



TRABAJO FIN DE GRADO
INGENIERÍA INFORMÁTICA

Wordle+

Autor

David Correa Rodríguez

Director

Juan Manuel Fernández Luna



Escuela Técnica Superior de Ingenierías Informáticas y
Telecomunicaciones

—
Granada, Julio de 2023

Índice

Wordle+	1
Capítulo I. Introducción	5
I.1. Motivación	5
I.2. Descripción de la aplicación	6
I.3. Estado del arte	7
I.3.1. Metodologías y enfoques	7
I.3.2. Proyectos similares	8
I.4. Objetivos	9
I.5. Descripción de la memoria	10
Capítulo II. Historias de usuario. Planificación y presupuesto	12
II.1. Descripción de la metodología	12
II.2. Historias de usuario	14
II.3. Sprints	28
II.3.1. Planificación de Sprints	28
II.3.2. División en tareas	33
II.4. Presupuesto	45
Capítulo III. Desarrollo	47
III.1. Elección de las tecnologías	47
III.1.1. Frontend	47
III.1.2. Backend	49
III.1.3. Despliegue	50
III.1.4. Control de versiones y seguimiento del Sprint	51
III.2. Diseño de la base de datos	56
III.2.1. Análisis del contexto	56
III.2.2. Diagrama Entidad-Relación	58
III.2.3. Paso a tablas	60
III.2.4. Fusiones	62
III.2.5. Normalización	66
III.2.5.1. Primera Formal Normal (1FN)	66
III.2.5.2. Segunda Formal Normal (2FN)	68
III.2.5.3. Tercera Formal Normal (3FN)	69
III.2.5.4. Forma Normal de Boyce-Codd (FNBC)	71
III.3. Implementación de los Sprints	73
III.3.1. Sprint 1	73
III.3.1.1. Infraestructura	73
III.3.1.2. Registro de usuarios	79

III.3.1.3. Interfaz de usuario	86
III.3.1.4. Revisión y retrospectiva	87
III.3.2. Sprint 2.....	87
III.3.2.1. Inicio y cierre de sesión	88
III.3.2.2. Interfaz de usuario	91
III.3.2.3. Revisión y retrospectiva	92
III.3.3. Sprint 3.....	92
III.3.3.2. Registro de partidas.....	93
III.3.3.2. Maquetación HTML	95
III.3.3.3. Lógica del juego.....	96
III.3.3.4. Interfaz de usuario	98
III.3.3.5. Revisión y retrospectiva	100
III.3.4. Sprint 4.....	100
III.3.4.1. Página principal (Main)	101
III.3.4.2. Wordle avanzado.....	104
III.3.4.3. Notificaciones	105
III.3.4.4. Modificación de información personal.....	107
III.3.4.5. Revisión y retrospectiva	109
III.3.5. Sprint 5.....	110
III.3.5.1. Creación del torneo	111
III.3.5.2. Listado de torneos	112
III.3.5.3. Revisión y retrospectiva	114
III.3.6. Sprint 6.....	115
III.3.6.1. Participaciones	115
III.3.6.2. Lógica del administrador	116
III.3.6.3. Revisión y retrospectiva	118
III.3.7. Sprint 7.....	118
III.3.7.1. Modelado de amigos y peticiones.....	119
III.3.7.2. Vistas de amigos y peticiones.....	122
III.3.7.3. Revisión y retrospectiva	125
III.3.8. Sprint 8.....	125
III.3.8.1. Modelado de partidas multijugador	126
III.3.8.2. Vistas del multijugador.....	128
III.3.8.3. Vistas de las partidas pendientes	130
III.3.8.4. Revisión y retrospectiva	131
III.3.9. Sprint 9.....	132
III.3.9.1. Lista de partidas completadas	132
III.3.9.2. Lista de torneos inscritos	135

III.3.9.3. Rondas y partidas.....	136
III.3.9.4. Jugando al torneo	140
III.3.9.5. Revisión y retrospectiva.....	141
III.3.10. Sprint 10.....	141
III.3.10.1. Lista de Wordles completados.....	142
III.3.10.2. Ranking.....	144
III.3.10.3. Revisión y retrospectiva.....	146
III.3.11. Sprint 11.....	147
III.3.11.1. Eliminación de amigo y de solicitud de amistad.....	147
III.3.11.2. Perfil de amigo.....	149
III.3.11.3. Revisión y retrospectiva.....	150
III.3.12. Sprint 12.....	151
III.3.12.1. Revisión y retrospectiva.....	151
Capítulo IV. Conclusiones y trabajos futuros.....	152
Bibliografía	154

CAPÍTULO I. INTRODUCCIÓN

I.1. Motivación

La motivación detrás de este proyecto se basa en varios aspectos clave que me impulsan a reimaginar y ampliar el juego Wordle, así como a aprender nuevas tecnologías y contribuir al conocimiento en el campo del desarrollo de aplicaciones multiplataforma.

