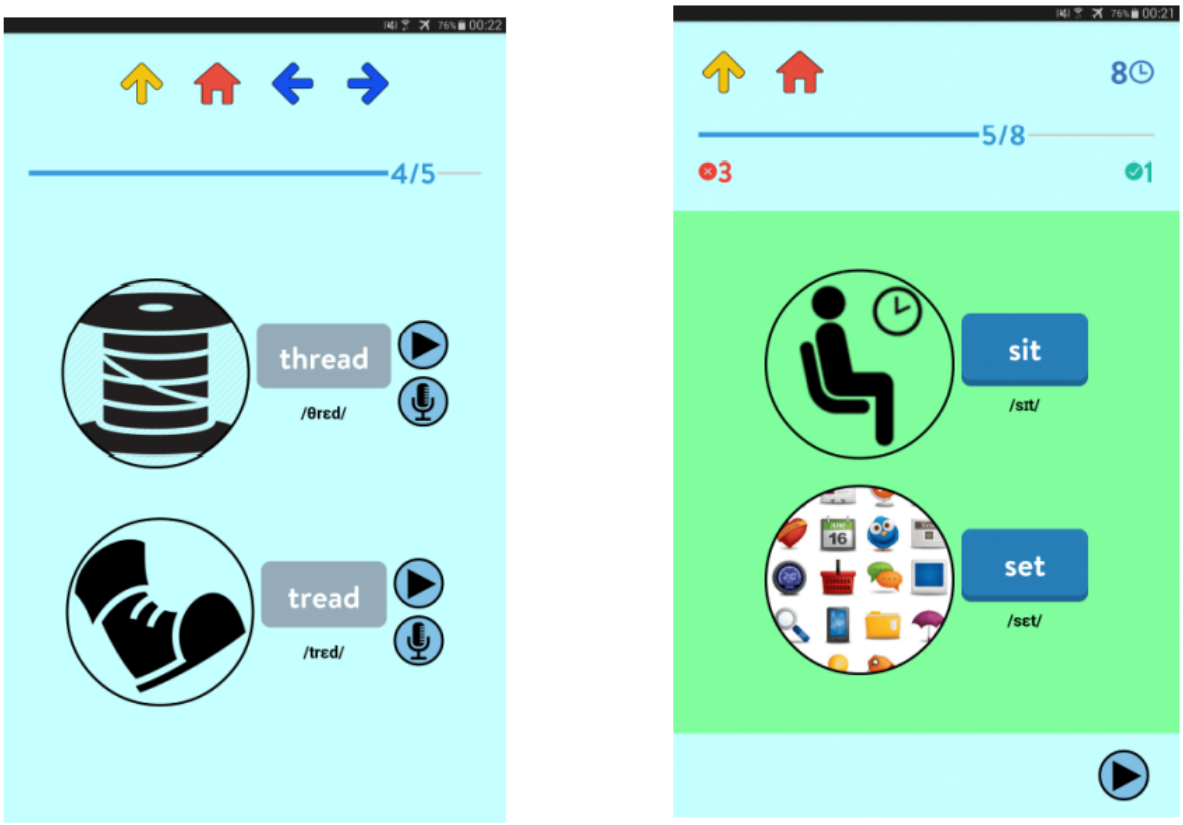

Figura 2.1: Apariencia de la aplicación TipTopTalk!

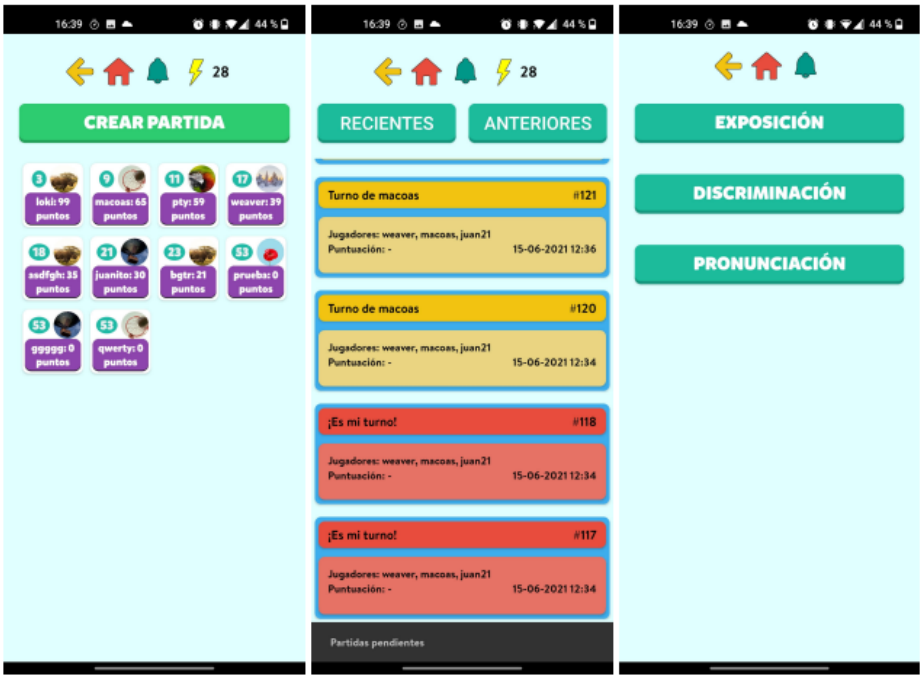

Figura 2.2: Sistema de creación de partidas multijugador de COP

2.1.1. Tipos de matchmaking

Un sistema de matchmaking es un mecanismo que incorporan los videojuegos o aplicaciones en las que participan varios jugadores que consiste en el emparejamiento de usuarios en función de una serie de criterios establecidos [8]. Estos factores de emparejamiento son muy variables y extensos y dependen del tipo de videojuego, de las modalidades que presente, de la extensión que tenga, del número de jugadores que acapare o de los requisitos establecidos por los desarrolladores.

En general, el principal cometido de los sistemas de matchmaking es crear en los jugadores una experiencia satisfactoria y de esa forma animarles a seguir jugando además de atraer a nuevos competidores. En la actualidad, se trata de un elemento crítico que puede hacer que el producto sea exitoso o ruinoso en cuestión de poco tiempo [15]. Si un jugador pierde muchas partidas consecutivas porque el matchmaking no hace del emparejamiento un encuentro equilibrado este se cansará de forma pronta. Por el contrario, si un jugador gana siempre el juego se tornará aburrido y monótono, conllevando también que deje de jugar y una repercusión negativa para la aplicación. El diseñador del matchmaking tiene por lo tanto una labor crucial a la hora de establecer los factores de emparejamiento y que sea de una manera equilibrada.

Para encontrar el modelo adecuado para el matchmaking que se va a desarrollar, se realizó un estudio de varias aplicaciones de carácter didáctico prestando atención al sistema de emparejamiento que tenían implementado para comprobar de primera mano qué posibilidades existen, qué tipos hay, qué características tienen, ver cómo funcionan y cómo utilizan las estadísticas que obtienen de los jugadores y, de esa forma, elegir qué factores se pueden utilizar en nuestro matchmaking genérico. Las tablas resultantes del estudio comparativo están recogidas en el Apéndice C.

A continuación se van a enumerar algunos tipos de matchmaking existentes junto con sus características, ventajas y desventajas, además de los principales inconvenientes que podemos encontrar.

2.1.2. Problemas genéricos de los matchmaking

Existen múltiples factores a tener en cuenta a la hora de desarrollar un matchmaking, así como varios problemas que pueden surgir que hay que tener presentes para evitar que sucedan, a la vez que existen métodos que los solucionan [16].

- Jugadores con mala actitud: Esto engloba a jugadores que abandonan partidas, que usan métodos de hacking para hacer trampas o que presentan actitudes hostiles hacia otros jugadores. Para ello los juegos implementan sistemas de reporte de usuarios, mecanismos de seguridad que eviten trucos o *cheats* y almacenamiento de estadísticas de abandono de partidas de tal forma que tengan restricciones en futuros emparejamientos.
- Colas excesivas: Se entiende por cola el lugar donde se van incorporando jugadores que solicitan partidas. Si estas colas crecen de manera excesiva se provocan retardos y congestiones del sistema que hacen que la experiencia de los usuarios sea mala. Una manera de solventar esto es

reduciendo el nivel de filtro del matchmaking para que sea más fácil encontrar un rival. Los juegos a día de hoy también informan sobre el estado de la cola, de forma que si los jugadores ven que el juego está congestionado pueden abandonar la cola permitiendo que se desatasque.

- Alta latencia: La alta latencia se debe normalmente a conexiones lejanas entre cliente y servidor o a que son de mala calidad. Este factor puede ser más o menos importante dependiendo del número de jugadores de la aplicación y su localización. Si el juego busca crecer internacionalmente deberá mejorar sus servidores e incorporar varios en diferentes partes del mundo de manera que las conexiones de todos los jugadores sean lo más cercanas posible.

- Roles no familiares: Este es un inconveniente de los juegos que incorporan roles, que son tipos de personajes que poseen habilidades diferentes entre ellos (como se hablará en la sección 2.1.4). El problema radica en los usuarios que no pueden escoger para una partida el rol con el que se sienten más cómodos (esto puede deberse a factores de preferencia según el nivel de jugador). Si un jugador juega muchas partidas con un rol impropio, su nivel de juego será menor traduciéndose en derrotas y por ende menos puntos de nivel, además de una insatisfacción que hará que deje de jugar. Una manera de minimizar este hecho es establecer mayor libertad en la elección, aumentar el rango de roles o utilizar las estadísticas para que las partidas estén conformados por jugadores de distinto rol de manera que todos puedan jugar con el que le es más cómodo.

2.1.3. Matchmaking aleatorios y semialeatorios

Estos matchmaking son aquellos que no siguen un criterio a la hora de establecer emparejamientos entre los usuarios (aleatorios), o bien siguen alguno de poca dificultad que es prácticamente despreciable (semialeatorios) [15]. Están pensados para juegos sencillos que no requieren establecer un equilibrio entre los rivales, o juegos de azar donde las habilidades del jugador no están relacionadas con el resultado de la partida.

También es implementado en aplicaciones que le dan mucha importancia al tiempo de espera que pueda sufrir el usuario, ya que con los matchmaking aleatorios y su ausencia de factores, el tiempo de espera a buscar rival y comenzar una partida es muy reducido.

Algunos juegos que implementan este tipo de matchmaking son aplicaciones de preguntas y respuestas cortas como Trivia Deluxe [17] o Preguntados [18].

2.1.4. Matchmaking por roles

Estos matchmaking basan su rendimiento en emparejar a los distintos jugadores en función del tipo de ”rol” que, por estadística, es su preferido [19]. Se entiende por rol un tipo de personaje con unas características únicas con respecto a otros (personaje de ataque, personaje de defensa...). Este sistema está pensado específicamente para juegos en los que las partidas involucran a varios jugadores a la vez y existen entre ellos diferentes avatares con características distintas.

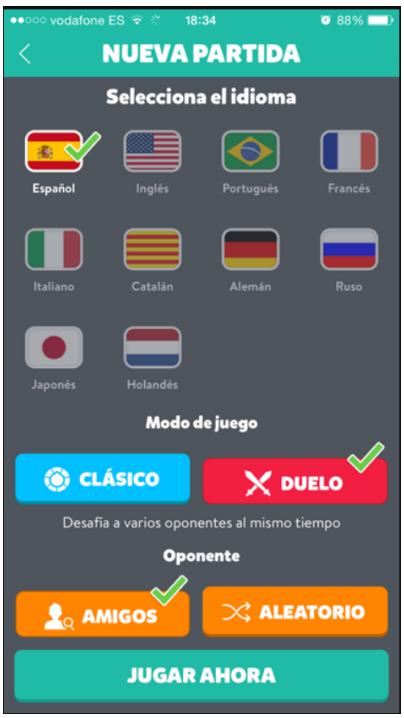

Figura 2.3: Búsqueda de partida de Preguntados

Supongamos un juego en el que las partidas son de equipos 5vs5 y cada jugador puede escoger un personaje que es de un tipo en concreto y no se pueden repetir. Si un jugador cuya preferencia es un personaje del tipo1 no puede elegirlo debido a que otro jugador ha sido más rápido, o bien tiene algún tipo de preferencia por ranking, y está obligado a escoger otro de otro tipo su nivel de satisfacción se verá reducido. Si esto además se repite en el tiempo, seguirá disminuyendo hasta que abandone el juego.

Por eso, este matchmaking busca que las partidas estén conformadas entre jugadores cuyo rol preferido sea diferente entre ellos, de forma que no haya conflictos en la elección y cada uno pueda jugar la partida con su tipo predilecto. Para ello, se guardan y utilizan minuciosamente las estadísticas de cada jugador con respecto a los roles que más usa y su índice de victorias/derrotas con cada uno.

El juego más famoso que implementa este tipo de matchmaking es el League of Legends [20].

2.1.5. Sistema de clasificación

Estos matchmaking son aquellos que utilizan una clasificación o factor numérico a la hora de emparejar jugadores [15]. Esto puede ser un ranking, una liga o un medidor de habilidad entre otros muchos ejemplos. Es el tipo de matchmaking más numeroso y utilizado y del que más estudios y variantes hay en la actualidad.

El principal cometido de los sistemas de clasificación es otorgar equilibrio entre las partidas de los jugadores, de manera que los enfrentamientos producidos sean entre rivales de una habilidad similar.

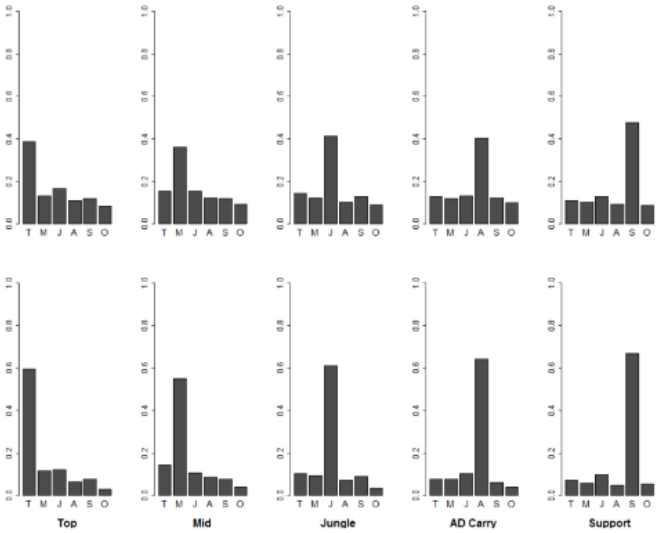

Figura 2.4: Gráfica con el uso de cada tipo de rol en League of Legends (Cada barra es un rol) (Primera fila: liga Bronce, Segunda fila: Liga Challenger) [19]

En este tipo de matchmaking se pueden establecer múltiples criterios propios de la aplicación en base a las estadísticas (balance de victorias/derrotas, por puntos en cada partida, por posición en una clasificación, por racha de victorias/derrotas, nivel de experiencia...), pero existen otros tipos de sistemas de clasificación ya existentes nacidos de estudios pasados que utilizan un medidor de habilidad como principal criterio de emparejamiento [21].

El primero de ellos fue el llamado Sistema Harkness [22] que tuvo su nacimiento en las partidas de ajedrez en la década de los 50. Cada jugador poseía una puntuación equivalente a su nivel y este sistema calculaba su ratio de victorias-derrotas al finalizar una partida. Si este ratio estaba por encima de un 50 % aumentaba su puntuación en 10 puntos por cada 1 % hasta llegar a 50. Si estaba por debajo, se restaba en la misma cantidad.

Ejemplo [22]: Un jugador con una puntuación de 1600 juega en la ronda 11 de un torneo con un balance de 2–8 (22.7 %) contra un rival que tiene de puntuación 1850. El ratio del jugador hasta llegar al 50 % es de 27.3 %, por lo que su nueva puntuación es *1850 - (10 × 27.3) = 1577*.

Años más tarde, el profesor Arpad Elo, evolucionó este método y creo el suyo propio: El sistema ELO [23]. Este método consistía también en tener un factor numérico que representaba el nivel de habilidad de un rival solo que la manera de obtenerlo era ligeramente diferente buscando más precisión en el cálculo. A diferencia de su predecesor, el algoritmo del sistema ELO modifica la puntuación de un rival en función del resultado de una partida y de una predicción previa al juego que se calcula según el nivel de habilidad de ambos competidores.

Esta predicción se obtiene mediante la fórmula:

$$Ea = \frac{1}{1 + 10^{-\frac{(Ra-Rb)}{400}}} \tag{2.1}$$

donde Ea es la predicción obtenida, y Ra y Rb son las puntuaciones de los jugadores A y B respec-

tivamente, ambos antes de disputarse la partida.

Teniendo en cuenta estos dos factores, la fórmula que determina la nueva puntuación de un jugador A es:

$$Ra' = Ra + K(Sa - Ea) \tag{2.2}$$

donde Ra es la puntuación del jugador antes de la partida, Sa es el resultado de dicha partida (valor comprendido entre 0 y 1), Ea es la puntuación esperada resultante de la predicción previa y K es un factor de ajuste constante (K=16 o K=32 según si los jugadores son maestros o no. Se entiende por maestro aquel jugador que supera los 2000 puntos ELO).

En la figura 2.5 se puede ver cómo se representan las nuevas puntuaciones y las probabilidades de dos jugadores en forma de gráfica.

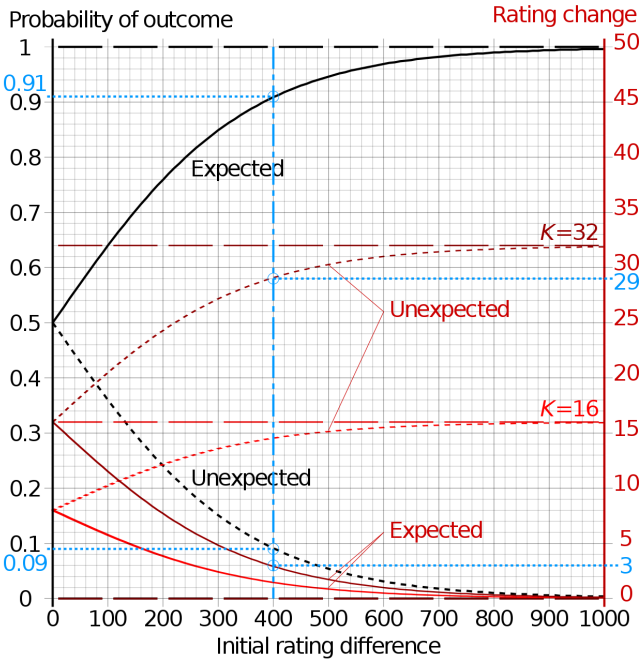

Figura 2.5: Gráfica que ilustra el cambio de puntuaciones según la puntuación inicial y la probabilidad de victoria de dos jugadores de 1400 y 1800 puntos [23]

Este método es aplicado en la actualidad en el mundo del ajedrez pero ha trascendido a más ámbitos como el que nos ocupa, los videojuegos, como pueden ser CS:GO [24], Hearthstone [25] o Call of Duty [26], eso sí, con modificaciones que lo adaptan a ese mundo o lo complementan con otro tipo de sistemas.

En los siguientes años los estudios sobre estos métodos de clasificación continuaron vigentes buscando sistemas aún más precisos a la hora de determinar el nivel de habilidad de un jugador y emparejarlo con usuarios de similar nivel. Es aquí donde aparece el algoritmo Glicko.

El sistema Glicko [27] está basado en la filosofía de un sistema ELO en cuanto a medir el nivel de habilidad a través de un dato numérico variable según resultados de partidas jugadas y la probabilidad

de ganarlas, con la diferencia de que se le añade un parámetro más que mide el tiempo sin jugar lo cual su inventor, Mark E. Glickman, pensaba que ese aspecto afectaba a la habilidad de un jugador. Este parámetro ratings deviation (RD), será más alto cuanto más tiempo sin competir lleve un jugador. De esta forma, Glicko otorga un nivel de precisión más alto que su predecesor.

Otra diferencia sustancial con ELO es que para llevar a cabo el cálculo de las puntuaciones, Glicko agrupa las partidas en lo que se denomina periodo, que no es más que un intervalo de tiempo que una vez finalizado calcula y actualiza todas las puntuaciones de todos los jugadores que se han involucrado compitiendo en al menos una partida. Este periodo de tiempo es variable y ajustable según las características de la aplicación o la carga de usuarios que presente.

El cálculo de las nuevas puntuaciones se realiza siguiendo tres pasos:

En primer lugar se calcula un RD inicial:

$$RD = min(\sqrt{(RD_0)^2 + c^2 t}, 350) \tag{2.3}$$

- RD_0 es el valor de RD que posee un jugador justo antes de comenzar el cálculo.
- t representa el número de periodos que el jugador lleva sin ser actualizado, y por consiguiente que lleva sin disputar una partida
- 350 es el valor por defecto que se le da a un jugador nuevo del que no se tienen registros.
- c es una constante que denota el crecimiento de la incertidumbre entre los periodos de puntuación. Normalmente actúa como constante y se establece en 34,6. Viene estipulado por la siguiente fórmula:

$$c = \sqrt{\frac{(350^2 - 50^2)}{100}} \approx 34,6 \tag{2.4}$$

  Donde 350 y 50 son el RD inicial y RD medio asumido por observación, mientras que 100 representa el número de periodos asumidos que son necesarios para recuperar el RD inicial partiendo desde un RD=50.

En segundo lugar se calculan las nuevas puntuaciones:

$$r = r_0 + \frac{q}{\frac{1}{RD^2} + \frac{1}{d^2}} \sum_{i=1}^{m} g(RD_i)(s_i - E(s|r_0, r_i, RD_i)) \tag{2.5}$$

Donde:

$$g(RD_i) = \frac{1}{\sqrt{1 + \frac{3q^2(RD_i^2)}{\pi^2}}} \tag{2.6}$$

$$E(s|r_0, R_i, RD_i) = \frac{1}{1 + 10^{(\frac{g(RD_i)(r_0 - r_i)}{-400})}} \tag{2.7}$$

$$q = \frac{\ln 10}{400} = 0,00575646273 \tag{2.8}$$

$$d^2 = \frac{1}{q^2 \sum_{i=1}^{m} (g(RD_i))^2 E(s|r_0, r_i, RD_i)(1 - E(s|r_0, r_i, RD_i))} \tag{2.9}$$

- m es el número de partidas disputadas por un jugador en ese periodo.
- r_0 representa la puntuación que posee el jugador antes del cálculo.
- r_i son las puntuaciones de cada rival antes del cálculo.
- s_i representa el resultado de cada partida [0,0.5,1].

Por último, se ajusta el RD calculado anteriormente una vez calculadas las nuevas puntuaciones:

$$RD' = \sqrt{(\frac{1}{RD^2} + \frac{1}{d^2})^{-1}} \tag{2.10}$$

Una vez realizados los cálculos, el resultado son una puntuación y un RD que serán utilizados como criterio de emparejamiento en el periodo siguiente.

2.2. Entorno y herramientas tecnológicas

En esta sección se van a exponer las tecnologías y herramientas utilizadas para el desarrollo de este TFG y las características principales de cada una de ellas. De manera general, el servidor de matchmaking está soportado sobre Node.js, las llamadas API necesarias están realizadas con Express.js y la base de datos utilizada es MongoDB. Así mismo, se han utilizado herramientas de soporte como Postman para simular llamadas.

2.2.1. Node.js

Node.js [28] es un entorno de tiempo de ejecución de JavaScript. Este entorno de tiempo de ejecución en tiempo real incluye todo lo que se necesita para ejecutar un programa escrito en JavaScript. Node.js utiliza un modelo de entrada (solicitudes) y salida (respuestas) sin bloqueo controlado por eventos que lo hace ligero y eficiente. Su gran virtud es la creación de aplicaciones de red rápidas, ya que es capaz de manejar una gran cantidad de conexiones simultáneas con un alto nivel de rendimiento, lo que equivale a una alta escalabilidad. Sus principales características son [29]:

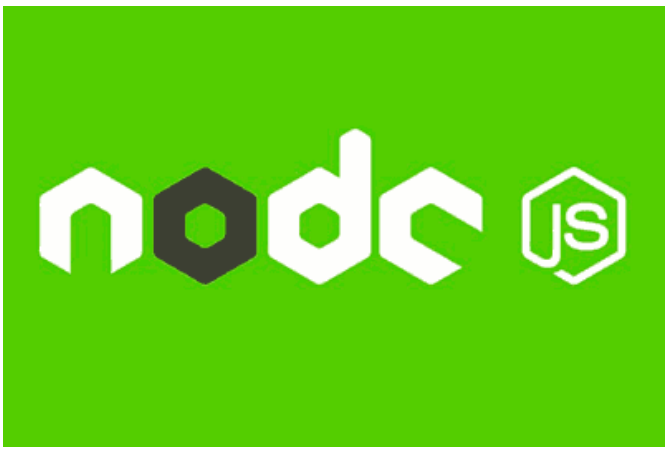

Figura 2.6: Logo de Node.js

- Multiplataforma: Se puede utilizar Node.js en numerosos dispositivos con sistemas operativos diferentes.
- Escalabilidad: Node.js permite crear aplicaciones que pueden aumentar su tamaño sin que se vea reducido su rendimiento.
- Alto tráfico: Node.js está diseñado para soportar altas cantidades de usuarios concurrentes ya que utiliza un único hilo que crea un evento por cada petición, a diferencia de los métodos tradicionales que creaban un hilo por cada acción y se bloqueaban hasta que terminasen (ver figura 2.7).
- Entorno asíncrono: Node.js no tiene operaciones bloqueantes como las de entrada y salida. La utilización de eventos permite a Node continuar con otras operaciones hasta que se le notifica del evento ha terminado para volver con él.
- Rapidez: La velocidad de ejecución de código de Node es de las más rápidas por la utilización del motor V8 de Google.
- Código abierto: El hecho de que Node.js sea un sistema de código abierto permite tener una extensa documentación y que existan soluciones a errores por parte de otros miembros de la comunidad aplicables a todos los usuarios.

2.2.2. npm

npm [31] es un gestor de paquetes desarrollado para JavaScript a través del cual podemos obtener cualquier librería con tan solo una sencilla línea de código. npm puede actuar de dos formas:

- Como un repositorio de código abierto. Es decir, un almacén utilizado para que cualquiera que lo desee pueda publicar y compartir herramientas propias escritas en JavaScript que después pueden ser utilizadas por otros usuarios para su cometido.
- Como una herramienta de línea de comandos que permite interactuar con plataformas en línea, ya sean navegadores o servidores. Esto da la posibilidad de instalar y desinstalar paquetes, ges-

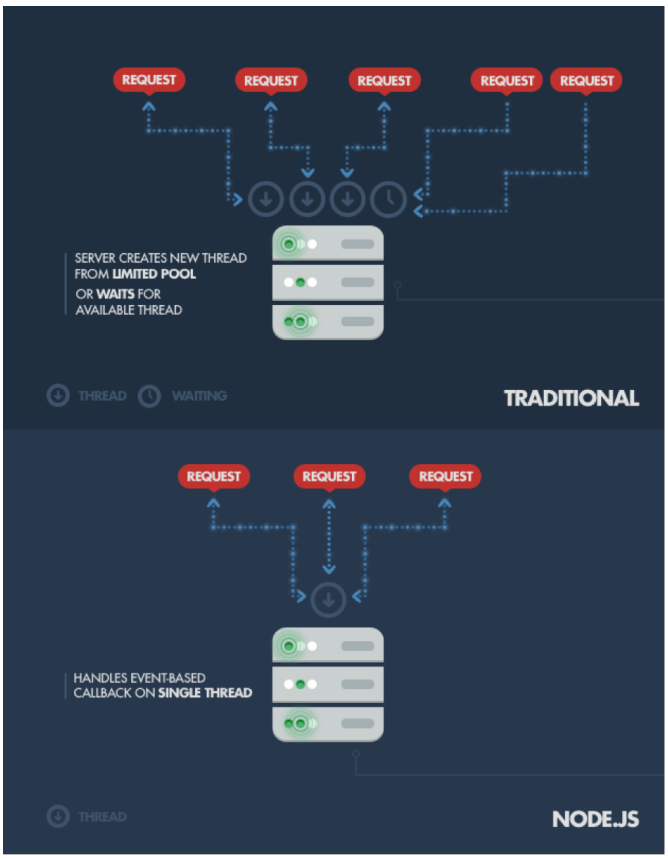

Figura 2.7: Esquema comparativo de Node.js con otras tecnologías en el uso de hilos para atender operaciones [30]

Figura 2.8: Logo de npm

tión de dependencias necesarias para ejecutar un proyecto o control de las versiones de una determinada librería y su compatibilidad.

2.2.3. Express

Express [32] es el framework web más popular de Node. Esta herramienta nos proporciona una serie de mecanismos para estructurar las llamadas a nuestro servidor:

- Escribir un conjunto de manejadores de peticiones con diferentes verbos HTTP para las rutas que se especifiquen.
- Añadir procesamiento de peticiones ”middleware” adicional en cualquier punto dentro del manejo de la petición.

Figura 2.9: Logo de Express

2.2.4. MongoDB

MongoDB [33] es una base de datos no relacional. Esta base de datos es orientada a documentos, cuyo nombre viene debido al tipo de registros donde almacena sus datos (documentos). Estos documentos son almacenados en BSON, que es una representación binaria de JSON. Una de las diferencias más importantes con respecto a las bases de datos relacionales es que no es necesario seguir un esquema fijo, sino que tienen cierta flexibilidad a la hora de elegir el formato adecuado para el almacenamiento de nuestros datos. Los documentos así mismo se almacenan en lo que se denomina colección (concepto similar a una tabla de una base de datos relacional).

Las principales ventajas de MongoDB son:

- Tiene una gran compatibilidad con JavaScript debido a la existencia de numerosas librerías y el formato JSON de sus entradas.
- Es una herramienta útil para fuentes de datos grandes y crecientes por su flexibilidad.
- Es una herramienta con un coste bajo y fácil de utilizar.
- Es ideal para entornos con pocos recursos de computación.

2.2.5. Postman

Postman [34] es una herramienta que nos permite realizar peticiones sobre una o varias APIs a fin de poder probar su funcionamiento. Postman actúa como cliente y lanza las llamadas al servidor que

Figura 2.10: Logo de MongoDB

sustenta los endpoints a los que queremos acceder para posteriormente mostrarnos los resultados de dicha petición. Las consultas son almacenables y reutilizables por medio de colecciones.

Figura 2.11: Logo de Postman

Capítulo 3

Planificación

3.1. Planificación inicial

El tiempo dedicado a este TFG se ha planificado teniendo en cuenta la situación del alumno que se encuentra realizando las prácticas de empresa durante la realización del mismo. De esta forma, se estableció al comienzo del proyecto un trabajo de 14 horas semanales desde finales de octubre hasta mediados de abril, lo que resultaría en un total de 378 horas, superando así las 300 horas mínimas que se especifica en la normativa vigente sobre el tiempo de trabajo que se debe dedicar a un TFG.

Las horas de trabajo se dividen en un total de 26 semanas, estimando unas 2 horas de trabajo diario, y la planificación del proyecto ha dado lugar a 7 iteraciones, repartidas de la siguiente forma:

- 1ª iteración: En esta primera fase se tendrá la primera reunión entre el alumno y los tutores para comentar las cuestiones pertinentes, establecer los requisitos que deberá cumplir el proyecto así como las tecnologías que se utilizarán para su desarrollo.
- 2ª iteración: Durante esta iteración, el alumno preparará su entorno de trabajo con las herramientas establecidas en la fase anterior y desarrollará un pequeño sistema que simule la comunicación entre un cliente y dos servidores, a fin de utilizarlo en pasos posteriores.
- 3ª iteración: En esta etapa, se hará un estudio sobre los diferentes matchmaking existentes en aplicaciones móviles de una temática similar a la que deseamos implementar, a modo de entender su funcionamiento, explorar las distintas posibilidades y proponer uno propio para su desarrollo.
- 4ª iteración: Desarrollo de la funcionalidad de los servidores de la aplicación.
- 5ª iteración: Desarrollo de interfaces de usuario para crear las vistas del servicio multijugador de la aplicación.
- 6ª iteración: Etapa de pruebas generales de todo el sistema a fin de optimizar la aplicación y comprobar que esté libre de errores.

- 7ª iteración: En la iteración final se desplegará la aplicación en las máquinas virtuales cedidas por la Universidad.

En la figura 3.1 se puede ver la distribución en semanas que ha seguido la planificación del desarrollo de cada iteración. Se puede apreciar como las iteraciones 4 y 5 son las que más tiempo requieren ya que son las correspondientes a toda la escritura de código.

En la figura 3.2 se puede ver como queda la planificación del reparto de horas totales en cada iteración. La gráfica sigue la misma forma que la gráfica anterior ya que no existen variaciones en las horas diarias aplicadas en cada etapa.

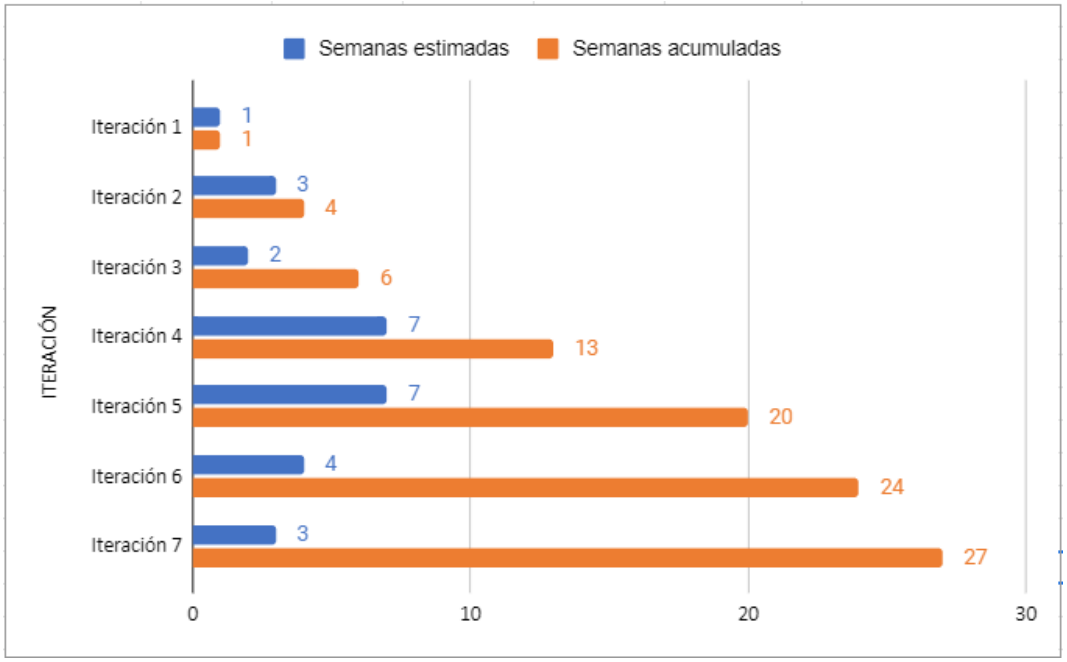

Figura 3.1: Distribución del número de semanas en cada iteración

3.2. Análisis de riesgos

En esta sección se van a detallar los riesgos cuya aparición podría conllevar retrasos respecto a la planificación inicial de cada una de las iteraciones y, por consiguiente, de la finalización del proyecto final. Este análisis de riesgos viene especificado en la Tabla 3.1, en la que se detalla el riesgo, la probabilidad de que ocurra, una breve descripción y los días de retraso que supondría.

En relación con este análisis de riesgos y la posibilidad de que estos ocurran, también se ha realizado un plan de acción detallado cuya función es mostrar las acciones a llevar a cabo para reducir la probabilidad de aparición de los riesgos, así como establecer un plan de mitigación que minimice lo máximo posible el tiempo de retraso si dicho riesgo llega a materializarse. Este plan de acción se encuentra en la Tabla 3.2, añadiendo el grado de exposición, el cual se calcula multiplicando las columnas de probabilidad de que el riesgo suceda y el retraso estimado.

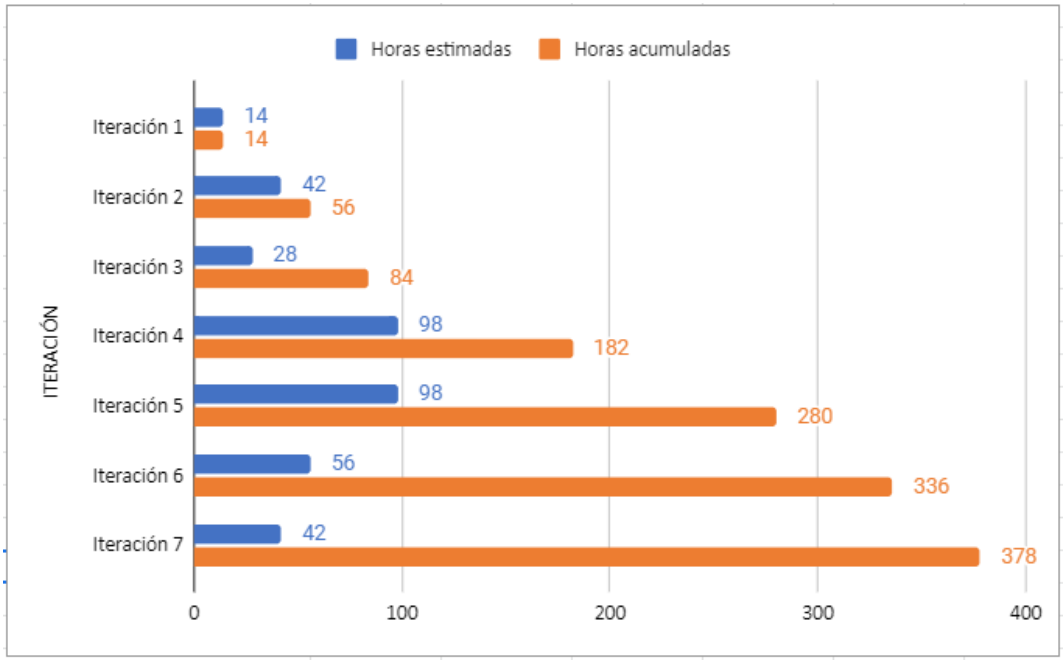

Figura 3.2: Distribución del número de horas en cada iteración

3.3. Entorno tecnológico

3.3.1. Herramientas utilizadas

A continuación se describen las herramientas que se han utilizado durante el desarrollo del proyecto junto con las funciones y utilidades que han aportado cada una de ellas.

- Astah Professional [35]: Herramienta de modelado UML utilizado para la creación de diagramas. de información.
- Excel [36]: Herramienta para la creación de gráficas y cálculos numéricos.
- Git [37]: Herramienta de control de versiones del código del proyecto.
- Gitlab [38]: Repositorio remoto para almacenar todo el código fuente y datos del proyecto.
- Google Chrome [39]: Navegador web desplegado para la utilización de Overleaf y búsqueda de información.
- Google Drive [40]: Utilizado para almacenar y compartir ficheros en la nube relacionados con el proyecto.
- jest [41]: Es un sistema de test automatizados para Javascript, utilizado para realizar pruebas de funcionamiento del sistema.
- Jitsi [42]: Sitio web utilizado para realizar las reuniones por videollamada con los tutores del proyecto.

ID	Riesgo	Probabilidad	Descripción	Retraso (días)
1	Enfermedad	0.4	El alumno no se encuentra en facultades óptimas para el desarrollo del trabajo	5
2	Horas extra por trabajo	0.1	El alumno debe utilizar horas del TFG en su trabajo actual	2
3	Problemas instalación entorno	0.4	Retraso debido a inconvenientes en la preparación e instalación de las herramientas que compondrán el entorno de trabajo	7
4	Problemas equipo de trabajo	0.1	Retraso debido a fallos o percances con el equipo de trabajo (PC, portátil...)	15
5	Pérdida de datos	0.2	Retraso debido a la pérdida de archivos o versiones de trabajo	7
6	Dificultad con las herramientas de trabajo	0.5	El alumno de utilizar días extra en el estudio, refuerzo y/o aprendizaje de aquellos mecanismos que le resulten desconocidos	7
7	Planificación inadecuada	0.7	Una mala planificación conlleva la repetición de iteraciones anteriores o un ajuste de fechas	15
8	Modificación de los requisitos	0.25	Conlleva realización de trabajo extra debido a la aparición de funcionalidad no prevista	15

Tabla 3.1: Análisis de riesgos

- MongoDB [33]: Base de datos utilizada para almacenar los datos de jugadores y partidas necesarios en el funcionamiento del matchmaking.
- Mozilla Firefox [43]: Navegador web utilizado para ver los resultados de las llamadas API al servidor, acceder a la base de datos y búsqueda.
- Nginx [44]: Servidor web de código abierto que es usado como proxy inverso, cache de HTTP, y balanceador de carga.
- Node.js [28]: Herramienta Javascript para el desarrollo del servidor de matchmaking y las llamadas API.
- npm [31]: Actúa en la gestión de las dependencias y versiones de los módulos de la API con Node así como en su ejecución.
- Overleaf [45]: Editor LaTeX utilizado para la escritura de la memoria del proyecto.

ID	Riesgo	Exposición al riesgo	Plan de reducción del riesgo	Plan de mitigación del riesgo
1	Enfermedad	2	Cumplir todos los protocolos sanitarios	Utilizar horas extra para recuperar el trabajo perdido
2	Horas extra por trabajo	0.2	Llevar las tareas del trabajo dentro de los plazos estimados	Utilizar horas extra para recuperar el trabajo perdido
3	Problemas instalación entorno	2.8	Lectura de manuales o tutoriales de instalación y despliegue	Utilizar documentación oficial y horas extra para recuperar el trabajo perdido
4	Problemas equipo de trabajo	1.5	Tener un correcto mantenimiento del equipo de trabajo y tenerlo en un lugar seguro	Conseguir un reemplazo del equipo o una reparación del mismo en el menor tiempo posible
5	Pérdida de datos	1.4	Realizar copias de seguridad, utilizar dispositivos de almacenamiento y de control de versiones	Recuperar la versión más reciente si es posible, o en su defecto, utilizar horas extra para recuperar el trabajo perdido
6	Dificultad con las herramientas de trabajo	3.5	Utilizar documentación oficial sobre las herramientas específicas antes de la realización de la tarea	Utilizar documentación oficial de las herramientas y dedicar horas extra que permitan recuperar el tiempo perdido
7	Planificación inadecuada	10.5	Realizar las estimaciones de tiempo de manera no optimista	Utilizar horas extra para recuperar el trabajo perdido
8	Modificación de los requisitos	3.75	Poner énfasis en la planificación a modo de cubrir todos los requisitos necesarios. Hacer el código de manera flexible que pueda permitir cambios no previstos	Utilizar horas extra para recuperar el trabajo perdido

Tabla 3.2: Plan de Acción

- Postman [34]: Herramienta para lanzar peticiones API sobre el servidor creado. Ha servido para realizar pruebas de funcionamiento.
- Python [46]: Lenguaje de programación utilizado para crear clientes necesarios para la simulación del funcionamiento y las llamadas del sistema.
- Telegram [47]: Aplicación de mensajería utilizado para la comunicación con los tutores.
- Visual Studio Code [48]: IDE sobre el que se ha programado el servidor de matchmaking.

Lós módulos de npm utilizados para el desarrollo del proyecto son:

- Axios [49]: Librería para la creación y envío de llamadas utilizado para el cambio de información entre sistemas en la simulación.
- csvtojson [50]: Librería que permite a convertir un fichero de formato .csv a formato JSON y con ello poder subir el contenido de dicho fichero a MongoDB.
- dotenv [51]: Biblioteca que permite la utilización de constantes por medio de un fichero .env.
- Express [32]: Librería que incorpora una serie de características para las aplicaciones Node.js que facilitan su escritura y ejecución.
- Mongoose [52]: Módulo que incorpora comandos que facilitan la conexión con MongoDB, así como establecer esquemas de diseño en las colecciones.
- Node-cron [53]: Biblioteca que desencadena un código concreto por medio de un timer.
- Nodemon[54]: Biblioteca que permite realizar cambios en el código del servidor sin necesidad de reiniciarlo, sino que se actualiza directamente.
- supertest [55]: Herramienta de pruebas para peticiones HTTP.

3.3.2. Entorno de trabajo

En esta sección se detallan los equipos utilizados durante el desarrollo del proyecto así como sus características de rendimiento.

Ordenador Portátil	
Lenovo Ideapad Gaming 3	
Hardware	
Procesador	AMD Ryzen 5 4600H with Radeon Graphics 3.00 GHz
RAM	16,0 GB
SSD	256 GB
Software	
Sistema Operativo	Ubuntu MIRAR
Arquitectura	64 bits

Tabla 3.3: Caracterísiticas del ordenador portátil utilizado

Monitor	
Samsung Syncmaster T200HD	
Resolución	1680 x 1050
Tamaño	20”
Ratio de Aspecto	16:9

Tabla 3.4: Características del monitor utilizado

Servidor	
Máquina virtual	
Hardware	
RAM	2GB
Procesador	Common KMV processor
Velocidad CPU	2.2GHz
HDD	10GB
Software	
Sistema operativo	Ubuntu 20.04.4 LTS
Arquitectura	64 bits

Tabla 3.5: Características de la máquina virtual utilizada para el despliegue

3.4. Estimación de costes

En esta sección se va a detallar la estimación de los costes del proyecto. Esto abarca desde el salario del desarrollador hasta los costes que tienen los equipos utilizados, las licencias necesarias y los servicios. Todos estos costes son calculados teniendo en cuenta el IVA.

3.4.1. Salario del trabajador

Desde el punto de vista real, el desarrollador no está contratado por la Universidad de Valladolid para la realización del proyecto, por lo que en términos verídicos el salario sería 0. Por ello, se realiza una estimación de lo que sería el coste si este proyecto es desarrollado por un programador junior a sueldo, el cual oscila en torno a los 18000 y 22000€ [56] . Tomando la media en 20000€ anuales, una jornada laboral de 8h y descasando fines de semana, el coste sería de unos 10.41€/h.

3.4.2. Espacio de trabajo

El proyecto se ha realizado de forma íntegra en el domicilio del estudiante, de forma que los costes obtenidos en esta sección serán los existentes por los servicios de abastecimiento de la vivienda.

Servicio	Coste mensual	Nº de meses	Horas diarias	Coste total
Luz	50€	8	2 h/día	33.33€
Gas	60€	8	2 h/días	40€
Internet	40€	8	2 h/día	26.66€
Comunidad	45€	8	2 h/día	30€
Total:				**130€**

Tabla 3.6: Tabla de estimación de costes del espacio de trabajo

3.4.3. Costes del hardware

En esta sección se calculará el coste como el precio de compra del dispositivo.

Dispositivo	**Coste de compra**	**Coste total**
Ordenador portátil	700€	700€
Monitor	200€	200€
Total:		**900€**

Tabla 3.7: Tabla de estimación de costes del hardware

3.4.4. Costes licencias

En esta sección se calculan los costes de las licencias de los programas y sistemas utilizados en los que es necesario realizar un pago para su uso. El programa Astah tiene una licencia gratuita para los estudiantes de la UVa [35], pero para este caso se estima el coste que tendría de no ser así [57].

Servicio	**Coste mensual**	**Nº de meses**	**Coste total**
Astah	4.33€	8	34.64€

Tabla 3.8: Tabla de estimación de costes de servicios y licencias

3.4.5. Costes totales

En la tabla 3.9 se puede ver el resumen de todos los costes detallados en las secciones anteriores.

Nombre	**Coste total**
Salario del trabajador	4705€
Espacio de trabajo	130€
Hardware	900€
Servicios y licencias	34.64€
Total:	**5769.64€**

Tabla 3.9: Tabla de estimación de costes totales del proyecto

3.5. Planificación final

En esta última sección se van a comparar los tiempos de la planificación inicial estimada con los tiempos reales una vez se ha terminado el proyecto al completo. Más concretamente, vamos a comparar las horas de trabajo, las semanas o las reuniones con los tutores.

Lo primero que cabe destacar es que la estimación inicial de horas de trabajo se quedó corta, ya que estaba calculada en 378 horas y ha acabado siendo de 452 horas totales. La principal fuente

de retraso ha sido durante el desarrollo del código del sistema ya que ha resultado más extenso y complicado para el alumno de lo estimado en la primera planificación, lo cual ha llevado un retardo de un mes con respecto a las horas planificadas en una primera instancia. También, a lo largo del desarrollo continuaron apareciendo funcionalidades a implementar o correcciones que retrasaban el paso a siguientes iteraciones. Durante el periodo de navidad, y en el transcurso de la iteración 3 de estudio de los diferentes tipos de matchmaking, las vacaciones hicieron que no fuera posible concertar reuniones con los tutores para mostrar el avance por lo que el alumno usó las horas para la redacción de la memoria, pero derivó en un retraso en esa iteración ya que no hubo visto bueno para avanzar a la siguiente hasta que las vacaciones no terminaron. A la vuelta del periodo vacacional, los tutores pidieron una segunda revisión del estudio del matchmaking propuesto y un informe adicional sobre él, lo cual derivó en otro retraso de un par de semanas. El alumno también sufrió COVID durante este periodo y ocasionó que durante unos días no se destinara tiempo al TFG. Debido a estos percances, se aplicaron los planes de mitigación de los riesgos que ya estaban previamente contemplados en la tabla 3.1. Estos planes de acción fueron las entradas 1, 6 y 7 de la tabla 3.2.

Los resultados de las pruebas de carga también ocasionaron que el tiempo a destinar al desarrollo completo del sistema no siguiera la planificación inicial y se necesitase aumentar en un par de semanas el trabajo para probar mecanismos que redujeran el tiempo de ejecución del sistema en el cálculo de puntuaciones ya que era excesivo y no cumplía con los requisitos establecidos al principio del proyecto. A mayores, debido a todos estos retrasos, el alumno se focalizó en el desarrollo del código y no pudo completar la redacción de la memoria una vez finalizado el sistema de emparejamiento diseñado, por lo que se destinó alrededor de un mes más a completarla. En todos estos casos se aplicó el plan de mitigación para el riesgo previsto de tener una mala planificación (ver en la tabla 3.1 el ID-7) que era aplicar horas extras de trabajo (ver en la tabla 3.2 el ID-7).

En cuanto a las iteraciones, el número real fue de 6. Los principales cambios con respecto a la planificación inicial fue la eliminación de la iteración 5 ya que el desarrollo de una interfaz de usuario para el matchmaking de la aplicación no formó parte de este TFG, hecho que sí está recogido en la planificación inicial debido a que el alumno pensó que era necesario, y el cambio de la iteración 7 destinado al despliegue del matchmaking diseñado, que fue incluido dentro de la iteración 6 de pruebas generales, a redacción de la memoria restante. El tiempo destinado a las dos primeras iteraciones cumplió lo previsto en la planificación inicial, incluso con algún ahorro de horas. La iteración 3 fue la que sufrió el primer retraso con respecto a lo planificado por los motivos comentados en el párrafo anterior. La iteración 4 también sufrió un retraso de unas 3 semanas aunque la eliminación de la iteración 5 referente al desarrollo de interfaces de usuario conllevó un adelanto de unas 4 semanas, por lo que se redujo el retraso total pero este seguía estando por encima de lo planificado. Por último, las últimas dos iteraciones siguieron los plazos previstos en cuanto a semanas pero las horas de trabajo fueron mayores cumpliendo con las medidas del plan de acción.

Durante todo el proceso se han llevado a cabo 15 reuniones con los tutores para mostrar los avances en el proyecto. Estas reuniones eran periódicas bisemanales salvo en casos excepcionales por vacaciones o imposibilidades. Siempre hubo una reunión para dar el visto bueno a la anterior iteración y comenzar con la siguiente. Las iteraciones 3 y 4 fueron las que más reuniones necesitaron llevarse a cabo, con cuatro reuniones cada una. Para las demás fueron entre una y dos reuniones y para el proce-

so de redacción de la memoria se realizaron la mayoría de las consultas por Telegram y solo existieron reuniones en caso de revisiones muy concretas más la final.

En general, viendo del desarrollo del proyecto y comparándolo con la planificación inicial podemos concluir que las horas y las semanas destinadas han superado a las estimadas en un primer momento. En las figuras 3.3 y 3.4 se puede ver gráficamente la distribución de las horas y las semanas en las diferentes iteraciones acometidas y su comparación con las previstas.

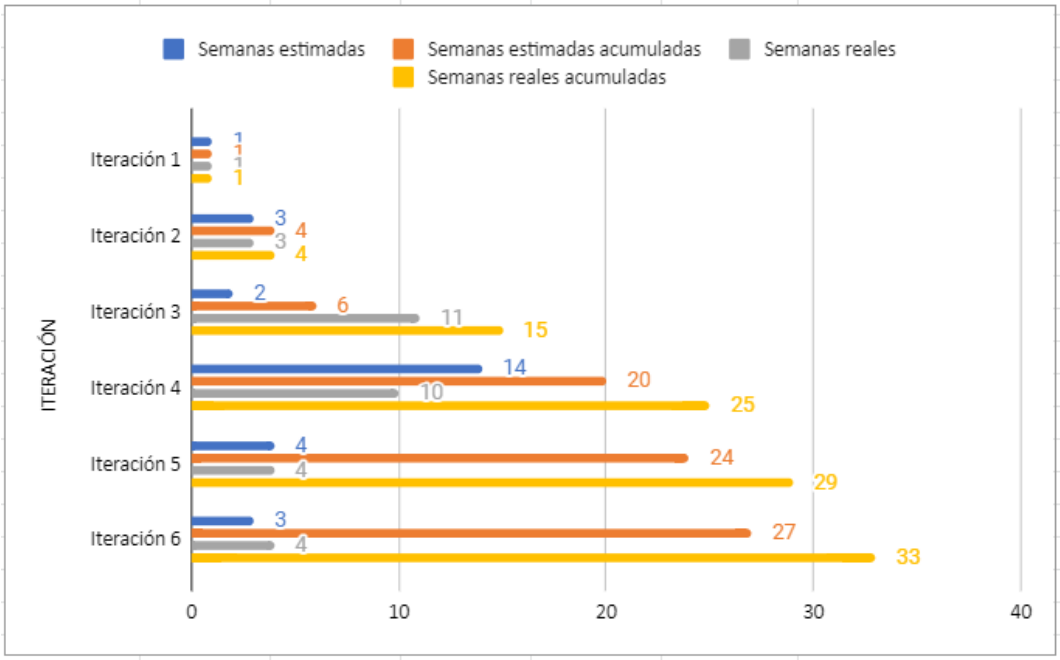

Figura 3.3: Gráfica comparativa de las semanas reales con las estimadas

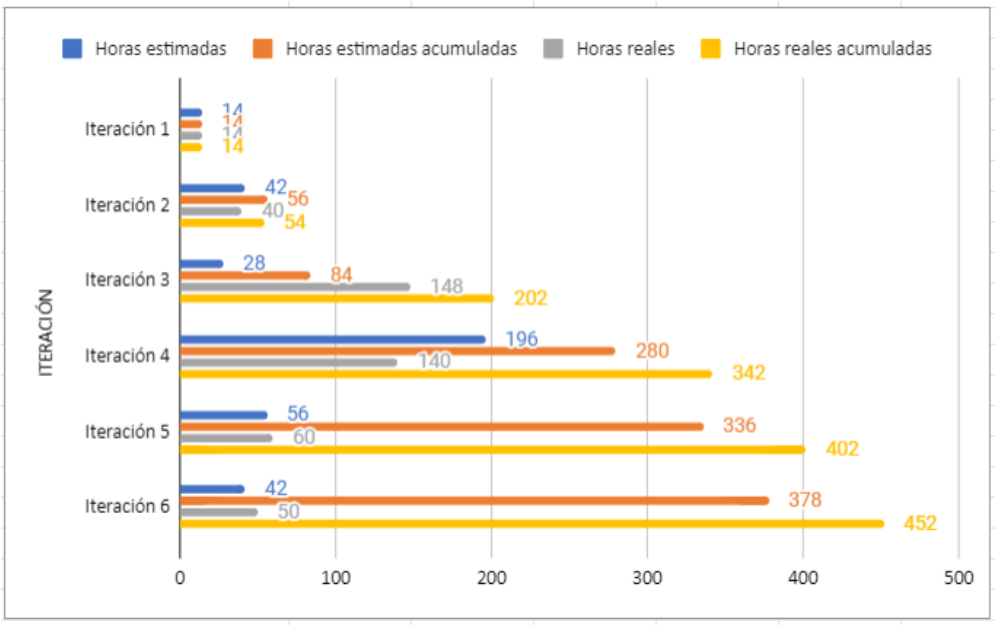

Figura 3.4: Gráfica comparativa de las horas reales con las estimadas

Capítulo 4

Descripción de las iteraciones

En esta sección se van a detallar las acciones realizadas en cada iteración del desarrollo del proyecto, lo cual engloba el estudio de las tecnologías y los criterios de matchmaking, el maquetado del prototipo inicial, las implementaciones, el desarrollo del sistema y las pruebas finales.

4.1. Iteración 1

El inicio del proyecto nació con una reunión entre el alumno y los tutores de este TFG en el que se expusieron las características generales del proyecto, en qué consistía, el alcance, los requisitos más importantes y los primeros pasos a seguir.

4.1.1. Requisitos

4.1.1.1. Requisitos funcionales

A continuación se describen los requisitos funcionales mencionados en la reunión inicial. Los requisitos funcionales son aquellos que describen una funcionalidad que el sistema a desarrollar debe implementar. En la Tabla 4.1 se listan estos requisitos funcionales con su ID, el nombre, una breve descripción y un nivel de prioridad (Alta, Media o Baja).

ID	Nombre	Descripción	Prioridad
RF01	Buscar partida	El sistema permitirá a un usuario buscar partida	Alta
RF02	Devolver lista de rivales	El sistema devolverá al usuario una lista con los rivales obtenidos en todo momento	Alta
RF03	Registrar jugador	El sistema permitirá crear y almacenar los datos de un nuevo jugador recién registrado	Alta
RF04	Buscar partidas equilibradas	El sistema se encargará de que los rivales idóneos sean de nivel similar al del jugador solicitante	Alta
RF05	Calcular puntuaciones	El sistema será capaz de calcular las nuevas puntuaciones para cada jugador en función de las partidas disputadas	Alta
RF06	Actualizar datos resultantes	El sistema almacenará y actualizará a cada jugador con los datos resultantes del cálculo	Alta
RF07	Crear partidas de varios jugadores	El sistema será capaz de procesar las partidas que involucren a más de un jugador	Media
RF08	Incluir varios modos	El sistema deberá diferenciar los datos y los jugadores por cada modo de juego existente	Alta
RF09	Poblar jugadores mediante fichero	El sistema permitirá actualizar la lista de jugadores mediante el contenido de un fichero	Baja

Tabla 4.1: Requisitos funcionales

4.1.1.2. Requisitos no funcionales

En esta sección se listan los requisitos no funcionales obtenidos durante la reunión, los cuales están descritos en la tabla 4.2 y siguen el mismo formato que los requisitos funcionales. Este tipo de requisitos son aquellos que describen funcionalidades técnicas y restricciones que definen cómo se debe comportar el sistema.

ID	Nombre	Descripción	Relevancia
RNF01	Multiplataforma	El sistema se puede utilizar en sistemas operativos diferentes	Deseable
RNF02	Almacenamiento en base de datos	El sistema almacenará los datos de jugadores y partidas en la base de datos MongoDB	Crítica
RNF03	Formato de parámetros	El sistema utilizará el formato JSON para todos los parámetros de llamadas entre partes	Crítica
RNF04	FormatoBD	El sistema almacenará los datos en la base de datos en formato JSON	Crítica
RNF05	Notificación de errores	El sistema informará al usuario de producirse algún error por medio de un mensaje o código	Crítica
RNF06	Adaptabilidad	El sistema estará diseñado buscando un esquema genérico que pueda ser implementado en otras aplicaciones educativas	Crítica
RNF07	Tiempo de búsqueda reducido	El sistema será capaz de buscar partida en un intervalo de tiempo aceptable para evitar demoras que disminuyan la satisfacción del usuario (5 segundos máximo)	Crítica
RNF08	Tiempo de cálculo reducido	El sistema será capaz de realizar el cálculo de puntuaciones en un intervalo de tiempo que no suponga un impacto negativo en la experiencia de juego del usuario	Crítica
RNF09	Cálculo de puntuaciones mediante Glicko	El sistema calculará las nuevas puntuaciones por medio del algoritmo Glicko	Crítica
RNF10	Facilidad de instalación	El sistema podrá ser instalado siguiendo los pasos del manual de instalación	Deseable

Tabla 4.2: Requisitos no funcionales

4.1.1.3. Requisitos de información

Por último, en esta iteración se describen los requisitos de información que son aquellos que especifican los datos que se deben guardar en el sistema. Estos requisitos se encuentran en la tabla 4.3. Además, en esta reunión se comentan y se acuerdan las diferentes tecnologías a utilizar durante el proyecto en función de la propuesta y los conocimientos del alumno. Finalmente, se realiza una planificación inicial y sus respectivas iteraciones con lo hablado durante la reunión.

ID	Nombre	Descripción
RI01	Datos del jugador	Los datos del jugador a almacenar son: ▪ id_usuario ▪ ptosELO ▪ RD ▪ periodosSinJugar ▪ id_modo
RI02	Datos de las partidas	Los datos de la partida a almacenar son: ▪ id_partida ▪ id_usuario ▪ resultado ▪ id_modo ▪ finalizada ▪ fecha

Tabla 4.3: Requisitos de información

4.2. Iteración 2

En esta segunda iteración se realiza un primer prototipo básico entre las diferentes entidades y se desarrollan las comunicaciones entre ellas a modo de ayudar a familiarizarse con las tecnologías a emplear y tener una base sobre la que implementar las demás mejoras y funcionalidades del proyecto final.

Este prototipo básico, el cual se desarrolló siguiendo un modelo de API REST [58], constó de un cliente sencillo y dos servidores desarrollados en Node.js: el Servidor App que es el encargado de simular el servidor que sustenta la aplicación, y el Servidor MM que es el que implementará el matchmaking que se va a utilizar. El funcionamiento de la simulación era una petición POST con un mensaje ”HolaMundo” lanzada por el cliente a uno de los servidores, el cual al recibirla creaba otra del mismo tipo y la enviaba al segundo servidor. El segundo servidor procesaba la llamada y devolvía un nuevo mensaje como respuesta a esa petición del primer servidor y este como respuesta al cliente.

De esta forma se realizaba una primera toma de contacto con las comunicaciones entre las diferentes partes que componen el flujo a fin de probar las sensaciones con las tecnologías y encontrar posibles dificultades que hicieran rectificar las herramientas a utilizar en una fase pronta del proyecto. En estas dificultades encontradas se optó por desarrollar el cliente en lenguaje Python en lugar de usar exclusivamente Node.js, así como se probaron diferentes librerías para la creación y envío de las

peticiones, optando por utilizar ”requests” para el cliente y ”axios” para los servidores.

En la figura 4.1 se puede ver un esquema del prototipo inicial diseñado y como es el paso de mensajes y la comunicación entre ellos.

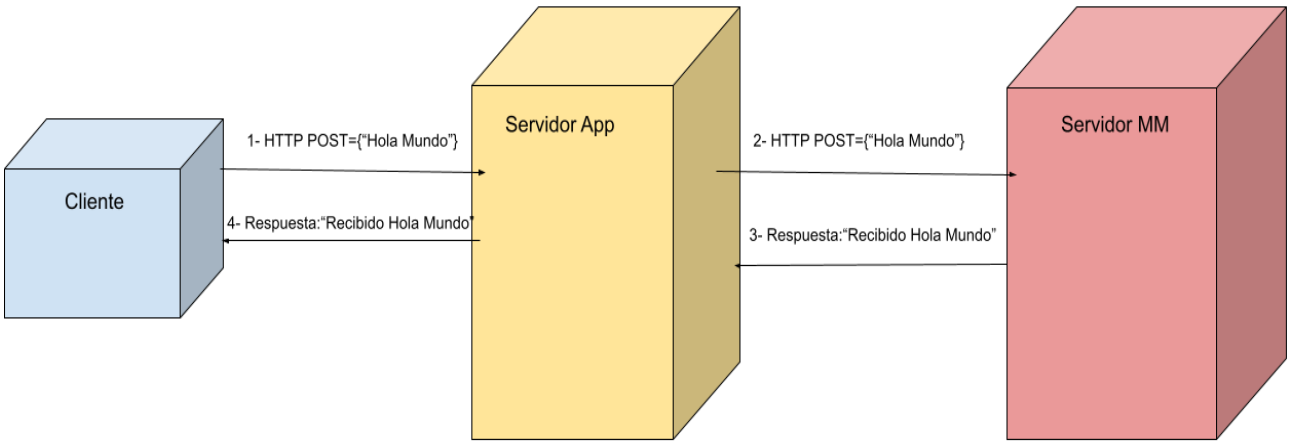

Figura 4.1: Esquema conceptual del primer prototipo desarrollado con sus componentes y mensajes

4.3. Iteración 3

En esta tercera iteración se llevó a cabo un estudio exhaustivo sobre el propósito de este TFG, en este caso, los diferentes tipos de matchmaking existentes, sus características y la forma de implementarlo. Para ello, el alumno leyó una serie de artículos referentes al tema en cuestión [15, 19, 16] y probó de primera mano una serie de aplicaciones móviles a fin de ver los matchmaking que tenían implementados y realizar una tabla comparativa que ayudara a escoger el más apropiado para nuestra aplicación. Estas tablas están recogidas en el apéndice C.

En esta iteración también se realizaron pruebas sobre el prototipo inicial creado en la iteración anterior. Las pruebas consistieron en aplicar concurrencia y estudiar el comportamiento del flujo de mensajes. Esta prueba debía simular como si una cantidad grande de jugadores deseasen buscar partida al mismo tiempo y nuestro sistema debía estar capacitado para atender todas las llamadas independientemente, sin dar lugar a bloqueos y sin que este sufriera un retardo excesivo en sus operaciones. También se aplicó el código necesario para que todas las llamadas se realizaran de forma asíncrona para así evitar interrupciones y flujo secuencial que se alejara de la realidad del propósito final del proyecto.

Como principal inconveniente, durante esta iteración se perdió el prototipo inicial ya creado debido a un error informático. Debido a esto, el alumno debió replicarlo de nuevo desde cero y realizar pasos ya acometidos en la iteración 2.

4.4. Iteración 4

En esta iteración y tras haber elegido definitivamente el tipo de matchmaking a implementar así como haber estudiado en profundidad su funcionamiento, se lleva a cabo el desarrollo del código que dote de la funcionalidad especificada a nuestro prototipo. Para ello, se optó por una metodología incremental de tal forma que se fueran añadiendo partes de funcionalidad de forma progresiva, probando cada una de ellas antes de pasar a la parte siguiente y poder corregir errores sin complicación debido a la extensión de código.

En primer lugar se implementó el flujo de llamadas que haría la búsqueda de partidas de un jugador concreto, el cual viene detallado en la figura 4.2.

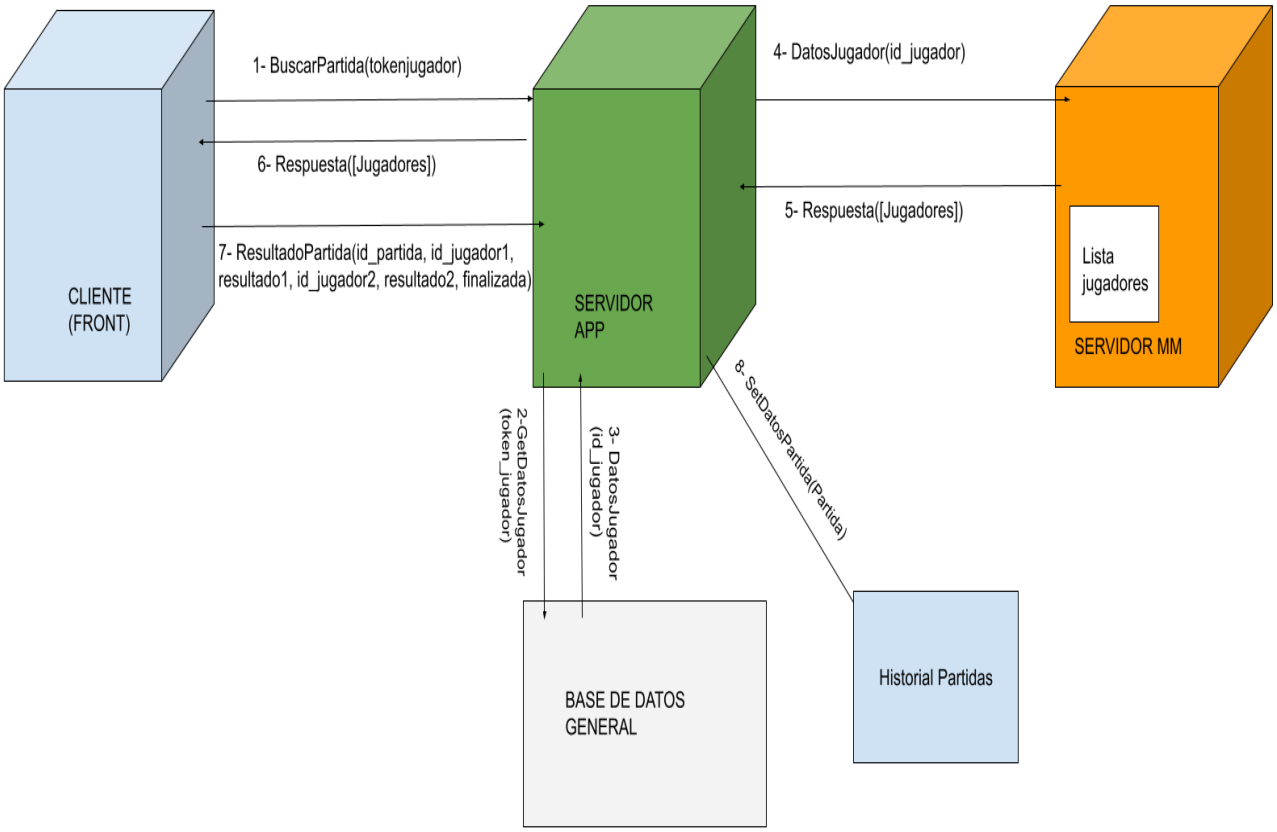

Figura 4.2: Esquema de la primera versión del paso de mensajes entre las entidades para el caso de un jugador que busca partida

El cliente solicita buscar partida y envía una petición POST al Servidor App con sus datos de usuario, en este caso, un token. El Servidor App procesa dicha llamada y coteja el token con un fichero que simula la base de datos de la propia aplicación COP para obtener el $id\_jugador$, el cual se pasa como parámetro al servidor de matchmaking y que este pueda comenzar el proceso de elección y emparejamiento con los rivales apropiados.

Una vez el servidor de matchmaking recibe la llamada con el id del jugador solicitante, este obtiene de un fichero los datos de este usuario necesarios para los criterios establecidos de matchmaking. En este caso son los puntos ELO, periodos sin jugar y el nivel. A través de comparaciones con los datos de

los demás usuarios registrados, se obtiene un array con todos los candidatos, el cual se devuelve al Servidor App y este a su vez al cliente. En este momento es donde el jugador seleccionaría rival y tendría lugar la partida entre ambos, pero queda fuera de nuestra simulación, por lo que solo se simulan los resultados. En este punto, y simulando el final de la partida y el registro de los resultados, se realiza desde el cliente una segunda llamada al Servidor App, pasando como parámetros los datos relevantes de la partida (en este caso: $id\_partida, [id\_jugador_i, id\_jugador_j, ...], [resultado_i, resultado_j, ...], finalizada$). El Servidor App procesa la llamada e inserta dichos datos en un fichero que servirá a modo de historial para el posterior cálculo de puntuaciones. Además, en esta parte se programa el código y el flujo de llamadas para el cálculo de puntuaciones a final de un periodo.

Para este proceso se ha utilizado un job periódico que activa el código encapsulado cada cierto espacio de tiempo que le indicamos. Una vez desencadenado el cálculo de las puntuaciones, en primer lugar se obtienen los jugadores que deben ser actualizados a través del parámetro ”actualizar” incluido en el fichero ”lista_jugadores”. Después, se realiza una llamada al Servidor App para obtener el fichero con el historial de partidas de este periodo. A continuación, para cada uno de los jugadores a actualizar, se obtienen por medio del historial las partidas en las que ha estado involucrado, su resultado de cada una de ellas y el $id\_jugador$ de los rivales a los que se ha enfrentado. Con el id de los rivales, es necesario obtener de cada uno los datos de matchmaking (ptosELO, RD y periodos sin jugar).

Una vez se han obtenido todos los parámetros, tiene lugar el cálculo de las nuevas puntuaciones por medio de las fórmulas pertenecientes al algoritmo Glicko anteriormente visto en el capítulo 2. El resultado es un array con los nuevos datos de cada jugador, que posteriormente será utilizado para actualizar la lista de jugadores de cara a los periodos futuros y los emparejamientos dentro de él. El flujo de mensajes entre las partes para el cálculo de puntuaciones se puede ver en la figura 4.3

Toda esta sección finalizó con las pruebas de funcionamiento donde se comprobó la obtención de los datos esperados. Todas fueron exitosas excepto las partidas que involucraban a más de dos jugadores (1 vs varios), para las cuales el algoritmo Glicko no estaba pensado, por lo que quedó pendiente un estudio de una posible solución; además del proceso de obtención de los jugadores que no debían de ser actualizados que no era correcto.

En una segunda iteración, se implementaron mejoras y correcciones a fin de optimizar código y hacerlo más simple y ajustado a la realidad de la cuestión. Para ello, primero se cambió la forma de obtener a los jugadores que debían ser actualizados durante el proceso de cálculo de puntuaciones. Se eliminó el atributo ”actualizar” de la lista de jugadores y la obtención de dichos jugadores se realizó reuniendo los id de los participantes en el historial de partidas. De esta forma las llamadas para modificar el atributo ”actualizar” cada vez que finalizaba una partida dejaron de ser necesarias.

Se almacenaron en el historial las partidas finalizadas y no finalizadas, eliminando un filtro que hacía que solo se almacenaran las finalizadas. Este filtro se aplicó posteriormente en el cálculo de puntuaciones.

Se implementó una posible solución para resolver el conflicto con las partidas que implicaban varios jugadores. Como el problema radicaba en que Glicko solo estaba pensado para partidas 1 vs 1, esta solución consistía en dividir las partidas 1 vs varios en varias partidas de 1 vs 1 (por ejemplo, si una

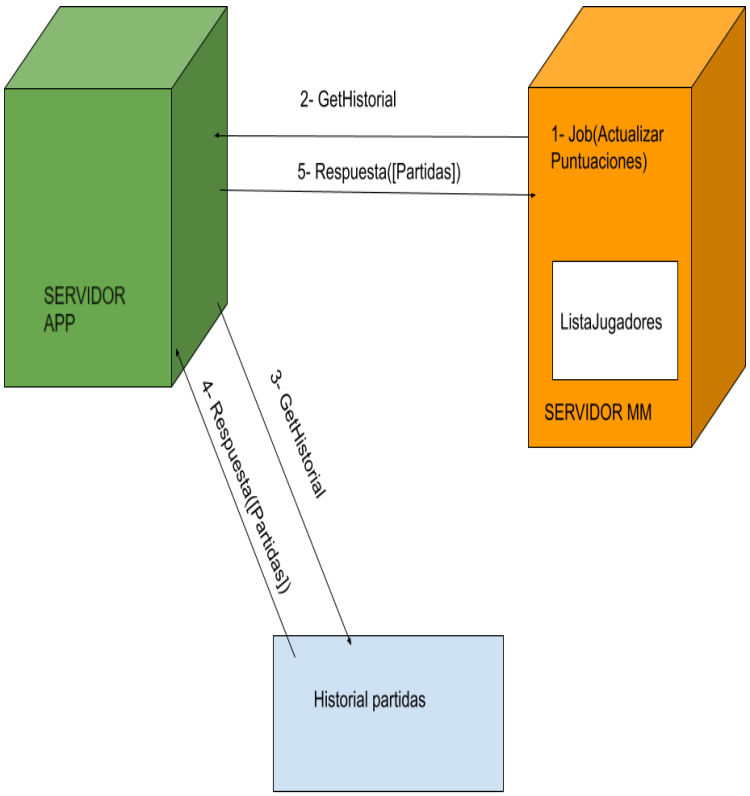

Figura 4.3: Esquema de la primera versión del paso de mensajes entre las entidades para el cálculo de puntuaciones

partida implicaba a los jugadores 1, 2 y 3, se parseaban los datos de tal forma que resultara en 3 partidas: 1 vs 2, 1 vs 3 y 2 vs 3) y después aplicar un peso para que las nuevas puntuaciones fueran las resultantes de ganar una única partida y no varias.

A modo de optimización se aplicaron funciones y la utilización de un fichero de variables de entorno utilizado para constantes, el cual dio problemas pero quedó planteada una versión inicial.

Una vez terminada la iteración anterior y después de que casi la totalidad del código funcionaba correctamente, se optó por introducir la base de datos y sustituir los ficheros que complicaban el código debido a la forma de extraer su contenido de manera correcta. La base de datos elegida fue MongoDB.

Antes de su implementación, se realizó una pequeña prueba de instalación y utilización a modo de aprender su funcionamiento, así como un modelo de dominio para ver el formato que debía tener nuestra colección. Después, comenzó la fusión del código con la base de datos. Para ello utilizamos el módulo ”mongoose”, el cual, diseñado especialmente para Node.js, otorga una serie de comandos y esquemas que facilitan la comunicación entre código y colección.

Una vez integrado nuestro código con la base de datos, comenzó la sustitución de los ficheros y la inserción de consultas directas con las que se realizaba la misma funcionalidad pero de manera más óptima y lógica. Algunos ejemplos de estas consultas se pueden ver en la figura 4.4

Una actualización importante en esta parte del desarrollo del matchmaking fue la incorporación

```
await HistorialPartidas.deleteMany()
await ListaJugadores.deleteMany()
ListaJugadores.insertMany(nuevos_datos_totales)
```

Figura 4.4: Fragmento de código de alguna consulta utilizada para sustituir el uso de ficheros

de los modos de juego al flujo del programa. Esto repercute en que cada jugador tendrá unos datos diferentes para cada modo de juego que exista en la aplicación, un usuario tendrá que especificar el modo cuando solicite buscar partida, la extensión de la base de datos aumenta y las partidas también irán asociadas a un modo determinado. En esta parte se implementaron también dos llamadas más: /registrarJugador, /poblarJugadores:

- /registrarJugador: Esta llamada permite registrar un jugador en la lista de jugadores para que pueda ser emparejado. Esta llamada surge debido a que con el flujo de código actual y la forma de actualizar la lista de jugadores, un usuario que se registrara durante un periodo nunca podría aparecer en la base de datos.
- /poblarJugadores: Estas llamadas utilizan un fichero con extensión .csv para poblar su documento de la base de datos de golpe. Estas llamadas son utilizadas de posible backup ante una pérdida de datos o para realizar pruebas con entradas concretas. Para la implementación del funcionamiento de estos endpoints se ha utilizado el módulo ”csvtojson” el cual realiza la conversión de formato csv a JSON para su posterior volcado a la base de datos y que esta lo haga correctamente sin errores de parseo.

Otras implementaciones en esta fase de desarrollo han sido la eliminación del atributo ”nivel” como criterio para buscar rivales en el matchmaking, realizar el cálculo de las nuevas puntuaciones para cada modo de juego existente y una mayor optimización de las consultas de búsqueda e inserción en la base de datos.

En la última iteración, se implementan las últimas mejoras y la corrección definitiva al gran problema del flujo: las partidas 1 vs varios.

La versión anterior para este problema quedó pendiente a falta de encontrar un peso adecuado con el que las puntuaciones resultantes fueran acordes al algoritmo. Debido a que no se encontró, se optó por un enfoque distinto, el cual consistía en obtener los datos de todos los rivales de esa partida y calcular la media de todos ellos. De esta forma obtenemos un único jugador con los datos medios de los jugadores y se cumple que la partida sea 1 vs 1, por lo que aplicando el algoritmo tal cual, los datos son correctos (ver figura 4.5). Además, se arregla definitivamente el fichero de *enviroments* para el uso de constantes y se agregan *middlewares* y clases independientes para cada *endpoint* del programa (ver figura 4.6).

En este punto y tras una revisión exhaustiva se encuentran puntos de mejora que se intentan implementar. Uno de ellos es añadir un atributo fecha a las partidas de manera que puedan diferenciarse

```
for (var j = 0; j < buscaPartida.length; j++){      //Se recorre cada partida para obtener los datos de los rivales
    var buscaRivalesAux = await HistorialPartidas.find({$and:[{finalizada:1},{id_modo : indiceModo},{id_partida : buscaPartida[j].id_partida},
        {id_jugador:{$ne: jugadores_actualizar[i]}}]}, {id_jugador:1})
    ptosELOSum = 0
    RDSum = 0
    for (var k = 0; k < buscaRivalesAux.length; k++){       //Se recorre cada rival de una partida acumulando sus puntos ELO y RD
        datos_rival = await ListaJugadores.find({$and:[{id_modo: indiceModo}, {id_jugador: buscaRivalesAux[k].id_jugador}]},{ptosELO: 1, RD: 1})
        ptosELOSum = ptosELOSum + parseInt(datos_rival[0].ptosELO)
        RDSum = RDSum + parseInt(datos_rival[0].RD)
    }
    //Se calcula la media de los datos de los rivales
    ptosELOMedia = ptosELOSum / buscaRivalesAux.length
    RDMedia = RDSum / buscaRivalesAux.length
    var rivalDef = {
        "ptosELO": ptosELOMedia,
        "RD": RDMedia
    }
    datos_rivales.push(rivalDef)
}
```

Figura 4.5: Fragmento de código que muestra el cálculo de las puntuaciones medias de todos los participantes de una misma partida

```
//routes
app.use("/buscarPartida", require("./routes/buscarPartida"))
app.use("/poblarjugadores", require("./routes/poblarJugadores"))
app.use("/registrarjugador", require("./routes/registrarJugador"))
```

Figura 4.6: Fragmento de código que muestra los *middleware* utilizados para dirigir las llamadas a las clases independientes que las contienen

cuales pertenecen a un periodo concreto. De esta forma, no es necesario borrar el historial de partidas cada vez que comienza un nuevo periodo, sino que por medio de este atributo fecha se puede obtener el subconjunto de partidas que nos hacen falta calcular las nuevas puntuaciones.

También, durante esta etapa, se intenta implementar paralelismo para reducir el tiempo de ejecución del flujo del cálculo de puntuaciones pero por falta de tiempo y ante la dificultad de un correcto funcionamiento se opta por dejarlo como trabajo futuro.

A mayores, se optimiza más el código con más funciones y añadiendo más constantes al fichero *.env*, además de eliminar las llamadas ”/registrarJugador” y ”/poblarJugadores” del Servidor App y que el cliente se comunicara directamente con el Servidor MM. Esta última modificación se realiza para descargar de funcionalidad innecesaria al Servidor App.

El esquema final del flujo de mensajes entre las partes que conforman el sistema se puede ver en las figuras 4.7 y 4.8.

4.5. Iteración 5

Esta iteración comienza en cuanto el desarrollo de la funcionalidad del Servidor MM ha quedado finalizado y las pruebas para comprobarlo han resultado exitosas.

En este punto se pasaron a realizar pruebas de carga para observar el comportamiento del sistema cuando el número de usuarios y partidas era muy elevado y, en caso de tener un rendimiento bajo en

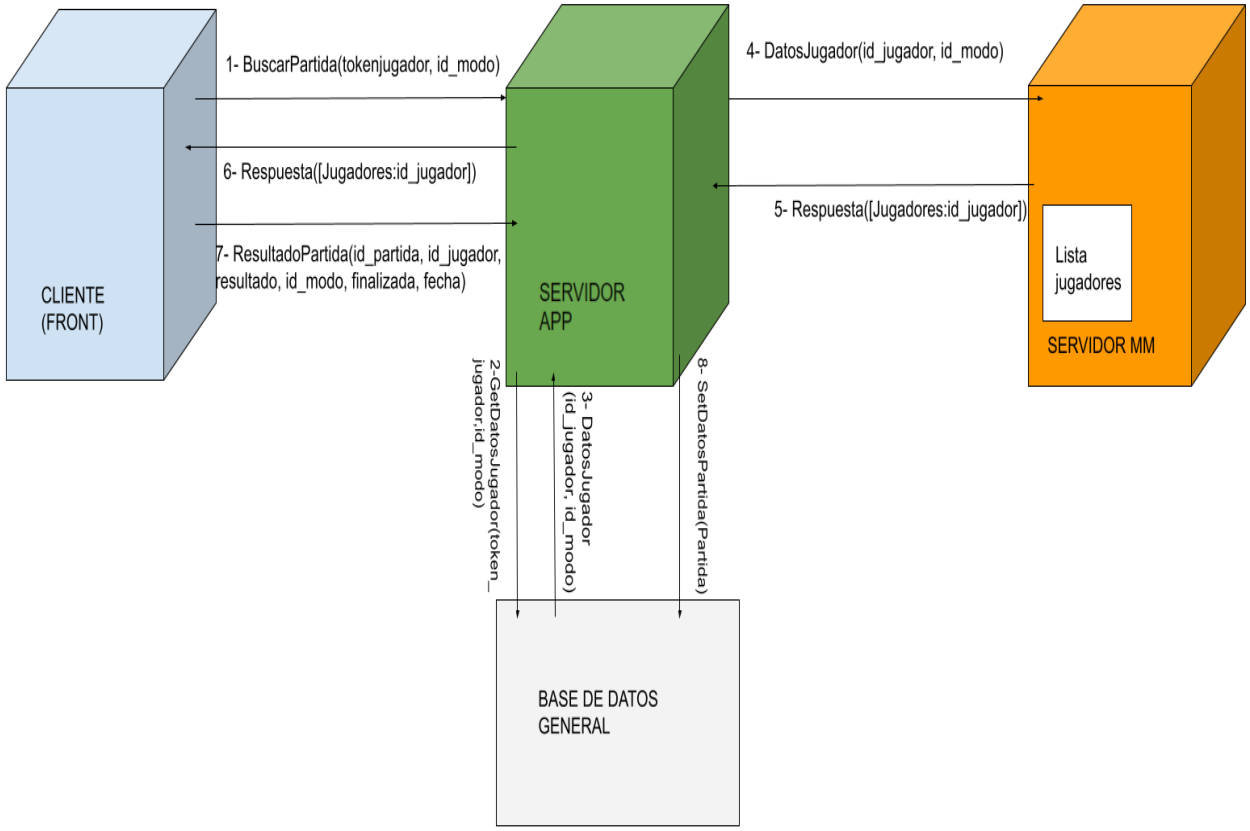

Figura 4.7: Esquema de la primera versión del paso de mensajes entre las entidades para el cálculo de puntuaciones

términos de tiempo, optimizar el código y realizar mejoras donde fuese pertinente. Para estas pruebas se utilizaron una serie de timers que se encargaron de medir tanto el tiempo de ejecución del flujo de programa total como de medir el tiempo del código por secciones a fin de controlar en qué zonas la pérdida era mayor. Además, se optó por un enfoque incremental, de tal forma que el número de jugadores y partidas fuese escalando progresivamente.

Como resultado se pudo apreciar que para una carga alta de jugadores el tiempo del cálculo de las nuevas puntuaciones era excesivo, por lo que en esta iteración se implantó una mejora sustancial con la utilización de un fichero auxiliar que almacenara las puntuaciones de cada jugador en el mismo bucle en el que se calculan, en lugar de utilizar un bucle más que simplemente modificara las puntuaciones de la base de datos, reduciendo así el número de iteraciones y por consiguiente el tiempo de ejecución del flujo.

En esta iteración también se realizaron los despliegues y configuraciones en las máquinas virtuales cedidas por la Escuela de Ingeniería Informática y las respectivas pruebas de funcionamiento. Los pasos seguidos durante el despliegue están recogidos en el manual de despliegue del Apéndice D.

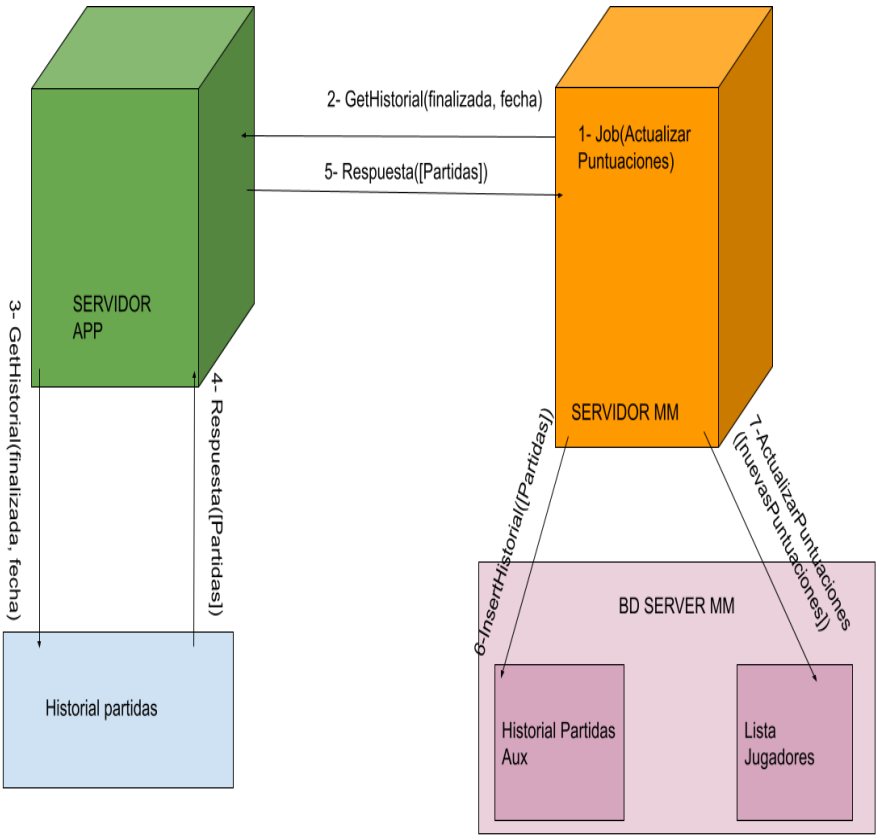

Figura 4.8: Esquema de la primera versión del paso de mensajes entre las entidades para el cálculo de puntuaciones

4.6. Iteración 6

Esta última iteración ha consistido en la redacción restante de la memoria que faltaba para completarse y se han hecho las correcciones necesarias después de las reuniones y las revisiones de los tutores para mejorar el documento final.

Además, en esta etapa también se ha realizado la entrega del TFG después de haber tenido el visto bueno de los tutores.

Capítulo 5

Estado final del sistema

Este TFG ha sufrido varias etapas hasta llegar a su estado final. En primer lugar existió una primera toma de contacto con los objetivos del proyecto y su propósito, después un estudio previo de los tipos de matchmaking existentes mediante una comparativa de varias aplicaciones educativas, para a continuación, comenzar con el desarrollo del Servidor MM que implementara el sistema de emparejamiento escogido. Finalmente, tras finalizar el desarrollo del servidor, se realizaron las pruebas de funcionamiento y de rendimiento para comprobar si se realizaba de manera estable y los tiempos de ejecución eran aceptables. El esquema conceptual del proceso seguido en este TFG se encuentra en la figura 5.1.

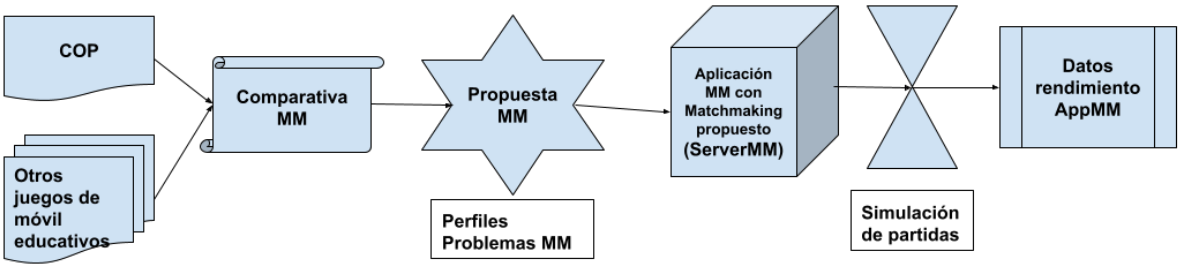

Figura 5.1: Esquema conceptual del proyecto

5.1. Diagramas de análisis

5.1.1. Casos de uso

En esta sección se detallarán los diferentes casos de uso del proyecto. En la figura 5.2 está dibujado cada uno de ellos relacionados con el actor que los inicia. Cabe destacar que el caso de uso de ”Calcular Puntuaciones” lo lanza el propio sistema de forma automática por medio de un *cronjob* de Node llamado *node-cron* [53]. Se trata de un módulo de Node.js que establece un temporizador que indica cuando se ejecuta el flujo de código que engloba. Todas estas llamadas son independientes y no interactúan unas

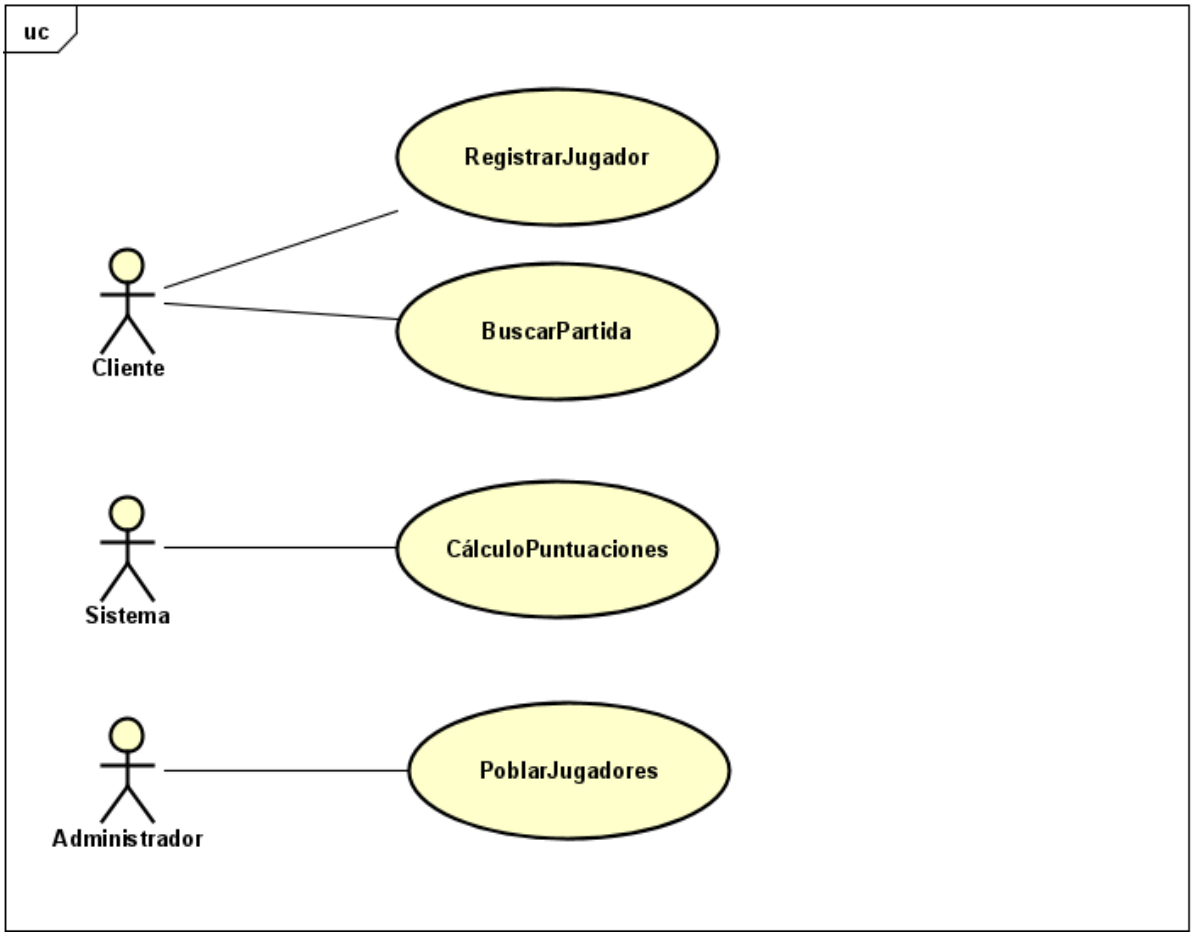

Figura 5.2: Diagrama de casos de uso del sistema

con otras.

Caso de uso	CU01
Nombre	RegistrarJugador
Dependencias	RF03
Actor principal	Cliente
Actor secundario	Ninguno
Descripción	Un usuario desea poder registrar sus datos en la lista de jugadores para poder emparejarse con otros jugadores
Precondiciones	1. Estar conectado a internet
Postcondiciones	Ninguna
Excepciones	Ninguna
Frecuencia	Media

Tabla 5.1: Caso de uso CU01

Cabe destacar que existe una llamada adicional pero que no se aplica en el Servidor MM como las demás sino en el servidor propio de la aplicación y que en nuestro sistema se ha utilizado principalmente para pruebas y comprobar los resultados con una entrada de datos concreta. Esta llamada es

Caso de uso	CU02
Nombre	BuscarPartida
Dependencias	RF01, RF02, RF04. RF07
Actor principal	Cliente
Actor secundario	Ninguno
Descripción	Un usuario quiere buscar partida y emparejarse con jugadores de nivel similar y que las partidas jugadas se almacenen en la base de datos
Precondiciones	1. Estar conectado a internet 2. Estar registrado en el sistema
Postcondiciones	Ninguna
Excepciones	Ninguna
Frecuencia	Alta

Tabla 5.2: Caso de uso CU02

Caso de uso	CU03
Nombre	PoblarJugadores
Dependencias	RF09
Actor principal	Administrador
Actor secundario	Ninguno
Descripción	Un usuario desea poder poblar la lista de jugadores por medio de un fichero mediante una única operación
Precondiciones	1. Estar conectado a internet 2. Tener el fichero en el formato adecuado
Postcondiciones	Ninguna
Excepciones	Ninguna
Frecuencia	Media

Tabla 5.3: Caso de uso CU03

la de ”Poblar Partidas”, la cual puebla automáticamente el historial de partidas de la base de datos del Servidor App con las partidas que contiene un fichero .csv que se le pasa como parámetro.

5.1.2. Modelo de Dominio

En la figura 5.3 se puede ver el diagrama de clases en el que se representa el modelo de dominio que sigue el sistema desarrollado.

Clase: Player

- Descripción: Clase que modela los datos de un jugador registrado en el sistema.

Caso de uso	CU04
Nombre	CalcularPuntuaciones
Dependencias	RF05, RF06, RF08
Actor principal	Sistema
Actor secundario	Ninguno
Descripción	Un usuario Desea que las puntuaciones y periodos sin jugar de todos los jugadores existentes se actualicen en función de su participación durante el periodo
Precondiciones	1. Estar conectado a internet
Postcondiciones	Ninguna
Excepciones	Ninguna
Frecuencia	Alta

Tabla 5.4: Caso de uso CU04

- Responsabilidades: Modelar un jugador con sus datos de registro y datos Glicko de forma que pueda ser identificado y modificado por las operaciones del sistema.
- Atributos:
  - eloPoints: Número que representa el nivel de habilidad del usuario utilizado para emparejar.
  - rd: Grado de incertidumbre del jugador que relaciona el nivel de habilidad con el tiempo sin jugar.
  - intervalsWithoutPlay: Número de periodos en los que el jugador no ha disputado ninguna partida.
  - mode: Identificador del modo de juego al que pertenecen los datos del jugador.

Clase: Match

- Descripción: Clase que modela los datos de una partida disputada por un jugador.
- Responsabilidades: Modelar una partida disputada por un jugador con todos los datos relevantes para posteriores registros y cálculos.
- Atributos:
  - mode: Número identificativo del modo en el que se ha disputado la partida.
  - finish: Entero que indica si la partida ha llegado a finalizar o no.
  - date: Día y hora en la que se ha disputado la partida.

Clase: Matchmaking

- Descripción: Clase que representa el mecanismo de emparejamiento entre jugadores.

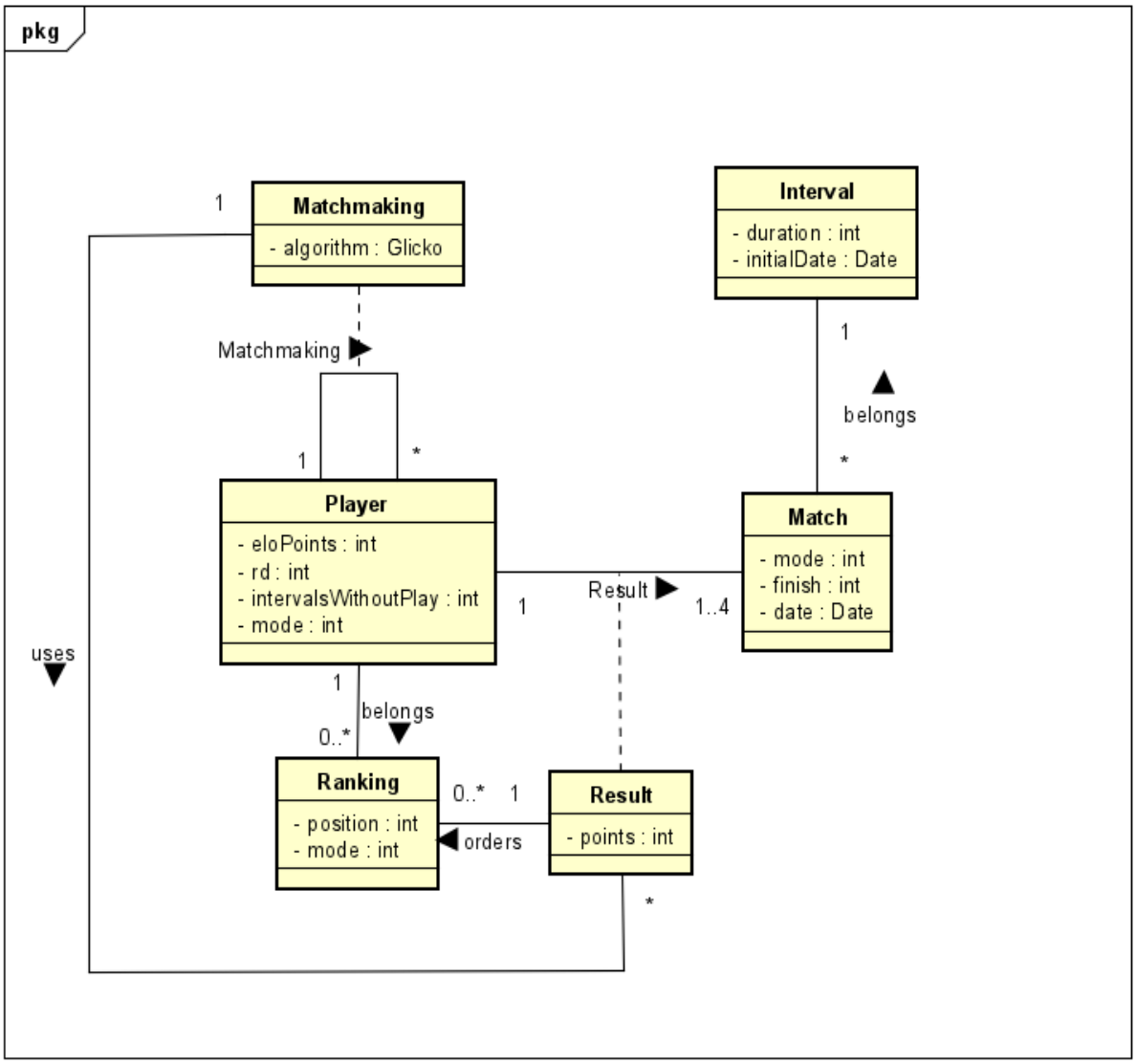

Figura 5.3: Modelo de dominio del sistema

- Responsabilidades: Modelar la entidad que representa el algoritmo con el cual los jugadores son emparejados en función de su habilidad cuando se solicita partida.
- Atributos:
  - algorithm: Conjunto de fórmulas que componen el algoritmo Glicko.

Clase: Result

- Descripción: Clase que modela los atributos del resultado de una partida.
- Responsabilidades: Modelar el resultado obtenido de un jugador tras el desarrollo de una partida.
- Atributos:
  - points: Valor que representa el número de puntos obtenidos por un jugador en una partida.

Clase: Ranking

- Descripción: Clase que modela los datos de una clasificación entre los jugadores.
- Responsabilidades: Modelar una clasificación entre los jugadores en función de su nivel de habilidad.
- Atributos:
  - position: Número que representa la posición del jugador dentro de la clasificación.
  - mode: Identificador del modo de juego al que pertenecen los datos del jugador.

Clase: Interval

- Descripción: Clase que modela el periodo de tiempo en el que se almacenan las partidas.
- Responsabilidades: Modelar un intervalo de tiempo en el que se almacenan los datos de las partidas para el posterior cálculo de puntuaciones.
- Atributos:
  - duration: Número que representa la duración de un periodo.
  - initialDate: Día y hora en la que da comienzo el periodo.

Un **Player** participa en diferentes **Matches** mediante un **Matchmaking** que empareja a un **Player** con un máximo de otros 4 **Player** y que utiliza los **Results** para su algoritmo y calcular los niveles de habilidad de cada **Player**. Cada **Match** genera un **Result** al acabar que sirve para actualizar un **Ranking** compuesto por todos los **Players**. Estos **Matches** pertenecen a un **Interval** en el que los **Player** disputan sus **Matches**.

5.1.3. Diagramas de actividad

En el siguiente apartado se presentan los diagramas de actividad con el flujo que siguen los casos de uso especificados en la sección 5.1.1. Se detalla el flujo seguido por todas las partes que conformarían el sistema real, incluyendo cliente y Servidor App, las cuales se representan en color azul debido a que no son partes diseñadas en este TFG.

5.1.3.1. BuscarPartida

La figura 5.4 muestra el diagrama de actividad para el caso de uso ”Buscar partida”. Cuando el cliente inicia la llamada, se comunica con el Servidor App, el cual comprueba en la base de datos de la aplicación la identidad del usuario y en caso negativo se avisa del error. En caso afirmativo ya se procede al flujo que busca rivales para el jugador que solicita partida según sus datos almacenados en

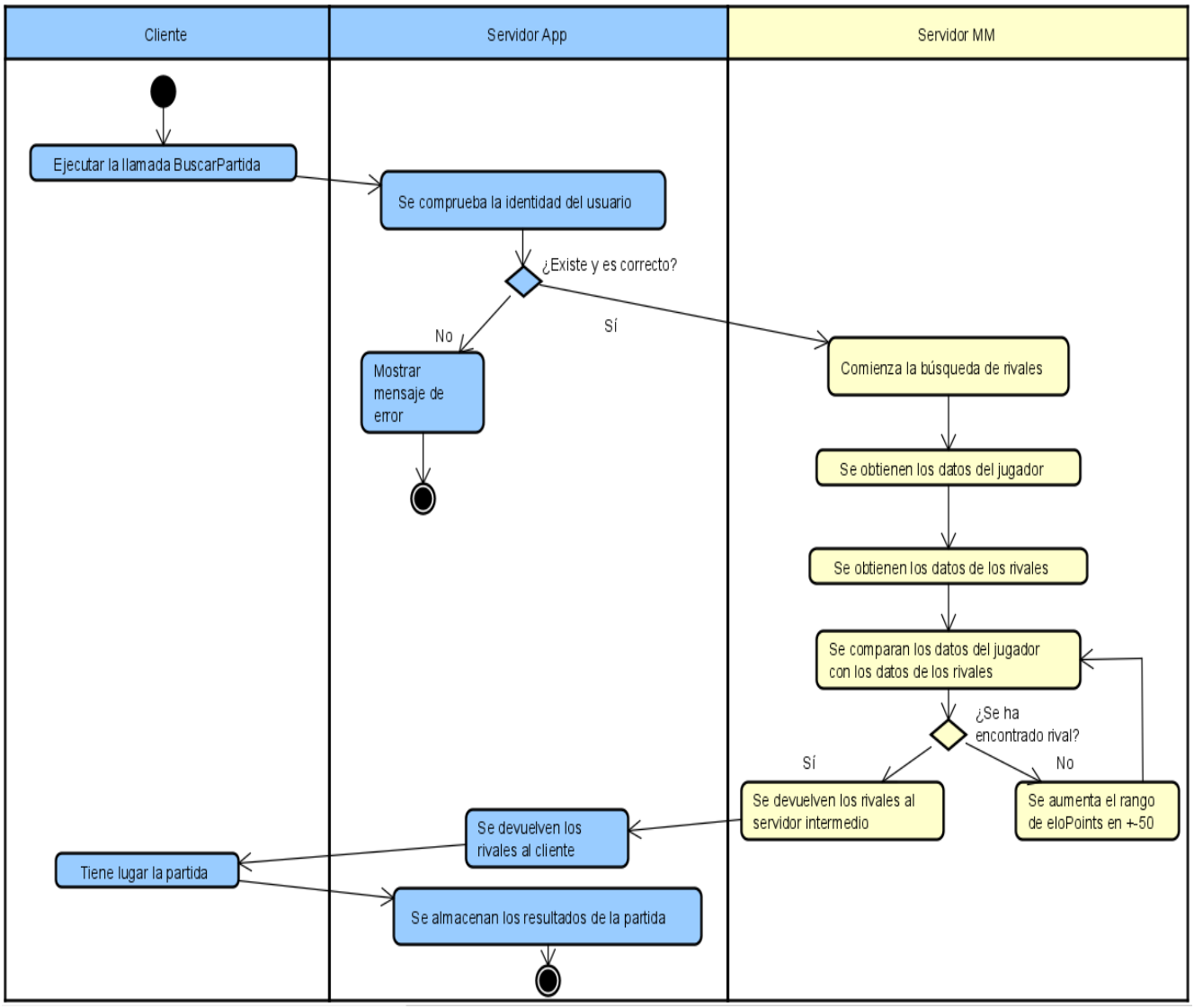

Figura 5.4: Diagrama de actividad de BuscarPartida

el sistema. Uno de los requisitos del sistema es devolver siempre un rival, por lo que en caso de hacer una barrida y no encontrar un jugador que cumpla las condiciones se aumenta el rango de eloPoints y se vuelve a proceder hasta que exista al menos un rival adecuado. Esa lista de rivales es devuelta al jugador para que este escoja y tenga lugar la partida, cuyos datos son almacenados cuando el jugador finaliza su acción.

5.1.3.2. RegistrarJugador

La figura 5.5 muestra el diagrama de actividad para el caso de uso ”Registrar Jugador”. En este caso se deben comprobar varios factores para insertar el jugador en la base de datos. En primer lugar que los datos tengan un formato correcto, después que este jugador no exista previamente en la lista de jugadores y, por último, que el modo de juego se corresponda con los existentes en la aplicación. Si se cumplen todas las condiciones se crea el usuario nuevo con los datos ELO por defecto, se almacena

en la base de datos y se confirma la acción con un mensaje de éxito.

Cabe destacar que el actor que desencadena este caso de uso puede ser el cliente o el Servidor App, lo cual dependerá de la implementación del juego que utilice el sistema.

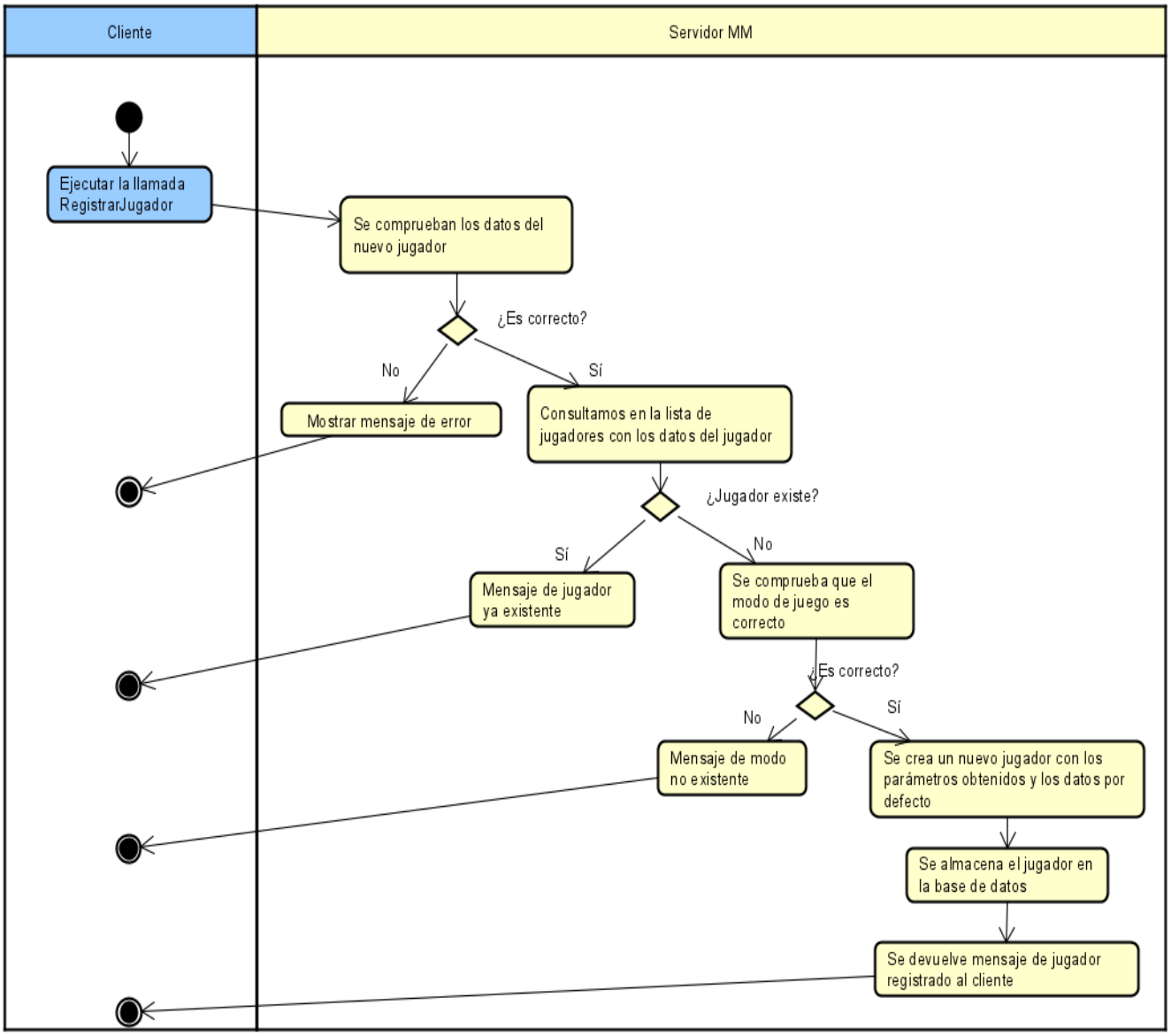

Figura 5.5: Diagrama de actividad de RegistrarJugador

5.1.3.3. PoblarJugadores

La figura 5.6 muestra el diagrama de actividad para el caso de uso ”Poblar Jugadores”. Para este caso de uso simplemente se comprueba que el fichero de entrada que contiene los jugadores a insertar existe, y en cuyo caso se vuelca el contenido en la base de datos y se devuelve un mensaje de éxito.

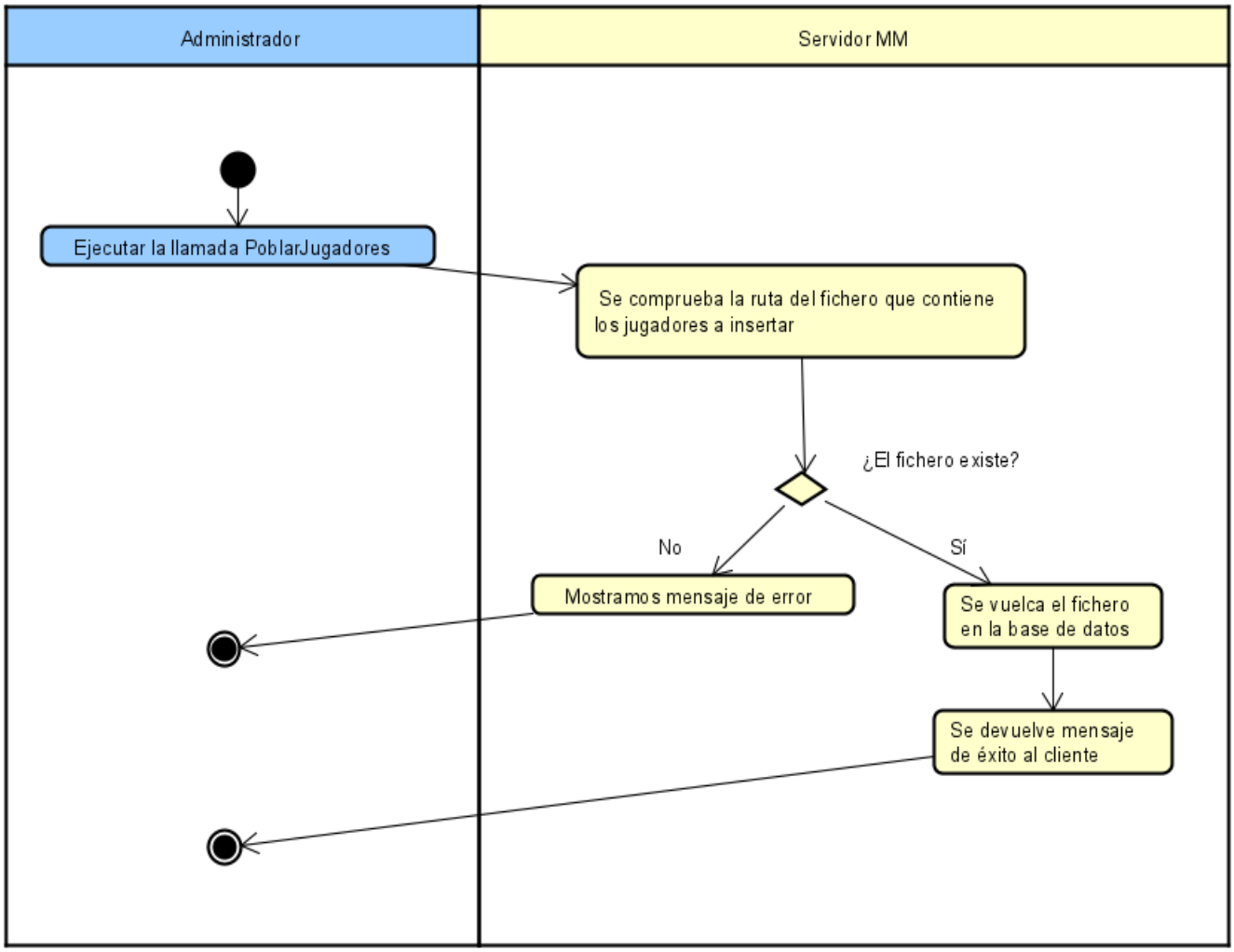

Figura 5.6: Diagrama de actividad de PoblarJugadores

5.1.3.4. CálculoPuntuaciones

Las figuras 5.7 y 5.8 muestran el diagrama de actividad para el caso de uso ”Cálculo Puntuaciones”. Este flujo se activa periódicamente cada 30 minutos. En este caso el Servidor MM solicita las partidas correspondientes al periodo al servidor de la app. Una vez obtenidas junto con los datos de los jugadores almacenados se obtienen quienes han participado y deben ser actualizados y para cada uno de ellos, mediante el algoritmo Glicko, se obtienen las nuevas puntuaciones de cara a los nuevos periodos, que como último paso se almacenan en la base de datos. Este proceso se realiza una vez por cada modo de juego existente en la aplicación.

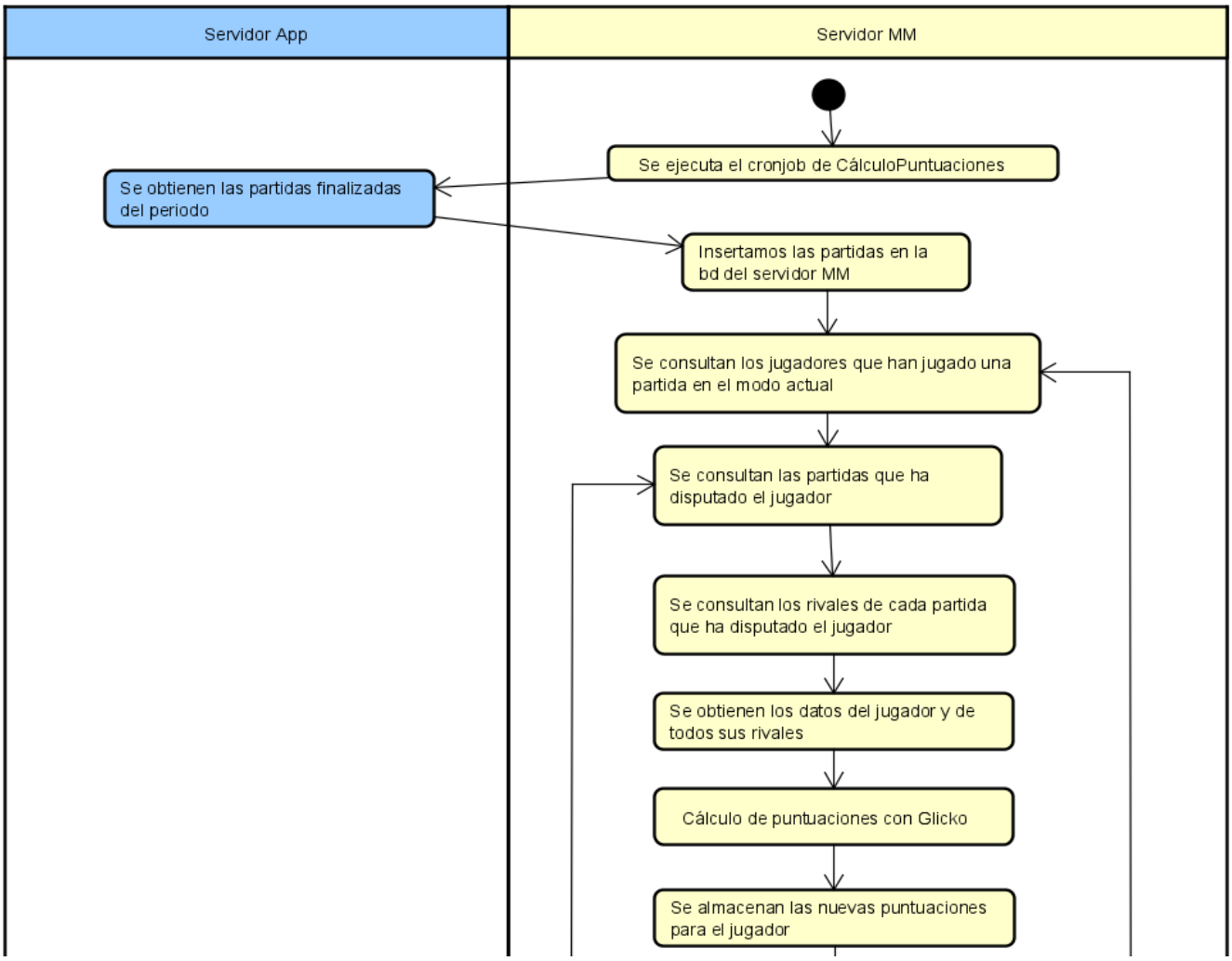

Figura 5.7: Diagrama de actividad de CálculoPuntuaciones (parte 1)

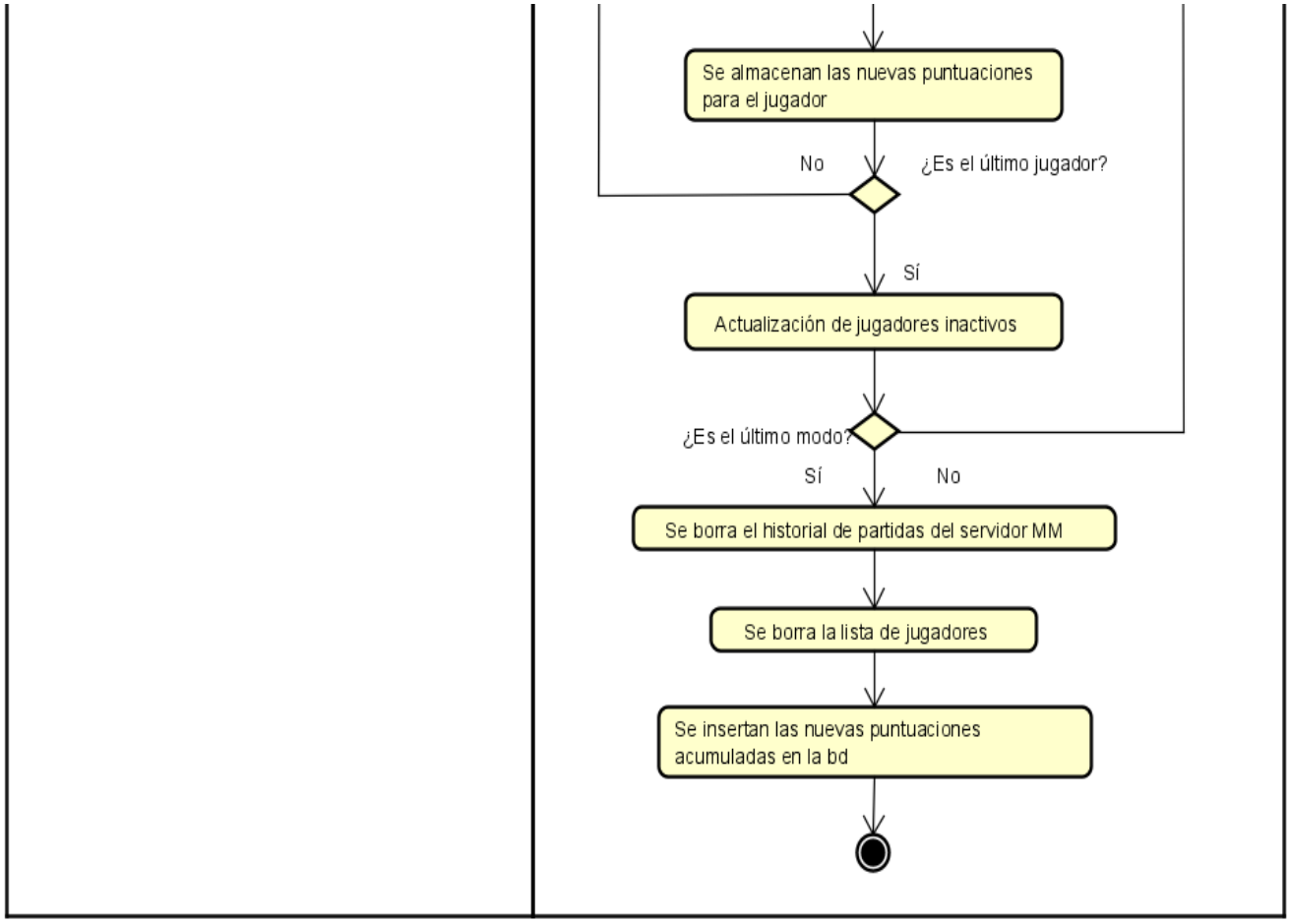

Figura 5.8: Diagrama de actividad de CálculoPuntuaciones (parte 2)

5.2. Estructura del proyecto

5.2.1. Características del Servidor MM

En esta sección se enumeran las funcionalidades que se han implementado en el Servidor MM y los servicios que ofrece, basándonos en los requisitos del sistema.

- **Búsqueda de partida**: El Servidor MM contiene la funcionalidad de búsqueda de rivales para un jugador que solicita buscar partida.
- **Partidas equilibradas**: El Servidor MM devuelve un listado de jugadores candidatos que han cumplido una serie de filtros en busca de la igualdad entre los usuarios que participan.
- **Concurrencia**: El sistema está diseñado para que ninguna petición sea bloqueante y varios usuarios puedan solicitar partida al mismo tiempo.
- **Registrar jugadores**: El Servidor MM ofrece la posibilidad de agregar a la base de datos a un jugador que se registra en el sistema
- **Poblar jugadores**: El Servidor MM permite poblar la lista de jugadores mediante un fichero con extensión .csv.
- **Algoritmo Glicko**: El sistema calcula las nuevas puntuaciones para cada jugador mediante un algoritmo que garantiza precisión a la hora de medir el grado de habilidad de un usuario.
- **Generalización y escalabilidad**: El sistema está diseñado de una forma que permita la adaptación a numerosas aplicaciones de carácter educativo. Además, la utilización de Node.js facilita que el sistema se pueda usar con cargas de usuarios muy elevadas.

5.2.2. Estructura del Servidor MM

En la figura 5.9 está especificada la estructura que se ha utilizado para el Servidor MM:

- **models**: En esta carpeta están los modelos de Mongoose que representan cada colección de la base de datos. En este caso ”lista_jugadores” y ”historial_partidas”.
- **node_modules**: Carpeta en la que se encuentran todos los módulos de Node y npm y sus dependencias instalados para el funcionamiento de la aplicación.
- **routes**: En este directorio se encuentran las diferentes clases en las que están divididos los endpoints de la aplicación, cada uno con su lógica y su flujo.
- **tests**: Carpeta con las clases de test automatizados jest. Estos test son los utilizados para la mayoría de pruebas de funcionamiento llevados a cabo en la quinta iteración del proyecto.

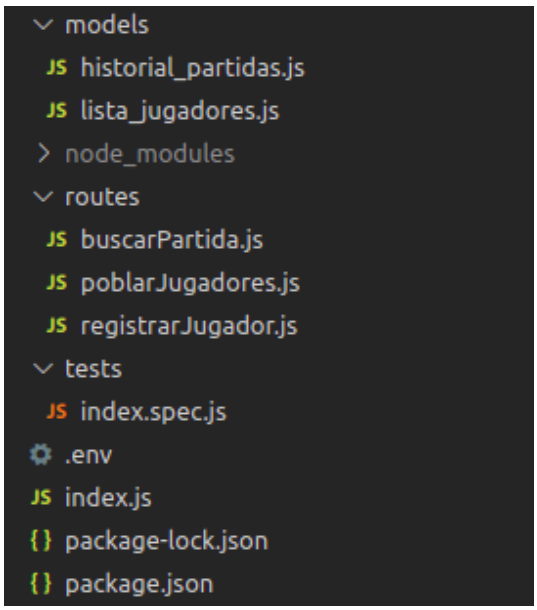

Figura 5.9: Estructura del Servidor MM del proyecto

- **.env**: Fichero que contiene todas las variables de entorno utilizadas en el proyecto. Este fichero sirve para centralizar constantes y que su modificación afecte al proyecto entero en ligar de modificar las diferentes clases una por una.

- **index.js**: Es el fichero que inicia la aplicación y dónde se encuentra el flujo principal así como el controlador de las rutas del proyecto. Es el fichero que se debe inicializar para arrancar el servidor.

- **package.json**: Estructura que contiene la lista de los paquetes npm utilizados en el proyecto, así como sus dependencias, versiones y posibles scripts para modificar la ejecución del código. Este listado sirve para decirle al proyecto los módulos que son necesarios en el momento de instalar el proyecto.

- **package-lock.json**: Estructura que se genera al instalar el proyecto y que contiene información sobre los módulos npm instalados pero de una forma más detallada.

5.2.3. Patrones de diseño

5.2.3.1. Arquitectura de capas

La arquitectura de capas se basa en agrupar sus componentes según su funcionalidad de forma que quede un sistema más simplificado y en el que estén bien definidos los propósitos de cada capa y las comunicaciones entre sí [59]. En nuestro caso, existe una capa de negocio que será el Servidor MM y una de persistencia que serán los modelos de MongoDB. La capa de presentación la conformará el frontend de la aplicación en la que se implemente este sistema de matchmaking.

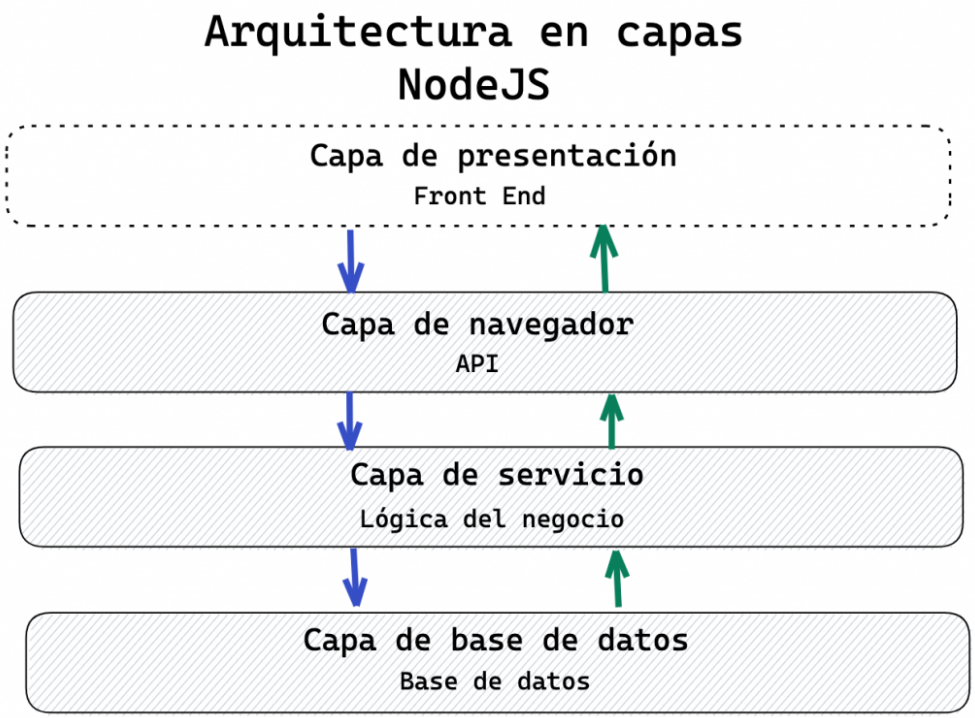

Figura 5.10: Imagen gráfica de la arquitectura de capas (extraída de [59])

5.2.3.2. Módulo

Este patrón de diseño creacional aparece cada vez que se crea un objeto con sus atributos y/o propiedades mediante la sintaxis '{ }' [60]. Este patrón es el más utilizado ya que el uso de objetos es una de las formas más sencillas de manejo de datos en lenguaje Javascript. En nuestro caso es utilizado en los envíos de las llamadas API encapsulando los parámetros en objetos.

5.2.3.3. Singleton

Este patrón de diseño se utiliza para asegurar que se crea una única instancia de una clase de forma que esta sea el único punto de acceso a las operaciones [60]. En nuestro proyecto es utilizado cuando aplicamos la librería Axios [49], ya que su funcionamiento está basado en este patrón al crear una única instancia para realizar las peticiones HTTP.

5.2.3.4. Observador

Patrón de diseño muy común en el lenguaje Javascript que se basa en el control y gestión de eventos por medio de dos objetos, dónde uno realiza la acción y el otro se queda en estado de escucha esperando algún cambio. En nuestro proyecto se ha incorporado principalmente en el uso de promesas [60].

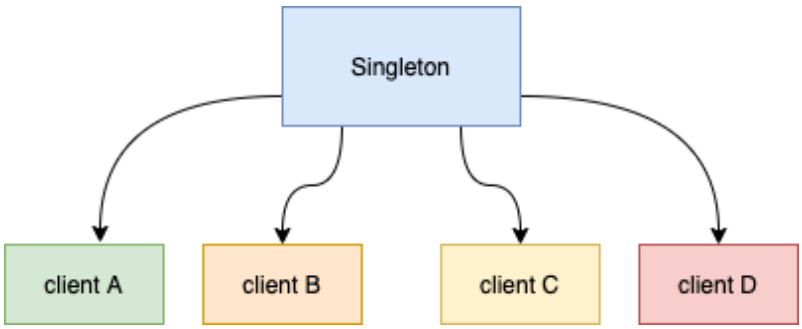

Figura 5.11: Imagen gráfica del patrón Singleton (extraída de [61])

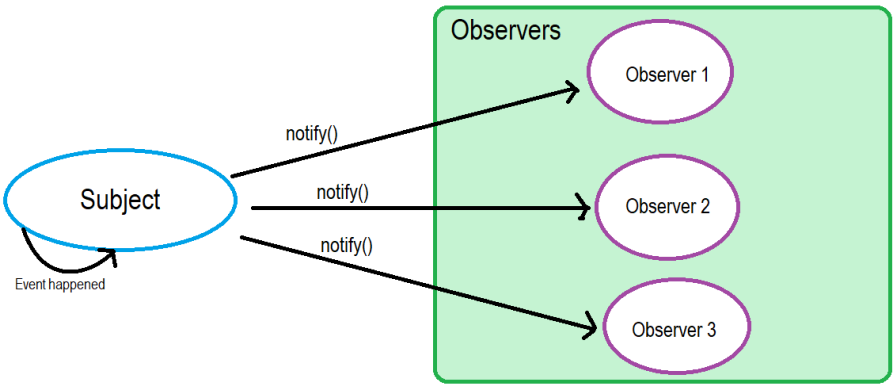

Figura 5.12: Imagen gráfica del patrón Observador (extraída de [62])

5.2.3.5. Middleware

Este patrón consiste en utilizar la salida de una función como la entrada de otra, de manera que queden encadenadas [63]. Es un patrón de diseño que permite reutilizar código y hacerlo más legible. Este patrón se ha utilizado al obtener el historial de partidas y usarlo como entrada para el flujo del cálculo de puntuaciones.

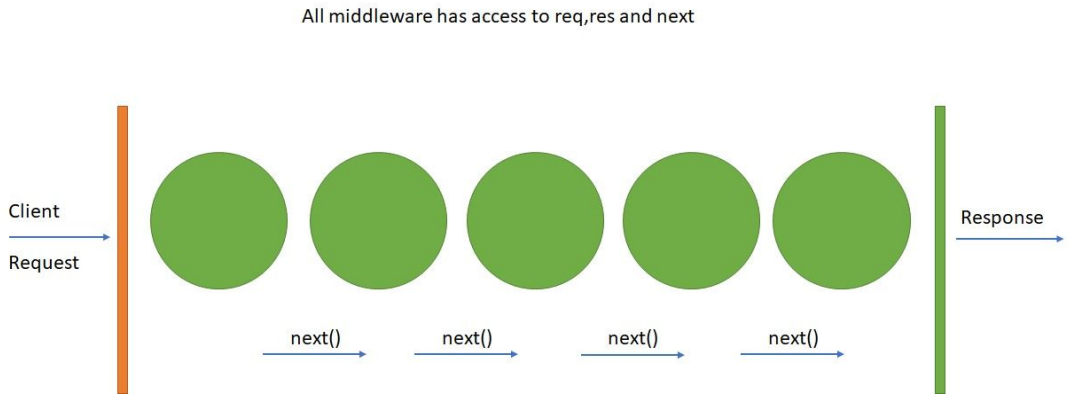

Figura 5.13: Imagen gráfica del patrón Middleware (extraída de [63])

5.3. Diagramas de diseño

5.3.1. Modelo de la base de datos

La base de datos utilizada para el proyecto en MongoDB. Esta base de datos es no relacional por lo que no presenta una forma única de modelar los datos que almacena [64]. En este caso, se ha optado por utilizar un modelo de relaciones entre documentos, esquema que se asemeja al utilizado en bases de datos relacionales y que es más conocido por el alumno.

Cabe destacar que se han utilizado dos bases de datos diferentes y separadas entre ellas, una para cada servidor. Esto se ha hecho así ya que se ha preservado la base de datos propia de la aplicación que almacena todos sus datos que no son importantes para nuestro proyecto excepto el historial con todas las partidas que de juegan y de las que se guardan registros, La otra base de datos es la propia del Servidor MM y que almacena los datos necesarios para realizar los emparejamientos y el cálculo de puntuaciones.

En la figura 5.14 se puede observar el modelo que siguen las entidades que conforman la base de datos del Servidor MM, así como sus atributos y relaciones.

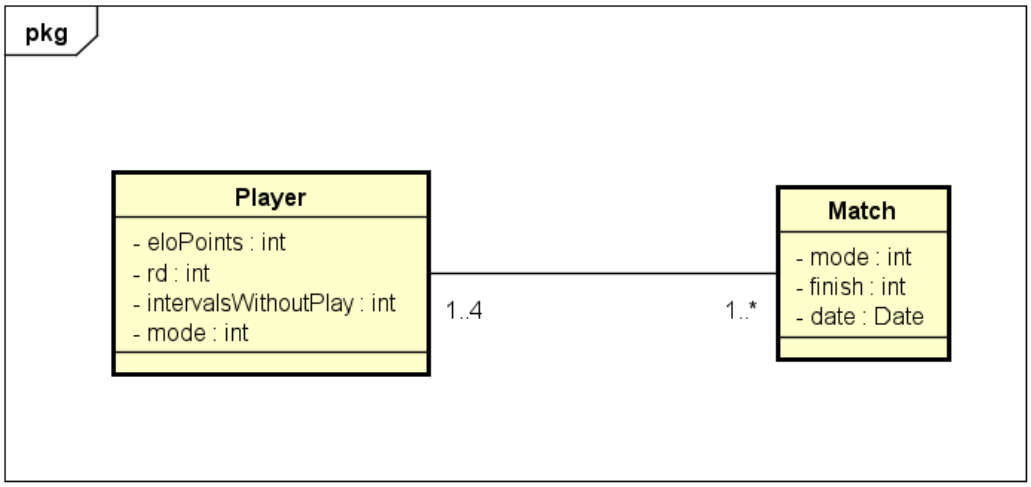

Figura 5.14: Diagrama de modelo de datos del Servidor MM

En el diagrama se ven dos entidades principales relacionadas entre sí. La entidad **Player** es la encargada de modelar los datos almacenados de un jugador. Esta entidad tiene una relación 1...4 a varios con la entidad **Match**, que es la destinada a modelar los datos de las partidas. Esta relación representa que un jugador puede disputar tantas partidas como desee y a la vez una partida está conformada por entre 1 y 4 jugadores.

La base de datos para el Servidor App implementada para nuestro proyecto solo recoge las partidas totales guardadas por la aplicación. En la figura 5.15 se puede ver el diagrama de datos para este caso, el cual sigue el mismo esquema para la entidad **Match** del diagrama anterior y las relaciones que pueda tener con los demás elementos de la aplicación no están reflejadas ya que no eran parte del desarrollo de este proyecto.

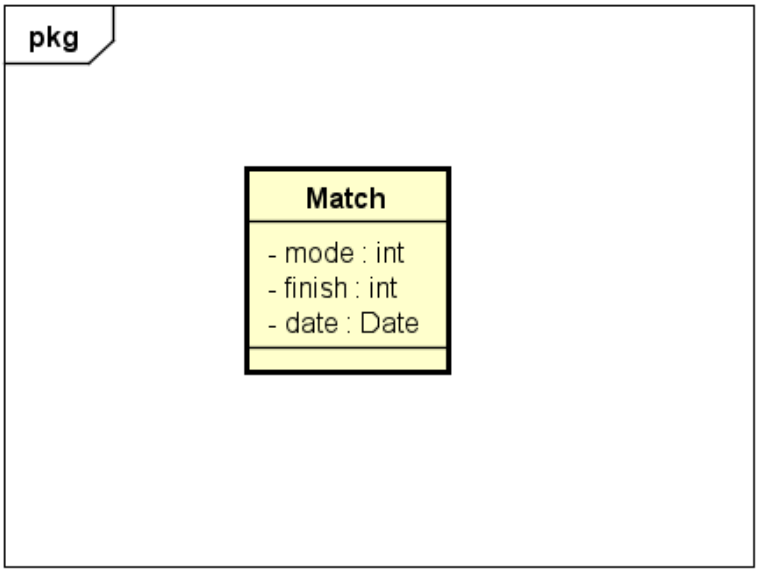

Figura 5.15: Diagrama de modelo de datos del Servidor App

5.3.2. Diagrama de paquetes

A continuación se muestra una representación en forma de diagrama de los paquetes que componen la aplicación:

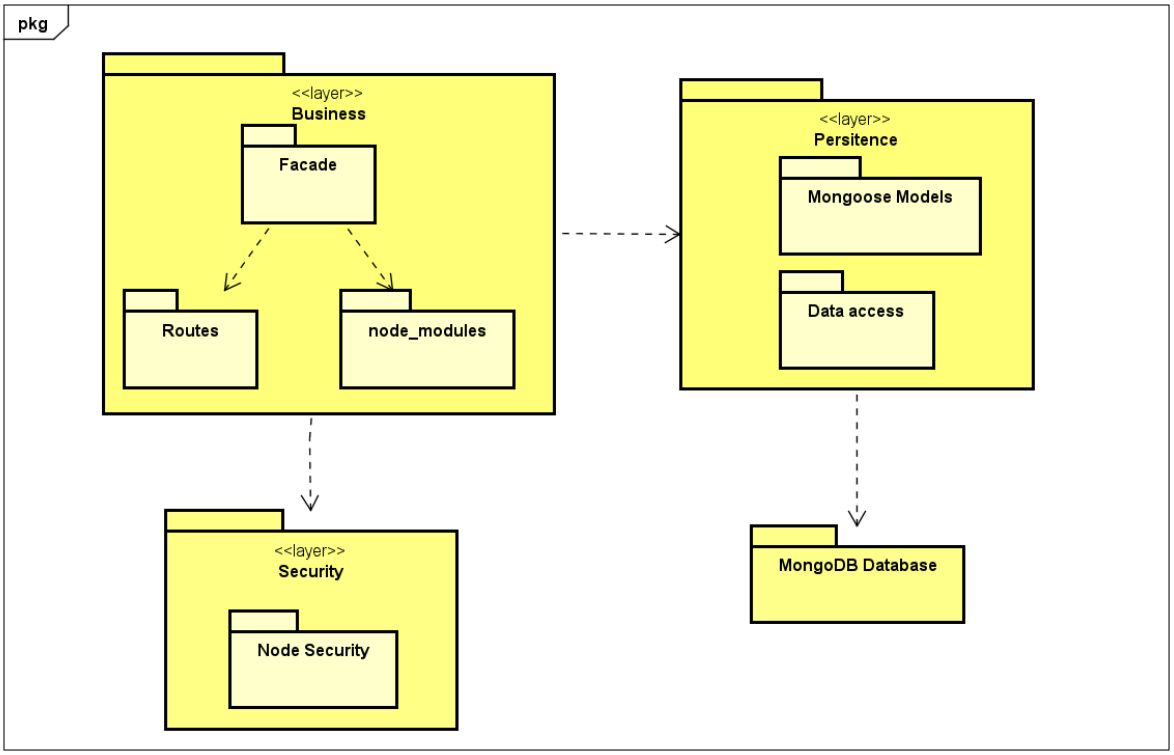

Figura 5.16: Diagrama de paquetes del sistema

El proyecto está dividido en varias capas:

- **Business layer:** Es la capa que tiene todas las operaciones para el correcto funcionamiento de la aplicación. En esta capa se reciben las peticiones API y se procesan los envíos para las comunicaciones entre las demás capas
  - Application facade: Lugar donde se reciben las peticiones y se comunica con las clases que poseen la funcionalidad específica.
  - Routes: Entidades que reciben los datos del facade, los procesan, operan y devuelven una respuesta.
  - Node_modules: Conjunto de paquetes de Node.js que dota de funcionalidad a ciertas tareas de la aplicación.
- **Security layer:** Es la llamada capa de seguridad. Dentro se establecen los paquetes que ayudan a mantener la seguridad de la aplicación. Solamente se posee un paquete (Node security) que es el que contiene los propios mecanismos de seguridad de Node.js.
- **Persitence layer:** Esta capa es el enlace entre la aplicación y la base de datos, ya sea para conexiones, consultas o comunicaciones.
  - Data access: Este paquete contiene los parámetros y credenciales para realizar la conexión con la base de datos (usuario, contraseña, enlace a la bd...)
  - Mongoose Models: Son los modelos que se establecen por medio de la librería Mongoose que representan cada colección de la base de datos.
- **Mongoose database:** Es la base de datos donde se almacenan y se obtienen los datos requeridos para el funcionamiento de la aplicación.

5.3.3. Diagrama de despliegue

En el diagrama de la Figura 5.17 se ve el esquema real de lo que sería el sistema de matchmaking integrado con una aplicación y las comunicaciones entre todas las partes. Los elementos en azul son los que no forman parte real del sistema diseñado en este TFG sino que son simulados. Además, en el diagrama se encuentran representados con el esquema con el que han sido desarrollados para el proyecto pero pueden tener una estructura diferente a decisión de los desarrolladores de la aplicación.

El sistema de matchmaking está soportado por un servidor Ubuntu. Este servidor a su vez contiene un servidor Nginx [44] que actúa como proxy que procesa y enruta las peticiones HTTP que le llegan hacia el servidor Node.js. En este servidor Node.js se encuentra la aplicación Express con el matchmaking desarrollado, la cual se comunica con la base de datos MongoDB mediante Mongoose. Esta base de datos se encuentra en la nube [65] pero es compatible que se encuentre físicamente en otro sistema o en la misma máquina que el servidor.

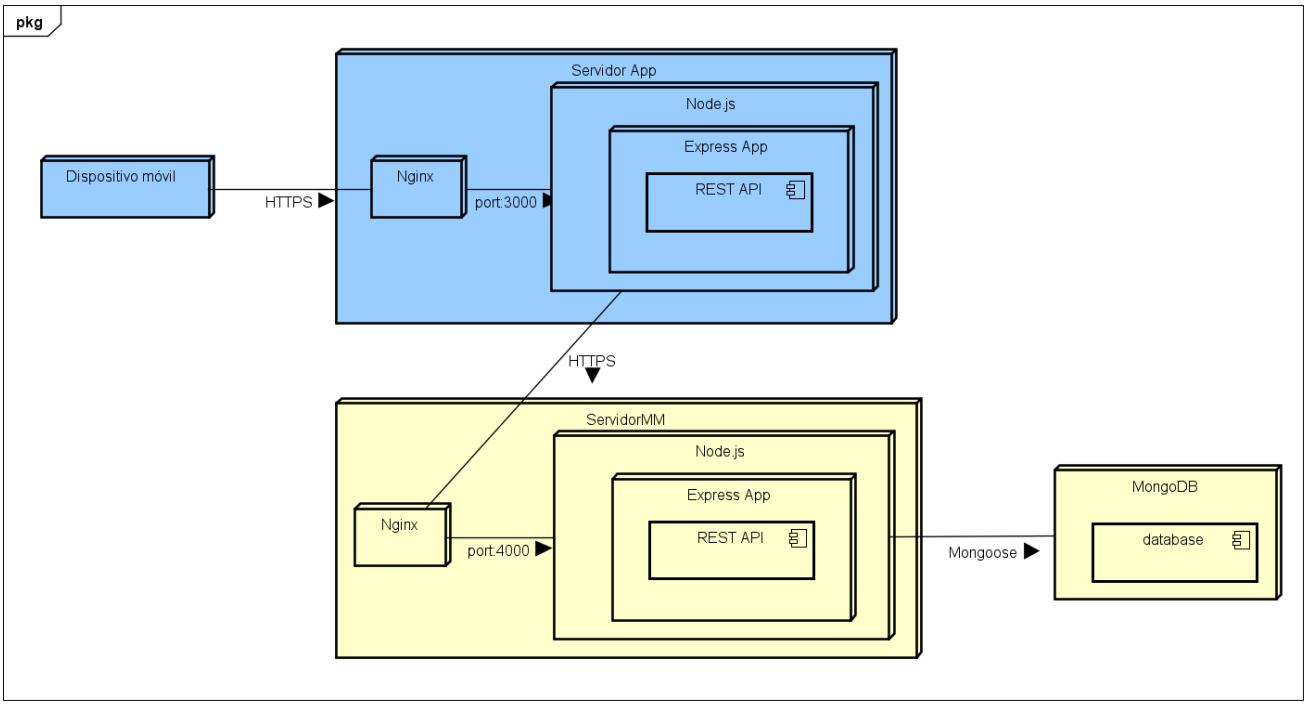

Figura 5.17: Diagrama de despliegue del sistema

Capítulo 6

Pruebas

En esta sección se detallan las pruebas de funcionamiento realizadas sobre la aplicación. Estas pruebas se han ido realizando tanto a lo largo del proyecto como en la penúltima iteración la cual ha estado dedicada explícitamente a ellas.

Estas pruebas se pueden dividir en dos tipos, pruebas de caja negra [66] y pruebas de carga [67]. Las pruebas de caja negra son las cuales se realizan para comprobar si el resultado final es correcto sin importar cómo se ha llegado a él. Las pruebas de carga son aquellas que muestran el comportamiento del sistema en función del tamaño de las entradas en términos de funcionamiento y tiempo.

El formato que seguirán las tablas recogidas en el apéndice B que muestran las pruebas realizadas será una letra P con el número de la prueba en cuestión, el nombre, una breve descripción general, el resultado esperado y el resultado final obtenido.

6.1. Pruebas de caja negra

Para la realización de estas pruebas se ha utilizado el programa Postman [34] en una primera instancia para comprobar que las llamadas eran correctas (el fichero con las pruebas realizadas en Postman se encuentra en el repositorio git que contiene el proyecto) y, finalmente, los módulos jest y supertest de Node.js, cuya función es crear y automatizar los test mediante código.

Jest [41] es un framework de Javascript que permite la creación de test automatizados de forma que se pueden realizar pruebas de secciones de código a fin de comprobar su funcionamiento. Estos test basan su forma de trabajo en activar el flujo de código que se quiere probar y comprobar las respuestas esperadas, para después mostrar por pantalla el mensaje de éxito o de error junto con el por qué y el lugar donde se ha producido. Para las pruebas de este proyecto además de haber utilizado Jest, se ha acompañado de supertest [55], que es un módulo de npm que está especializado en realizar test con llamadas API. Con este módulo se pueden comprobar parámetros de las llamadas como el código de estado, las cabeceras o el mensaje de respuesta y así poder ver si estos elementos también son los esperados para las peticiones realizadas [68].

Cabe destacar que algunas de estas pruebas automatizadas han dado problemas por temas de código asíncrono y de correlación con la base de datos en aquellas que involucraban una inserción y una posterior consulta de búsqueda para obtener los elementos introducidos y comprobar que esa inserción se realizaba de manera correcta. Estos test no devolvían la respuesta esperada mediante jest (devolvían un array vacío) pero la base de datos si contenía los elementos introducidos. También ocurrieron problemas con la parte de cálculo de puntuaciones y problemas de sincronización de jest con el *cronjob* que activa ese flujo.

Debido a estos inconvenientes y a la falta de tiempo en esta parte del proyecto, en su lugar se han realizado las pruebas de forma manual comprobando los resultados por salida de pantalla o retroalimentación de la base de datos, siendo un proceso mucho más exhaustivo y comprobando cada salida al detalle.

A continuación, se hará una breve explicación de cada una de las pruebas realizadas divididas por los distintos casos de uso del sistema. Todas las tablas con las pruebas y los resultados se encuentran en el Apéndice B.

6.1.1. Pruebas Poblar Jugadores

En las tablas B.1 y B.2 se muestra el resultado de las pruebas de funcionamiento del *endpoint* cuando se realiza una llamada sobre él con el nombre adecuado (código HTTP 200) y uno erróneo (código HTTP 404).

En la tabla B.3 se prueba que el resultado obtenido tras la llamada al *endpoint* es correcto y cumple con su función. En este caso la prueba consistió en pasar como entrada un fichero con 1000 jugadores por 3 modos de juego y comprobar tras una consulta a la lista de jugadores que la longitud del array obtenido es de 3000 jugadores.

6.1.2. Pruebas Buscar Partida

En las tablas B.4 y B.5 se vuelve a comprobar que las llamadas al *endpoint* se realizan correctamente si el enlace está escrito adecuadamente (código 200) o lanza un mensaje de error si no encuentra la ruta especificada (código 404).

En las tablas B.6 y B.7 se han comprobado los distintos casos existentes según el usuario que solicita buscar partida. En el primer caso se comprueba si se envía el mensaje adecuado si el usuario no tiene registro en la base de datos. En el segundo se comprueba que el resultado obtenido es un array con los rivales encontrados y que son los que cumplen las condiciones del matchmaking establecido en el sistema.

6.1.3. Pruebas Registrar Jugador

En las tablas B.8, B.9 y B.10 se realizan las pruebas pertinentes con las llamadas al *endpoint* y ver su funcionamiento con los códigos HTTP devueltos (correcto: código 200, no encontrado: código 404). En este caso se realiza una prueba a mayores que consistía en lanzar la petición con una ruta vacía y comprobar que también se capturaba este caso con un error 400.

En las tablas B.11, B.12 y B.13 se comprueban las diferentes casuísticas existentes para este caso de uso en función de la entrada proporcionada por el usuario. En este caso, se comprueba que un usuario a registrar ya estaba insertado en la base de datos, si el modo de juego proporcionado no existe en la aplicación o, si cumple ambos requisitos, el jugador se registra correctamente. En cada uno de ellos se lanza un mensaje de información sobre lo ocurrido y se comprueba que es el pertinente.

En las tablas B.14, B.15 y B.16 se comprueba mediante consultas a la base de datos, para los mismos casos del párrafo anterior, que el jugador aparece en la lista de jugadores si se ha registrado de manera correcta, o si por el contrario no se ha insertado si se ha producido alguno de los dos casos erróneos anteriores (repetición de jugador o modo no existente).

6.1.4. Pruebas Cálculo Puntuaciones

Este caso de uso se activa por medio de un *cronjob* y no de una llamada API a un *endpoint* determinado por lo que en este caso no existen pruebas de funcionamiento esperando códigos de respuesta.

En las tablas B.17 y B.18 se muestran las pruebas realizadas para comprobar que el resultado del algoritmo Glicko que obtiene las nuevas puntuaciones para los jugadores es el esperado. Se realiza una primera prueba con el flujo de un periodo único y otra con el flujo para varios periodos consecutivos a fin de ver que se utilizan como entrada del cálculo los resultados acumulados de los periodos anteriores.

A mayores, en la tabla B.19 se realiza una prueba para ver si la modificación del código para adaptar el algoritmo Glicko a partidas de 1 vs varios no afecta a los resultados finales y son los esperados.

6.2. Pruebas de carga

En esta sección se van a detallar las pruebas de carga realizadas y la evolución que estas han seguido en términos de tiempo de ejecución en función del tamaño de las entradas utilizadas. Los tiempos obtenidos son los correspondientes a una carga de 1000 jugadores en la lista de jugadores y 1000 partidas en el historial de partidas, tanto para el cálculo de nuevas puntuaciones como para la búsqueda de partida, ya que son las dos partes de código troncales y que más recursos consumen. El tiempo del resto de llamadas existentes es despreciable.

6.2.1. Mejoras implementadas

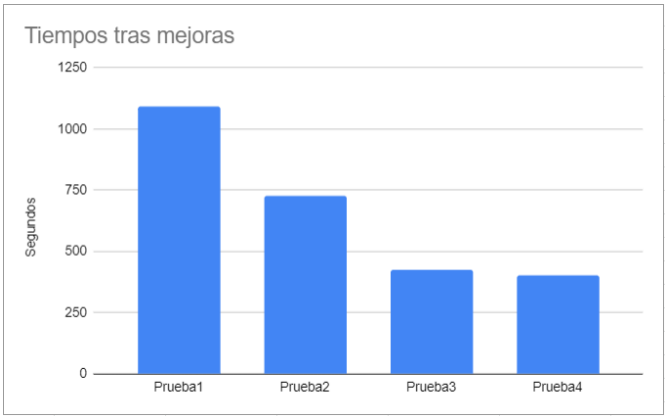

Figura 6.1: Segundos del tiempo de ejecución por cada mejora implementada

- Prueba 1: El código establecido al realizar esta prueba era el primitivo. El funcionamiento consistía en la utilización exclusivamente de ficheros para realizar el cálculo de puntuaciones. Para ello se utilizaban bastantes lecturas y escrituras a disco así como parseos para dividir los ficheros en líneas y campos para poder obtener los atributos necesarios y su posterior escritura. El tiempo total fue de 1093 segundos.

- Prueba 2: La primera mejora implementada fue la integración de la base de datos y el uso de consultas que sustituyeran todas las acciones con ficheros. De esta forma y con un trabajo de optimización de las propias consultas el tiempo se redujo considerablemente hasta los 728 segundos.

- Prueba 3: El tiempo de la anterior prueba se dividía en dos grandes sectores: Tiempo de cálculo y tiempo de actualización de las puntuaciones, los cuales estaban separados en dos bucles independientes que recorrían la lista de jugadores para su cometido. En esta mejora se reutilizaron ambos bucles de tal manera que se utilice el mismo índice para calcular las puntuaciones y a su vez actualizarlas. Para ello se utilizó un fichero auxiliar que iba guardando las puntuaciones nuevas de cada jugador para, una vez terminado el proceso con todos los jugadores, volcarlo a la base de datos en una única operación que consumía menos tiempo.

  Con esta mejora, el tiempo de actualización de todas las puntuaciones pasó a ser de menos de un segundo, por lo que el tiempo de ejecución total es el resultante del cálculo (424 segundos).

- Prueba 4: En esta última prueba se optimizó la declaración de variables y se sustituyó el fichero auxiliar por un array de forma que ahorráramos operaciones de máquina y lecturas a disco. Con esto se redujo algo más el tiempo total que pasó a ser de 402 segundos.

Cabe destacar que se pensó y se intentó implementar una quinta mejora que redujera aún más el tiempo de ejecución pero no se pudo llevar a cabo por complejidad y falta de tiempo. Esta mejora consistía en utilizar paralelismo con el bucle que recorre la lista de jugadores para obtener la puntuación nueva de cada uno. Al ser un bucle secuencial, el tiempo de este será el tiempo de cada iteración

multiplicado por el número de iteraciones. Utilizando el paralelismo, se podría dividir el número de iteraciones entre el número de procesadores del sistema, de forma que el tiempo de ejecución fuese ”tiempo de cada iteración x (número de iteraciones/número de procesadores)” [69].

6.2.2. Tiempos por tamaño de carga

En estas pruebas el objetivo es medir los segundos que tarda una carga determinada de usuarios concurrentes en buscar partida y cómo el sistema se comporta a estos niveles de exigencia.

Para la realización de estas pruebas se han utilizado hilos concurrentes que simulan la acción de un jugador que busca partida, almacenando el tiempo de cada uno de ellos y realizando la media.

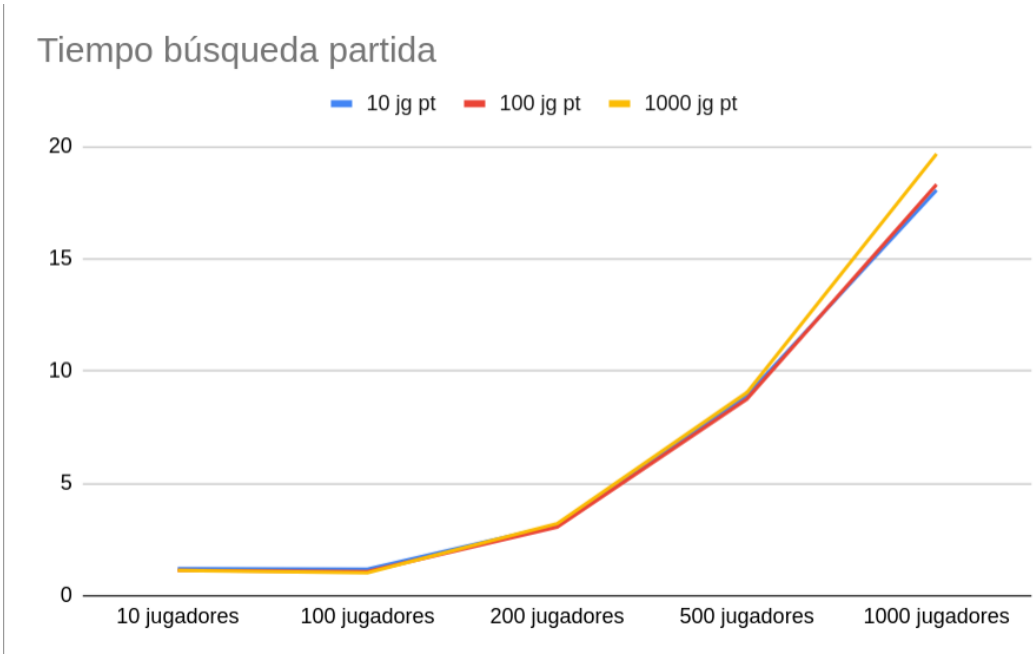

Figura 6.2: Tiempo de ejecución (segundos) de la búsqueda de partida en función del número de jugadores concurrentes

En la figura 6.2 se puede apreciar como la carga del fichero no afecta de manera ostensible al rendimiento del sistema que sigue una evolución exponencial en función del número de jugadores simultáneos que buscan partida al mismo tiempo. Los tiempos medios vienen detallados en la tabla 6.1.

Con 10 y 100 jugadores el sistema se comporta de la misma forma, es decir, puede soportar 100 usuarios al mismo tiempo sin que existan retrasos por exceso de carga. Comienza a empeorar ligeramente a partir de los 200, aumentando el tiempo de manera creciente a medida que aumenta el número de jugadores. Esto es lógico ya que se producen cuellos de botella en cuanto tiene que procesar un número de llamadas elevado.

La complejidad de tiempo del algoritmo de búsqueda de partida sería de $O(1)$. Esto es así ya que no existe relación entre el tiempo que tarda en ejecutarse y el número de jugadores que hay en la lista de jugadores y permanece lineal aunque la carga aumente.

Nº de jugadores concurrentes	10 jugadores en la bd	100 jugadores en la bd	1000 jugadores en la bd
10 jugadores	1.202 s	1.134 s	1.127 s
100 jugadores	1.18 s	1.081 s	1.028 s
200 jugadores	3.15 s	3.06 s	3.21 s
500 jugadores	8.92 s	8.76 s	9.05 s
1000 jugadores	18.077 s	18.33 s	19.69 s

Tabla 6.1: Tiempos medios (segundos) de la búsqueda de partida según la carga de la base de datos y los jugadores concurrentes

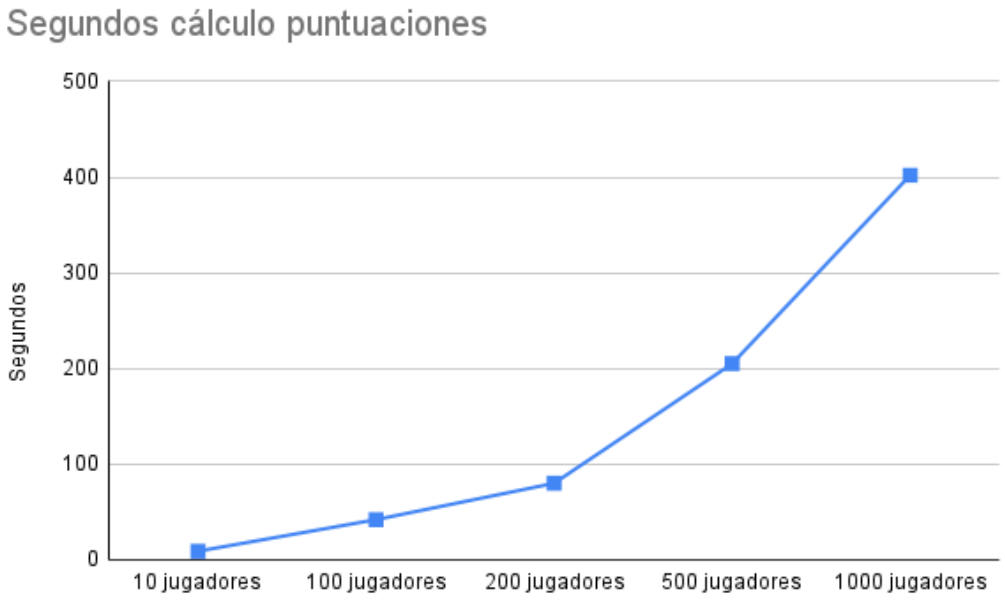

Figura 6.3: Tiempo de ejecución (segundos) del cálculo en función del número de jugadores y partidas durante un periodo

En la figura 6.3 se puede ver el tiempo de ejecución del flujo del cálculo de nuevas puntuaciones teniendo en cuenta el número de jugadores y partidas con las que opera. Se puede apreciar como a medida que en la base de datos se almacenan más jugadores y por consiguiente se registran más partidas disputadas el tiempo de ejecución es mayor. Este aumento es natural ya que el código tiene que procesar todos los jugadores participantes y a mayor número mayor es el tiempo de ejecución.

Viendo la gráfica y los tiempos de la tabla 6.2 podemos concluir que la complejidad para el algoritmo de cálculo de puntuaciones es de $O(N^2)$ ya que a medida que aumenta la carga en la lista de jugadores el tiempo que tarda en completarse todo el flujo es mayor y crece de una forma exponencial de una manera similar a la que muestra la gráfica 6.4.

Nº de jugadores	Segundos
10 jugadores	9.12 s
100 jugadores	42.50 s
200 jugadores	80.31 s
500 jugadores	205.09 s
1000 jugadores	402.83 s

Tabla 6.2: Tiempo medio (segundos) del cálculo de puntuaciones según el número de jugadores existentes en la base de datos

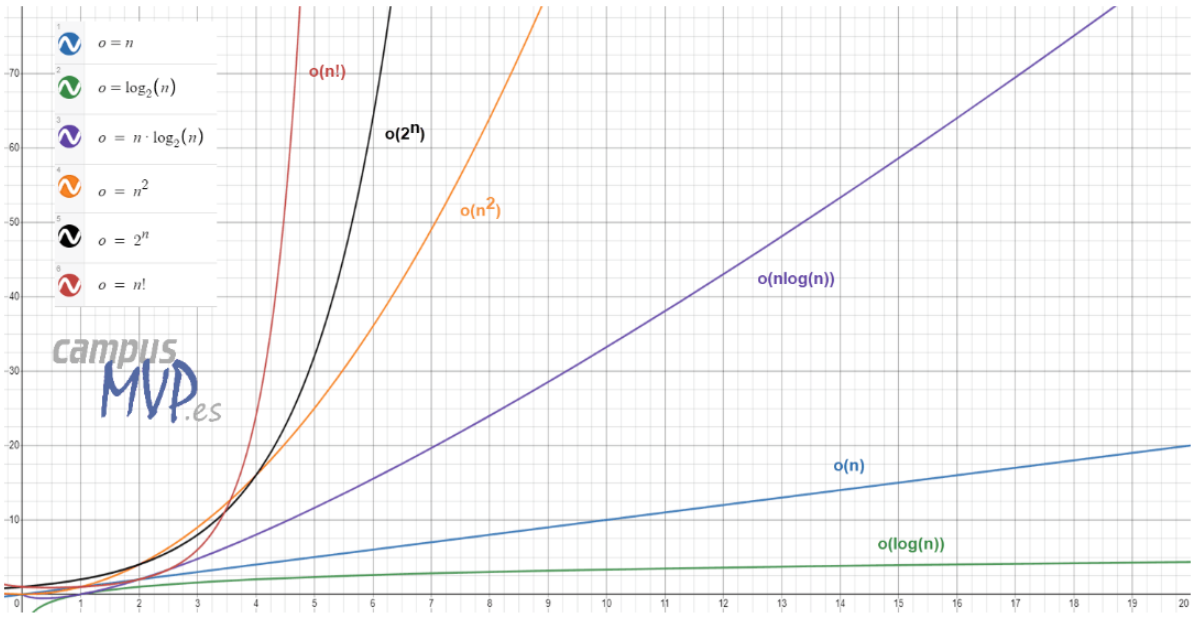

Figura 6.4: Gráfica comparativa de los distintos tipos de complejidad [70]

Capítulo 7

Conclusiones

En este capítulo se recogen las conclusiones alcanzadas al final de este proyecto, las sensaciones que el alumno ha ido teniendo según iba avanzando y las ideas de posibles implementaciones de cara al futuro que mejoren aún más el trabajo presentado.

En primer lugar, el trabajo realizado en este TFG ha sido la implementación de un sistema de emparejamiento (matchmaking) para aplicaciones educativas multijugador online que pudiera servir como un matchmaking genérico para otros juegos de forma escalable y de fácil adaptación. Como caso de prueba, se ha integrado en COP [2], una aplicación educativa para la mejora de la pronunciación extranjera, desarrollada en un TFG previo [13]. Este desarrollo del sistema de matchmaking ha seguido los requisitos previamente marcados y definidos en las primeras reuniones, además de alguna funcionalidad añadida posteriormente, buscando el objetivo de una mejora en la satisfacción y experiencia de uso de la aplicación por parte de los usuarios.

Para llevar a cabo todo el proceso, el alumno ha tenido que realizar un estudio previo de las diferentes posibilidades en el sistema de emparejamiento a implementar mediante la lectura de artículos de investigación sobre el tema, probando de primera mano aplicaciones con distintos matchmaking y comparando las ventajas e inconvenientes buscando la mejor opción en cuanto a adaptabilidad con el tipo de aplicaciones con las que se iba a trabajar. Además, el alumno también realizó un proceso de adaptación y aprendizaje a las tecnologías que se iban a aplicar con la realización de un sistema base que comunicara los elementos que iban a participar en el sistema final.

El sistema final desplegado ha constado de un servidor principal, Servidor MM, en el que se ha implementado el matchmaking desarrollado, y otro secundario, Servidor App, que simula el servidor de la aplicación que sustenta el matchmaking, en este caso, COP. Además, se han desarrollado una serie de clientes de prueba que han servido para simular y comprobar el funcionamiento del flujo de mensajes entre los servidores y ellos mismos. Esto ha de ser así ya que tanto los jugadores que participan como el propio servidor de la aplicación son independientes del emparejamiento y las características que estos tengan dependerán de los diseñadores de la aplicación en sí. En la parte del Servidor MM se pueden establecer restricciones, como por ejemplo, los formatos en los que el servidor de la app nos tiene que pasar la información de las llamadas o el número de posibles rivales que devuelve el matchmaking, pero

no cambiar su funcionamiento, sino adaptarlo. Para la implementación de los servidores se ha utilizado una tecnología Node.js con Express.js y MongoDB como base de datos y el módulo de Mongoose para las comunicaciones con ella. Para los clientes se ha utilizado lenguaje Python. Para todos ellos además se han utilizado librerías para la creación y envío de llamadas (axios y request).

Una de las principales dificultades comprobadas ha sido la de realizar una planificación óptima ya que son muchos los factores que pueden alterar las horas estimadas. En mi caso particular, la inexperiencia, la dificultad con el aprendizaje de las nuevas tecnologías o errores en el código que retrasaban el avance son algunos de ellos. Por eso es importante establecer en primera instancia una metodología de trabajo adecuada como, en el caso de este proyecto, un enfoque iterativo e incremental [9] que ha permitido a los tutores seguir el avance del alumno de manera más cercana y focalizada e impedir un retraso excesivo en caso de desarrollar algo que no cumplía los requisitos establecidos. También ha sido importante tener una estimación de posibles riesgos y un plan de mitigación para ellos. Algunos de estos riesgos que han llegado a materializarse han sido una mala planificación, enfermedad o o dificultad con las herramientas de trabajo, y en todos ellos se estableció el plan de acción correspondiente que era aplicar horas extra, sobre todo en los últimos dos meses del proyecto.

Otro aspecto importante en el proyecto han sido las pruebas, que ofrecen la retroalimentación necesaria que permite comprobar si el flujo de código del programa sigue el funcionamiento que nosotros hemos querido darle y el comportamiento del servidor no se ve afectado por la excesiva carga de trabajo. El enfoque de estas pruebas ha sido progresivo. Por cada pequeña funcionalidad implementada se ha comprobado que esta presentaba los resultados esperados, y de no ser así, no se avanzaba con lo siguiente hasta que el inconveniente estuviese resuelto. Agrupar mucha funcionalidad y probarla de golpe en una etapa más tardía no es conveniente ya que se pueden acumular los errores y se pueden relacionar entre ellos, siendo más complicado localizarlos y solventarlos. Para la realización de estas pruebas se ha usado *jest* acompañado del módulo *supertest*, los cuales han permitido automatizar los test y aumentar la retroalimentación.

Por último, en el aspecto personal ha sido un proyecto muy interesante y enriquecedor, tanto por la propuesta que ha sido de una temática muy afín a mis gustos, tanto por el aprendizaje que ha supuesto el estudiar cómo funciona un matchmaking por dentro, qué tipos existen y acabar desarrollándolo con tecnologías también nuevas y que son útiles de cara al futuro. También, este trabajo ha servido para ganar independencia en las destrezas tanto en programación como en planificación, aprender a resolver cuestiones por mi cuenta aún teniendo el apoyo de los tutores en todo momento y poder poner el práctica lo aprendido en todos los años de carrera y utilizarlo para realizar un proyecto largo y dedicado. En consecuencia, este TFG me ha permitido ganar en experiencia y permitirme evolucionar de cara a lo que está por venir después de la etapa universitaria.

7.1. Trabajo futuro

En esta sección se van a detallar los aspectos de mejora que se podrían implementar en un futuro al trabajo presentado:

- **Mejorar tiempo de ejecución**: Reducir el tiempo de ejecución del cálculo de puntuaciones de forma que se reduzca el uso de recursos y el tiempo en el que el servidor está inactivo en otras funciones. Esto se puede conseguir mediante paralelismo dividiendo el tiempo total de ejecución entre el número de procesadores del dispositivo, mediante una gestión avanzada de hilos y subprocesos.
- **Mayor seguridad**: Establecer mecanismos de defensa más allá de los que vienen por defecto, como por ejemplo, proteger contraseñas o tokens de usuarios o con listas de acceso.
- **Ampliación de funcionalidad**: Posibilidad de incluir submodos de juego, o nuevos requisitos específicos de jugadores.
- **Test con usuarios reales**: Probar la aplicación con usuarios reales para comprobar la efectividad del sistema de emparejamiento realizado.

Apéndices

Apéndice A

Acrónimos

- **API**: Application Programming Interface.
- **App**: Aplicación.
- **BD**: Base de Datos.
- **CoP**: Clash of Pronunciations.
- **CPU**: Central Processing Unit.
- **CSV**: Comma Separated Values.
- **CU**: Caso de Uso.
- **GB**: GigaByte.
- **GHz**: GigaHercio.
- **HDD**: Hard Disk Drive.
- **HTTP**: Hypertext Transfer Protocol.
- **IVA**: Impuesto sobre el Valor Añadido.
- **JSON**: JavaScript Object Notation.
- **LTS**: Long Term Support.
- **RAM**: Random Access Memory.
- **RD**: Rating Deviation.
- **REST**: REpresentational State Transfer.
- **RF**: Requisito Funcional.
- **RI**: Requisito de Información.

- **RNF**: Requisito No Funcional.
- **SSD**: Solid State Drive.
- **TFG**: Trabajo de Fin de Grado.
- **TFM**: Trabajo de Fin de Máster.
- **UML**: Unified Modeling Language.
- **UVa**: Universidad de Valladolid.
- **vs**: Versus.

Apéndice B

Pruebas realizadas en la aplicación de prueba de llamadas

La explicación de cada tabla está recogida en la sección 6.1.

ID	P01
Nombre	PoblarJugadores código 200
Descripción	Comprobación de llamada exitosa por medio del código HTTP de respuesta
Resultado esperado	200
Resultado obtenido	Correcto

Tabla B.1: Descripción de la prueba P01

ID	P02
Nombre	PoblarJugadores código 404
Descripción	Comprobación de llamada errónea por medio del código HTTP de respuesta
Resultado esperado	404
Resultado obtenido	Correcto

Tabla B.2: Descripción de la prueba P02

ID	P03
Nombre	PoblarJugadores insertar 1000
Descripción	Se utiliza la llamada para insertar un fichero con 1000 jugadores por modo de juego en la base de datos
Resultado esperado	Lista_jugadores.length = 3000
Resultado obtenido	Correcto

Tabla B.3: Descripción de la prueba P03

ID	P04
Nombre	BuscarPartida código 200
Descripción	Comprobación de llamada exitosa por medio del código HTTP de respuesta
Resultado esperado	200
Resultado obtenido	Correcto

Tabla B.4: Descripción de la prueba P04

ID	P05
Nombre	BuscarPartida código 404
Descripción	Comprobación de llamada errónea por medio del código HTTP de respuesta
Resultado esperado	404
Resultado obtenido	Correcto

Tabla B.5: Descripción de la prueba P05

ID	P06
Nombre	BuscarPartida Jugador Inexistente
Descripción	Un jugador sin referencia en la base de datos busca partida
Resultado esperado	”No se ha encontrado al jugador que solicita partida en la base de datos”
Resultado obtenido	Correcto

Tabla B.6: Descripción de la prueba P06

ID	P07
Nombre	BuscarPartida Correcto
Descripción	Un jugador con referencia en la base de datos busca partida
Resultado esperado	(Array de JSON con los rivales que cumplen las condiciones de búsqueda del jugador que busca partida)
Resultado obtenido	Correcto

Tabla B.7: Descripción de la prueba P07

ID	P08
Nombre	RegistrarJugador código 200
Descripción	Comprobación de llamada exitosa por medio del código HTTP de respuesta
Resultado esperado	200
Resultado obtenido	Correcto

Tabla B.8: Descripción de la prueba P08

ID	P09
Nombre	RegistrarJugador código 404
Descripción	Comprobación de llamada errónea por medio del código HTTP de respuesta
Resultado esperado	404
Resultado obtenido	Correcto

Tabla B.9: Descripción de la prueba P09

ID	P10
Nombre	RegistrarJugador código 400
Descripción	Comprobación de llamada errónea por datos vacíos por medio del código HTTP de respuesta
Resultado esperado	400
Resultado obtenido	Correcto

Tabla B.10: Descripción de la prueba P10

ID	P11
Nombre	RegistrarJugador correcto respuesta
Descripción	Respuesta del servidor al insertar correctamente a un jugador
Resultado esperado	”El jugador se ha registrado correctamente”
Resultado obtenido	Correcto

Tabla B.11: Descripción de la prueba P11

ID	P12
Nombre	RegistrarJugador incorrecto respuesta
Descripción	Respuesta del servidor al insertar a un jugador con datos incorrectos
Resultado esperado	”El modo no existe. Jugador no registrado”
Resultado obtenido	Correcto

Tabla B.12: Descripción de la prueba P12

ID	P13
Nombre	RegistrarJugador repetido respuesta
Descripción	Respuesta del servidor al insertar a un jugador ya existente
Resultado esperado	”El jugador a registrar ya existe”
Resultado obtenido	Correcto

Tabla B.13: Descripción de la prueba P13

ID	P14
Nombre	RegistrarJugador correcto BD
Descripción	Comprobación del estado de la base de datos al insertar a un jugador correctamente
Resultado esperado	(Array con los datos del jugador insertado después de realizar consulta a la base de datos)
Resultado obtenido	Correcto

Tabla B.14: Descripción de la prueba P14

ID	P15
Nombre	RegistrarJugador incorrecto BD
Descripción	Comprobación del estado de la base de datos al no insertar un jugador por tener datos erróneos
Resultado esperado	(Array vacío ya que la consulta a la base de datos no trae resultado)
Resultado obtenido	Correcto

Tabla B.15: Descripción de la prueba P15

ID	P16
Nombre	RegistrarJugador repetido BD
Descripción	Comprobación del estado de la base de datos al no insertar a un jugador por estar repetido
Resultado esperado	(Array con los datos del jugador insertado después de realizar consulta a la base de datos y solo una vez)
Resultado obtenido	Correcto

Tabla B.16: Descripción de la prueba P16

ID	P17
Nombre	Calculo puntuaciones 1 periodo
Descripción	Comprobación del resultado del cálculo de nuevas puntuaciones durante un único periodo
Resultado esperado	En la base de datos se encuentran sumados y restados los puntos ELO y RD para los jugadores ganadores y perdedores, así como sumado 1 a los periodos sin jugar de los jugadores que no han participado. Borrado del historial de partidas del Servidor MM
Resultado obtenido	Correcto

Tabla B.17: Descripción de la prueba P17

ID	P18
Nombre	Calculo puntuaciones periodos consecutivos
Descripción	Comprobación del resultado del cálculo de nuevas puntuaciones durante varios periodos consecutivos
Resultado esperado	En la base de datos se encuentran sumados y restados los puntos ELO y RD para los jugadores ganadores y perdedores, así como sumado 1 a los periodos sin jugar de los jugadores que no han participado, acumulados con los resultados de los periodos anteriores. Borrado del historial de partidas del Servidor MM
Resultado obtenido	Correcto

Tabla B.18: Descripción de la prueba P18

ID	P19
Nombre	Calculo puntuaciones partidas 1 vs varios
Descripción	Comprobación del resultado del cálculo de nuevas puntuaciones existiendo partidas que involucran más de dos jugadores
Resultado esperado	En la base de datos se encuentran sumados y restados los puntos ELO y RD para los jugadores ganadores y perdedores de las partidas 1 vs varios, siendo estos los correctos según el mecanismo establecido en el algoritmo para estos casos
Resultado obtenido	Correcto

Tabla B.19: Descripción de la prueba P19

Apéndice C

Comparativa de matchmaking de aplicaciones educativas

Nombre	Nº de descargas	Partidas 1 jugador
Atriviate [71]	(GPlay) 5.000.000+	Sin modos que no afecten a las estadísticas y posible matchmaking.
Apalabrados [72]	(GPlay) 10.000.000+	Modos en solitario para ganar monedas con las que comprar ventajas en las partidas 1vs1.
Preguntados [18]	(GPlay) 100.000.000+	Modos en solitario para ganar monedas con las que comprar ventajas en las partidas 1vs1.
Preguntados 2 [73]	(GPlay) 10.000.000+	Modos en solitario para ganar monedas con las que comprar ventajas en las partidas 1vs1.
Mezcladitos 2 [74]	(GPlay) 5.000.000+	Modos en solitario para ganar monedas con las que comprar ventajas en las partidas 1vs1.
Lingo! [75]	(GPlay) 100.000+	Sin modos que no afecten a las estadísticas y posible MM.
Kryss - Batalla de palabras [76]	(GPlay) 1.000.000+	1vs1 (Con la IA): No afecta a las estadísticas de matchmaking.
Scrabble Go [77]	(GPlay) 100.000.000+	▪ Modo en solitario (Word Drop): Consiste en buscar palabras estilo crucigrama. ▪ Modo en solitario (Rush): Consiste en conseguir la mayor puntuación posible en un tablero pequeño. ▪ Modo en solitario (Tumbler): Consiste en formar palabras usando anagramas dados.
Fight List [78]	(GPlay) 10.000.000+	Sin Modos que no afecten a las estadísticas y posible matchmaking.
Cuestionados [79]	(GPlay) 500.000+	Preguntas en solitario.
Trivia Deluxe [17]	(GPlay) 1.000.000+	No existen partidas de 1 jugador.
Quiz your English [80]	(GPlay) 1.000.000+	1vs1 con amigos eligiendo categoría, sin ranking.

Tabla C.1: Tabla comparativa del matchmaking de varias aplicaciones (parte 1)

Nombre	Partidas Multijugador	Tipo de matchmaking
Atriviate	■ 1vs1 (Con desconocidos y con amigos) ■ 1vs2-6 (Con desconocidos o con amigos/party seleccionados manualmente) ■ 1vs7 (Torneo): 8 jugadores con eliminaciones en cada encuentro 1vs1.	■ Ranked (Con desconocidos): Por nivel en las 1v1 y estadísticas de aciertos + ranking de TOP jugadores (global y semanal) ■ Privado (Con amigos).
Apalabrados	■ 1vs1 (Con desconocidos y amigos) (Casual): posible usar ventajas. ■ 1vs1 (Con desconocidos y amigos) (Rápido): turnos de 5 min. ■ 1vs1 (Con desconocidos y amigos) (Clásico): sin ventajas.	(Transparencia limitada) ■ Ranked (Con desconocidos): No se puede visualizar el rango del oponente. Se empareja por contrincantes del mismo rango. ■ Privado (Con amigos).
Preguntados	■ 1vs1 (Con desconocidos y amigos)	■ Ranked (Con desconocidos): Se empareja con mismo rango, aunque existe algún emparejamiento con otros rangos (Máxima diferencia: ±1 rango ■ Privado (Con amigos).
Preguntados 2	■ 1vs1 (Con desconocidos y amigos) ■ 1vs7 (Con desconocidos y amigos)	■ Ranked (Con desconocidos): Se empareja con mismo rango, aunque existe alguno con otros rangos. (Máxima diferencia: ±1 rango). Se empareja con un nivel similar. ■ Privado (Con amigos).
Mezcladitos 2	■ 1vs1 (Con desconocidos y amigos)	(Transparencia limitada) ■ Random (Con desconocidos): Mucha diferencia entre la experiencia. ■ Privado (Con amigos).
Lingo!	■ 1vs1 (Con desconocidos y amigos): Por turnos. ■ 1vs1 (Con desconocidos y amigos): Tiempo real.	■ Ranked (Con desconocidos): Por estadísticas de respuestas correctas, partidas ganadas, %victorias. ■ Privado (Con amigos).
Kryss - Batalla de palabras	■ 1vs1 (Con desconocidos y amigos) (Normal): Por turnos de 1 min, pudiendo salir de la partida y volver más tarde. ■ 1vs1 (Con desconocidos y amigos) (Rápido): Por turnos de 45 s, sin poder salir de la partida.	■ Ranked (Con desconocidos): Empareja con "Kryss Score"similar. Diferencia máxima: ±60 puntos. ■ Privado (Con amigos).
Scrabble Go	■ 1vs1 (Con desconocidos y amigos) (Normal): Por turnos. ■ 1vs1 (Con desconocidos) (Duels): Partidas rápidas por tiempo limitado y rondas limitadas.	■ Ranked (Con desconocidos): Emparejamientos con jugadores de nivel similar. Estadísticas sobre la puntuación media de cada palabra o la puntuación media final de cada partida. ■ Privado (Con amigos).
Fight List	■ 1vs1 (Con desconocidos y amigos)	■ Ranked (Con desconocidos): por estadísticas de respuestas correctas, respuestas con más puntuación, partidas ganadas, %victorias, clasificación (TOP %). ■ Privado (Con amigos).
Cuestionados	■ 1vs1 (Con desconocidos y amigos) (Clásico): Por turnos. ■ 1vs7 (Con desconocidos y amigos) (Torneo): Eliminaciones en partidas 1vs1. ■ 1vs1 (Con desconocidos y amigos) (Reto): Ronda de preguntas sueltas a ambos jugadores.	■ Ranked (Con desconocidos): Jugadores tienen ELO que indica su rango. ■ Ranked 2 (Con desconocidos): Matchmaking distinto en el modo Reto, con otro valor para este modo. ■ Privado (Con amigos).
Trivia Deluxe	1vs1 con usuarios aleatorios o amigos, respondiendo preguntas por turnos	■ Aleatorio con desconocidos. ■ Privado (Con amigos).
Quiz your English	■ 1vs1 entre usuarios del mismo nivel de inglés (B1,A1...), 5 preguntas de completar y más puntos a más rapidez de respuesta. ■ 1vs1 con usuarios buscados por nombre en todo el mundo.	■ Ranked (Con desconocidos): por nivel de inglés y categoría escogida. No tiene en cuenta clasificación de trofeos ni nivel. ■ Privado (Con amigos).

Tabla C.2: Tabla comparativa del matchmaking de varias aplicaciones (parte 2)

Apéndice D

Manual de despliegue

D.1. Despliegue del servidor

En esta sección se van a detallar los pasos para desplegar el servidor que contiene la API de matchmaking realizado en este TFG. En este caso el despliegue se ha realizado sobre un sistema operativo Linux: Ubuntu 20.04.4 LTS, de tal forma que para otros sistemas pueda haber pequeños cambios.

D.2. Instalación de git y clonación del repositorio

A continuación se enumeran los pasos a seguir para llevar el código del proyecto contenido en un repositorio git a nuestro ordenador.

1. En primer lugar, abriremos una terminal (no importa el directorio en el que situarnos)
2. A continuación, deberemos instalar git en nuestro dispositivo. Para ello utilizaremos el comando:

```
sudo apt-get install git
```

3. Después, nos moveremos al directorio dónde queremos clonar el repositorio
4. Clonamos el repositorio con el comando:

```
git clone <<url-proyecto-gitlab>>
```

(La url del repositorio que contiene el proyecto es:
”https://gitlab.inf.uva.es/marpana/tfg-panaderopalazuelos.git”).

5. En caso de no ver las carpetas del proyecto, utilizar el comando:

```
git checkout main
```

   Con este comando cambiaremos a la rama en la que se encuentra el proyecto.

D.3. Instalación de Node.js y módulos

En esta sección se detallan los pasos para la correcta instalación tanto de Node.js como de los módulos necesarios que este utiliza para la ejecución del programa. Las versiones utilizadas son v17.9.0 para Node.js y v8.5.5 para npm.

1. Abrimos una terminal y escribimos el siguiente comando:

```
curl -fsSL https://deb.nodesource.com/setup\_17.x | sudo -E bash -
```

   De esta forma se descarga la versión de Node.js correspondiente al parámetro ”setup” del comando anterior, pudiendo modificarse en caso de querer instalar otra versión diferente a la utilizada en este proyecto.

2. Una vez descargada la versión deseada ejecutamos el comando:

```
sudo apt install nodejs
```

   Así habremos instalado tanto Node.js como npm. Se puede comprobar que se ha realizado correctamente utilizando los comandos:

```
node -v
```

   y

```
npm -v
```

3. A continuación nos desplazamos al directorio que contiene el proyecto y ejecutamos el comando:

```
npm i
```

   de tal forma que se instalarán todos los módulos que utiliza mediante el package.json.

D.4. Instalación y configuración de Nginx

A continuación se enumeran los pasos a seguir para la correcta instalación de Nginx, que nos permitirá realizar las peticiones y llamadas entre los diferentes dispositivos.

1. Abrimos una terminal e instalamos nginx con el siguiente comando:

   ```
   sudo apt install nginx
   ```

   Podemos comprobar que se ha instalado correctamente con el comando:

   ```
   systemctl status nginx.service
   ```

   o bien realizando una llamada al servidor con:

   ```
   curl http://<<ip o dominio del servidor>>
   ```

   y recibiendo la página de bienvenida por defecto creada por el propio nginx.

2. Con el comando:

   ```
   sudo chown -R \$USER:\$USER /var/www/html
   ```

   nos hacemos propietarios del directorio especificado.

3. Después, se edita el fichero default que contendrá la configuración por defecto del servidor nginx que manejará nuestras llamadas. Para ello se usa el comando:

   ```
   sudo nano /etc/nginx/sites-available/default
   ```

4. Editar el fichero de manera que el resultado sea el siguiente:

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

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

       server_name 157.88.124.247:65212 www.157.88.124.247:65212;
   ```

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

El número de puerto será el correspondiente al puerto http, que por defecto es el 80 para la propia máquina. El server_name contiene la dirección IP de la máquina y en la cual se encuentra escuchando nginx junto con el puerto habilitado para las conexiones http externas, en este caso el 65212 debido a que este despliegue se está realizando sobre una máquina virtual y es el que ha sido levantado para permitir las conexiones remotas desde el exterior.

5. Guardamos los cambios realizados sobre el fichero y salimos del editor.
6. Ejecutamos el comando:

```
sudo ln -s /etc/nginx/sites-available/default /etc/nginx/sites-enabled/
```

Esta línea crea un enlace simbólico entre los dos archivos que hemos especificado.

7. Comprobamos que todo está bien sintácticamente y sin errores de escritura con:

```
sudo nginx -t
```

8. Reiniciamos el servicio nginx con el comando:

```
sudo systemctl restart nginx
```

Podemos comprobar si se ha completado correctamente con:

```
systemctl status nginx.service
```

y ver si se encuentra activo.

9. Por último, volvemos a abrir el fichero etc/nginx/sites-available/default y añadimos lo siguiente:

```
location / {
    proxy_pass http://localhost:3000;
    proxy_http_version 1.1;
    proxy_set_header Upgrade $http_upgrade;
    proxy_set_header Connection 'upgrade';
    proxy_set_header Host $host;
```

```
        proxy_cache_bypass $http_upgrade;
    }
    location /buscarPartidas {
        proxy_pass http://localhost:3000/buscarPartidas;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection "upgrade";
        proxy_set_header Host $host;
        proxy_set_header X-Real-Ip $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
        proxy_redirect off;
    }
```

Este código da las características a cada endpoint, en este caso al raíz y al de buscarPartidas, para que nginx redirija las llamadas hacia los puertos internos en los que tenemos levantado el servidor, en este caso el 3000. Habría que añadir el mismo código para cada endpoint siguiendo el ejemplo de /buscarPartidas.

D.5. Configuraciones en el Servidor App y Servidor MM

En esta sección se van a comentar una serie de configuraciones que habrá que añadir en el código del Servidor App para que la comunicación con el Servidor MM sea correcta y el flujo de ambos servidores funcione sin errores.

La más importante es la implementación de una llamada POST que devuelva el historial de partidas del periodo solicitado por el Servidor MM cuando se dispone a calcular las nuevas puntuaciones. Esta llamada recibe la fecha de inicio del periodo y realiza una consulta a la base de datos obteniendo los datos de las partidas finalizadas cuya fecha es posterior a dicha fecha de inicio. La forma de realizar esta modificación dependerá de cómo el servidor de la aplicación esté implementado. En el caso de nuestro proyecto, deberá hacerse en el fichero *index.js* del Servidor App. Un ejemplo del código se puede ver en la figura D.1.

```
app.post("/getHistorial", async (req,res) => {
  console.log("llamada para historial recibida")
  var fechap = new Date(req.body.fecha)
  var historialPartidas = await HistorialPartidasSCOP.find({$and:[{finalizada:1},{fecha:{$gte:fechap}}]})
  res.send(historialPartidas)
})
```

Figura D.1: Llamada API que devuelve el historial de partidas al Servidor MM

El formato de respuesta de esta llamada será un array de JSON con todas las partidas que cumplen los requisitos, estando formado cada uno por todos los atributos que presenta una partida y su valor

$(id\_partida, id\_jugador, resultado, id\_modo, finalizada, fecha)$.

Los cambios que hay que hacer en el Servidor MM son los que implican adaptar la llamada al Servidor App con la ruta adecuada, y a la periodicidad del cronjob para que sea la deseada por el desarrollador que implemente este matchmaking.

Para modificar la ruta de la llamada que solicita el historial de partidas para que se dirija al Servidor App se cambiará el atributo *GETHISTORIAL* recogido en el fichero de variables de entorno *.env* (ver figura D.2), y se cambiará la parte derecha del signo $=$ por el valor que se necesite asignar.

```
#rutas
HISTORIALPARTIDAS = "/home/mariopanpal/Escritorio/p.csv"
POBLARPARTIDAS = "/home/mariopanpal/Escritorio/1000/1000partidas.csv"
GETHISTORIAL = "http://localhost:3000/getHistorial"
```

Figura D.2: Fragmento con la sección de código a modificar con la ruta al Servidor App adecuada

En cuanto a la modificación de la periodicidad del cronjob deberemos situarnos en el fichero *index.js* del Servidor MM y buscar la sección de código que presenta la línea *cron.schedule*, alrededor de la línea 220 (ver figura D.3). Esta periodicidad viene expresada por medio de cinco $*$, y cada uno representa, leyendo de derecha a izquierda, día de la semana, mes, día del mes, hora y minuto (con la posibilidad de añadir un sexto que representa los segundos). La sintaxis de node-cron también permite establecer fórmulas que indiquen al job cuando lanzarse. En la figura D.3 se puede ver la fórmula $*/30$ en la sección de los minutos, lo cual significa que se ejecutará cada 30 minutos de forma periódica. Toda esta información y cómo operar con esta simbología para que se establezca la deseada por el desarrollador está recogida en el manual del propio módulo [53].

```
cron.schedule('*/30 * * * *', async function() {
    HistorialPartidas.deleteMany()
    var partidasTotales
    getHistorial().then(resp =>{
        partidasTotales = resp
        HistorialPartidas.insertMany(partidasTotales, (err, result) => {
            if (err) console.log(err);
            calculo()
        });
    })

})
```

Figura D.3: Fragmento de código que muestra el job utilizado

Apéndice E

Manual de uso

En esta parte se van a detallar los pasos a seguir para hacer funcionar la aplicación y cada una de las funcionalidades que presenta una vez se encuentra desplegado el sistema en la máquina.

En cada sección siguiente se mencionan los pasos a seguir desde cero pero el levantamiento de servidores puede reutilizarse y no es necesario reiniciarlos cada vez que se desee realizar otra acción.

E.1. Registrar jugador

Esta llamada registra un nuevo jugador en la lista de jugadores con sus datos de matchmaking por defecto.

Precondiciones:

- El formato de los parámetros de envío debe ser un objeto JSON con dos atributos: {id_jugador, id_modo}.
- El jugador no debe existir ya en la base de datos.
- El modo de juego debe existir.

Pasos:

- Acceder al directorio ServerMM y utilizar el comando: *npm start*. Deberá aparecer un mensaje con el servidor levantado y el puerto donde se encuentra.
- Acceder al directorio ClienteRegistrar y utilizar el comando: *python3 clienteRegistrar.py*.

E.2. PoblarJugadores

Esta llamada rellena la lista de jugadores con los jugadores contenidos en un fichero .csv. (En el repositorio se ha incorporado un directorio con ficheros de prueba).

Precondiciones:

- El formato de los parámetros de envío debe ser un objeto JSON con un atributo que será la ruta del fichero que se quiere insertar: {pathListaJugadores}.
- El formato del fichero debe ser, en primer lugar, una cabecera con los atributos establecidos para un jugador en el mismo orden, separados por comas y sin espacios y después los daros de cada jugador siguiendo el mismo patrón (Ver figura E.1)
- El fichero debe existir.
- La ruta debe ser correcta.

```
id_jugador,ptosELO,RD,periodosSinJugar,id_modo
1,1600,280,0,0
2,1713,189,0,0
3,1500,350,0,1
4,1610,300,1,1
5,1587,239,0,0
6,1711,130,0,2
7,1600,241,1,2
8,1574,200,0,0
9,1587,295,0,0
10,1543,333,0,2
```

Figura E.1: Imagen de un ejemplo del formato del fichero para poblar jugadores

Pasos:

- Acceder al directorio ServerMM y utilizar el comando: *npm start*. Deberá aparecer un mensaje con el servidor levantado y el puerto donde se encuentra.
- Acceder al directorio ClientePoblar y utilizar el comando: *python3 clientePoblar.py*.

E.3. BuscarPartida

Esta llamada lanza el flujo de búsqueda de rivales para un jugador y registra los datos de la partida resultante en la base de datos.

Precondiciones:

- El formato de los parámetros de envío debe ser un objeto JSON con dos atributos: {token, id_modo}.

- El jugador deberá estar ya registrado en la base de datos
- El modo de juego debe existir.

Pasos:

- Acceder al directorio ServerSim y utilizar el comando: *npm start*. Deberá aparecer un mensaje con el servidor levantado y el puerto donde se encuentra.
- Acceder al directorio ServerMM y utilizar el comando: *npm start*. Deberá aparecer un mensaje con el servidor levantado y el puerto donde se encuentra.
- Acceder al directorio Cliente y utilizar el comando: *python3 cliente.py*.

Apéndice F

Contenido del TFG

Junto con la memoria del TFG, cuyo nombre es MemoriaTFGMarioPanaderoPalazuelos.pdf podemos encontrar un repositorio de GitLab de la Escuela de Ingeniería Informática de la Universidad de Valladolid, accesible mediante la url: `https://gitlab.inf.uva.es/marpana/tfg-panaderopalazuelos.git`. Dicho repositorio incluye:

- ServerSim: Directorio que contiene la funcionalidad del Servidor App, los ficheros que utiliza y sus dependencias.
- ServerSim: Directorio que contiene la funcionalidad del Servidor MM, los ficheros que utiliza y sus dependencias.
- Cliente: Directorio que contiene el cliente que simula la búsqueda de partidas.
- ClienteRegistrar: Directorio que contiene el cliente que simula la inserción de un jugador en la base de datos.
- Cliente: Directorio que contiene el cliente que simula la funcionalidad de poblar la lista de jugadores mediante un fichero .csv.
- FicherosJugadores: Directorio que contiene cinco ficheros .csv para poblar la lista de jugadores (10, 100, 200, 500 y 1000 jugadores).
- .gitignore: Fichero que filtra qué extensiones se guardan en el repositorio en los pushes posteriores.
- Pruebas llamadas.postman_collection.json: Colección de Postman con las pruebas de las llamadas a cada endpoint del sistema.
- README.md: Fichero con información acerca del contenido del repositorio y su utilización.

Bibliografía

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

[2] Clash of pronunciations. Visitado: 2021-11-21. [Online]. Available: https://play.google.com/store/apps/details?id=uva.eca.simm.clashofpronunciation&hl=en_US&gl=US

[3] Ending support for multiplayer apis in play games services. Visitado: 2021-11-21. [Online]. Available: https://support.google.com/googleplay/android-developer/answer/9469745?hl=en

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

[5] J. G. Diéguez, *Migración de aplicación Android con Google Play Games a API REST*. Universidad de Valladolid, 2020. [Online]. Available: https://uvadoc.uva.es/handle/10324/50373

[6] En busca del matchmaking perfecto: cómo los juegos online utilizan la estadística para que no te frustres. Visitado: 2021-12-03. [Online]. Available: https://www.xataka.com/videojuegos/busca-matchmaking-perfecto-como-juegos-online-utilizan-estadistica-no-te-frustres

[7] Finding the perfect match. Visitado: 2021-12-03. [Online]. Available: https://www.guildwars2.com/en/news/finding-the-perfect-match/

[8] ¿Qué es el matchmaking y cómo funciona? Visitado: 2021-12-03. [Online]. Available: https://androidphoria.com/juegos/matchmaking-que-es-como-funciona

[9] Diseño iterativo, la metodología que perfeccionará tus proyectos. Visitado: 2021-12-12. [Online]. Available: https://medium.com/sue%C3%B1os-graficos/dise%C3%B1o-iterativo-la-metodolog%C3%ADa-que-perfeccionar%C3%A1-tus-proyectos-21034b0d277e

[10] Los 7 beneficios que tiene implantar una metodología agile. Visitado: 2021-12-12. [Online]. Available: https://www.equiposytalento.com/noticias/2018/12/13/los-7-beneficios-que-tiene-implantar-una-metodologia-agile

[11] C. Tejedor-García, *Design and evaluation of mobile computer-assisted pronunciation training tools for second language learning*. Universidad de Valladolid, 2016. [Online]. Available: https://uvadoc.uva.es/handle/10324/43663

[12] C. Tejedor-García, D. Escudero-Mancebo, E. Cámara-Arenas, C. González-Ferreras, and V. Cardeñoso-Payo, "TipTopTalk! mobile application for speech training using minimal pairs and gamification," in *Proc. IberSPEECH*, Lisbon, Portugal, Nov. 23–25, 2016, pp. 425–432.

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

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

[15] M. W. . M. Ćwil | Przemysław Chojecki | Kajetan Dąbrowski, *Analysis of Matchmaking Optimization Systems Potential in Mobile eSports*. University of Hawai'i at Mānoa, 2019. [Online]. Available: https://scholarspace.manoa.hawaii.edu/items/345848f3-8b73-4302-9fd5-4d4c767efd57

[16] M. F. Pramono, K. Renalda, H. Leslie, D. P. Kristiadi, and W. kusakunniran, *Matchmaking problems in MOBA games*. Exploration on Games and Gamers Workshop, SocInfo 2014, 2018. [Online]. Available: https://www.researchgate.net/publication/326845380_Matchmaking_problems_in_MOBA_games

[17] Trivia deluxe. Visitado: 2022-01-15. [Online]. Available: https://play.google.com/store/apps/details?id=com.etermax.triviagameshow&hl=es&gl=US

[18] Preguntados. Visitado: 2022-01-15. [Online]. Available: https://play.google.com/store/apps/details?id=com.etermax.preguntados.lite&hl=es_419&gl=MY

[19] M. Myślak and D. Deja, *Developing Game-Structure Sensitive Matchmaking System for Massive-Multiplayer Online Games*. Exploration on Games and Gamers Workshop, SocInfo 2014, 2014. [Online]. Available: https://www.researchgate.net/publication/268152085_Developing_Game-Structure_Sensitive_Matchmaking_System_for_Massive-Multiplayer_Online_Games

[20] Desarrollo: Mejorar los emparejamientos. Visitado: 2022-01-15. [Online]. Available: https://nexus.leagueoflegends.com/es-es/2018/03/dev-making-matchmaking-better/

[21] Clasificación de matchmaking. Visitado: 2022-01-22. [Online]. Available: https://www.ubisoft.com/es-es/game/rainbow-six/siege/news-updates/4hShcX2HZTG2ttIi3IIN9Y/clasificacin-de-matchmaking

[22] Sistema de clasificación de ajedrez. Visitado: 2022-01-30. [Online]. Available: https://hmong.es/wiki/Harkness_rating_system

[23] Sistema de calificación elo. Visitado: 2022-01-30. [Online]. Available: https://hmong.es/wiki/Elo_system

[24] Cs:go wiki: Matchmaking. Visitado: 2022-06-07. [Online]. Available: https://counterstrike.fandom.com/wiki/Matchmaking

[25] Hearthstone wiki: Matchmaking. Visitado: 2022-06-07. [Online]. Available: https://hearthstone.fandom.com/wiki/Matchmaking

[26] Call of duty warzone sbmm: qué es y cómo funciona la búsqueda por habilidad. Visitado: 2022-06-07. [Online]. Available: https://juegosadn.es/cod-warzone-sbmm-como-funciona-la-busqueda-por-habilidad-no-119773/

[27] The glicko system. Visitado: 2022-01-30. [Online]. Available: http://www.glicko.net/glicko/glicko.pdf

[28] Nodejs. Visitado: 2021-11-09. [Online]. Available: https://nodejs.org/es/

[29] ¿Qué es nodejs? Te contamos sus principales características. Visitado: 2022-06-07. [Online]. Available: https://memorandum.net/en/blog-post/que-es-nodejs-te-contamos-sus-principales-caracteristicas

[30] Mean, the full-stack javascript development quartet for web apps. Visitado: 2022-06-07. [Online]. Available: https://www.bbvaapimarket.com/en/api-world/mean-full-stack-javascript-development-quartet-web-apps/

[31] npm. Visitado: 2021-11-09. [Online]. Available: https://www.npmjs.com/

[32] Express. Visitado: 2021-11-09. [Online]. Available: https://expressjs.com/es/

[33] Mongodb. Visitado: 2021-11-09. [Online]. Available: https://www.mongodb.com/es

[34] Postman. Visitado: 2021-11-09. [Online]. Available: https://www.postman.com/

[35] Astah professional. Visitado: 2021-11-09. [Online]. Available: https://astah.net/products/free-student-license/

[36] Excel. Visitado: 2022-04-21. [Online]. Available: https://www.microsoft.com/es-es/microsoft365/excel

[37] Git. Visitado: 2022-06-05. [Online]. Available: https://git-scm.com/

[38] Gitlab. Visitado: 2022-06-05. [Online]. Available: https://gitlab.inf.uva.es/

[39] Google Chrome. Visitado: 2022-06-30. [Online]. Available: https://www.google.com/intl/es_es/chrome/

[40] Google Drive. Visitado: 2022-06-30. [Online]. Available: https://drive.google.com/

[41] Jest. Visitado: 2021-11-09. [Online]. Available: https://jestjs.io/es-ES/

[42] Jitsi. Visitado: 2022-06-30. [Online]. Available: https://meet.jit.si/

[43] Mozilla firefox. Visitado: 2022-06-30. [Online]. Available: https://www.mozilla.org/es-ES/firefox/new/

[44] Nginx. Visitado: 2021-11-09. [Online]. Available: https://www.nginx.com/

[45] Overleaf. Visitado: 2022-06-30. [Online]. Available: https://es.overleaf.com/

[46] Python. Visitado: 2021-11-09. [Online]. Available: https://www.python.org/downloads/

[47] Telegram. Visitado: 2022-06-30. [Online]. Available: https://web.telegram.org/

[48] Visual studio code. Visitado: 2022-05-13. [Online]. Available: https://code.visualstudio.com/docs

[49] Axios. Visitado: 2021-11-09. [Online]. Available: https://github.com/axios/axios

[50] csvtojson. Visitado: 2021-11-09. [Online]. Available: https://www.npmjs.com/package/csvtojson

[51] dotenv. Visitado: 2021-11-09. [Online]. Available: https://www.npmjs.com/package/dotenv

[52] Mongoose. Visitado: 2021-11-09. [Online]. Available: https://mongoosejs.com/

[53] Node-cron. Visitado: 2021-11-09. [Online]. Available: https://www.npmjs.com/package/node-cron

[54] Nodemon. Visitado: 2021-11-09. [Online]. Available: https://www.npmjs.com/package/nodemon

[55] supertest. Visitado: 2021-11-09. [Online]. Available: https://www.npmjs.com/package/supertest

[56] Cuánto gana un ingeniero informático. Visitado: 2022-01-23. [Online]. Available: https://universidadeuropea.com/blog/cuanto-gana-un-ingeniero-informatico/

[57] Astah professional price. Visitado: 2021-11-09. [Online]. Available: https://www.componentsource.com/es/product/astah-uml/prices?pf=new

[58] ¿Qué es una api de rest? Visitado: 2022-01-23. [Online]. Available: https://www.redhat.com/es/topics/api/what-is-a-rest-api

[59] Arquitectura de capas para nodejs. Visitado: 2022-01-23. [Online]. Available: https://www.geeksforgeeks.org/how-to-create-custom-middleware-in-express/

[60] Patrones de diseño y ejemplos de uso en javascript. Visitado: 2022-01-23. [Online]. Available: https://tecnops.es/patrones-de-diseno-y-ejemplos-de-uso-en-javascript/

[61] How to build middleware for node.js: A complete guide. Visitado: 2022-01-23. [Online]. Available: https://www.turing.com/kb/building-middleware-for-node-js

[62] Patrones de gestión de estado en javascript: compartir datos entre componentes. Visitado: 2022-01-23. [Online]. Available: https://ull-mii-sytws-1920.github.io/tema2-async/event-emitter

[63] How to create custom middleware in express? Visitado: 2022-01-23. [Online]. Available: https://ctrly.blog/es/arquitectura-capas/

[64] Estrategias de modelado de datos en mongodb. Visitado: 2022-03-06. [Online]. Available: https://adrianalonso.es/desarrollo-web/estrategias-de-modelado-en-mongodb/

[65] Base de datos mongo en la nube. Visitado: 2022-03-06. [Online]. Available: https://www.mongodb.com/es/atlas/database

[66] ¿Qué es una prueba de caja negra? técnicas, muestras y tipos. Visitado: 2022-03-06. [Online]. Available: https://ebooksonline.es/que-es-una-prueba-de-caja-negra-tecnicas-muestras-y-tipos/

[67] Pruebas de carga. Visitado: 2022-03-06. [Online]. Available: https://www.loadview-testing.com/es/pruebas-de-carga/

[68] Códigos de estado de respuesta http. Visitado: 2022-03-06. [Online]. Available: https://developer.mozilla.org/es/docs/Web/HTTP/Status

[69] Paralelismo (informática). Visitado: 2022-03-06. [Online]. Available: https://es.wikipedia.org/wiki/Paralelismo_(inform%C3%A1tica)

[70] Rendimiento de algoritmos y notación big-o. Visitado: 2022-06-07. [Online]. Available: https://www.campusmvp.es/recursos/post/Rendimiento-de-algoritmos-y-notacion-Big-O.aspx

[71] Atriviate. Visitado: 2021-12-06. [Online]. Available: https://play.google.com/store/apps/details?id=aul.irm.triviados&hl=es&gl=US

[72] Apalabrados. Visitado: 2021-12-06. [Online]. Available: https://play.google.com/store/apps/details?id=com.etermax.apalabrados.lite&hl=es&gl=US

[73] Preguntados 2. Visitado: 2021-12-06. [Online]. Available: https://play.google.com/store/apps/details?id=com.etermax.trivia.preguntados2&hl=es&gl=US

[74] Mezcladitos 2. Visitado: 2021-12-06. [Online]. Available: https://play.google.com/store/apps/details?id=com.etermax.wordcrack.lite&hl=es&gl=US

[75] Lingo. Visitado: 2021-12-06. [Online]. Available: https://play.google.com/store/apps/details?id=ru.ipartner.lingo&hl=es&gl=US

[76] Kryss - batalla de palabras. Visitado: 2021-12-06. [Online]. Available: https://play.google.com/store/apps/details?id=app.kryds.android&hl=es&gl=US

[77] Scrabble go. Visitado: 2021-12-06. [Online]. Available: https://play.google.com/store/apps/details?id=com.pieyel.scrabble&hl=es&gl=US

[78] Fightlist. Visitado: 2021-12-06. [Online]. Available: https://play.google.com/store/apps/details?id=fr.two4tea.fightlist&hl=es&gl=US

[79] Cuestionados. Visitado: 2021-12-06. [Online]. Available: https://play.google.com/store/apps/details?id=com.diablins.android.leagueofquiz&hl=es&gl=US

[80] Quiz Your English. Visitado: 2021-12-06. [Online]. Available: https://play.google.com/store/apps/details?id=org.cambridgeenglish.bravi.quiz&hl=es_HN&gl=US