El juego Wordle es ampliamente conocido y disfrutado por numerosas personas. Consiste en un juego en el que los jugadores intentan adivinar una palabra de cinco letras en un número limitado de intentos. Cada intento se compara con la palabra objetivo, y se muestra información sobre las letras correctas en la posición correcta (letras verdes) y las letras correctas en la posición incorrecta (letras amarillas). El concepto del juego es sencillo, lo que me lleva a buscar formas de innovar y mejorar la experiencia de juego. A través de la adición de funcionalidades como la personalización del tamaño de las palabras, el modo multijugador, el sistema de amistades y el desarrollo de torneos, aspiro a brindar a los jugadores una experiencia más enriquecedora y emocionante. Creo que estas mejoras pueden aumentar el interés en el juego y atraer a una base de jugadores más amplia, proporcionándoles nuevas formas de disfrutar y desafiarse a sí mismos y poder llegar a todo tipo de públicos.

Además de la mejora del juego en sí, este proyecto me ofrece la oportunidad de aprender nuevas tecnologías de desarrollo de aplicaciones multiplataforma. A través del desarrollo autodidacta, podré adquirir conocimientos en el uso de marcos de trabajo y herramientas específicas para crear aplicaciones que funcionen en diversas plataformas, como web, móvil y escritorio, utilizando como base los amplios conocimientos obtenidos durante la carrera. Este aprendizaje no solo ampliará mis habilidades técnicas, sino que también me permitirá estar preparado para futuros proyectos en el campo de la informática, donde la capacidad de desarrollar aplicaciones multiplataforma es cada vez más relevante.

Además, tengo el deseo de compartir el conocimiento adquirido con la comunidad. Al alojar el proyecto en un repositorio de GitHub, proporcionaré un acceso abierto al código fuente y los recursos relacionados con el juego Wordle mejorado. Esto permitirá a otros desarrolladores aprender de mi trabajo, colaborar en el proyecto y construir sobre él, lo que puede fomentar la innovación y el avance en el campo de los juegos y las aplicaciones. Contribuir al conocimiento y la divulgación es una forma significativa de retribuir a la comunidad de desarrollo y permitir que otros se beneficien de las soluciones y enfoques que he descubierto durante mi investigación y desarrollo.

I.2. Descripción de la aplicación

El siguiente proyecto tratará de desarrollar el típico juego de palabras Wordle, pero añadiéndole aspectos multijugador y competitivos para que tenga más entidad de juego completo. Se podrán gestionar torneos entre jugadores y retos 1vs1, gestionando un ranking de jugadores e históricos de partidas por jugador. Los jugadores se dividirán por rangos, en relación con la experiencia de juego de cada jugador (Wordles, torneos y retos completados). Las partidas podrán ser más personalizables, como por ejemplo poder elegir la longitud de la palabra en cuestión.

Por tanto, podemos dividir las características y funcionalidades de este proyecto en 3 niveles, en donde el primer nivel corresponde a las funcionalidades obligatorias, el segundo nivel a las funcionalidades muy recomendadas, y el tercer nivel corresponde a aquellas funcionalidades extra que se añadirán si el tiempo y las condiciones lo permiten:

Nivel 1:

- Registro de jugadores y de administradores.
- Inicio de sesión.
- Poder jugar la partida clásica de Wordle.
- Poder configurar la partida en cuestión (longitud de palabra).
- Los jugadores tendrán atributos asociados: Nivel de Experiencia, Rango (asociado al nivel de experiencia: hierro, bronce, plata, oro, platino), Wordles completados, retos ganados, torneos ganados
- Gestión de retos 1vs1 entre usuarios amigos.
- Gestión de torneos: se podrán crear salas de torneos de distintos tamaños en donde los usuarios podrán unirse; se podrán organizar torneos por los administradores quienes seleccionarán los participantes de los torneos.

Nivel 2:

- Gestionar un ranking con los mejores jugadores, que se divida por filtros (partidas ganadas, retos ganados, torneos ganados, nivel de experiencia).
- Gestionar un histórico de Wordles completados y retos 1vs1 por jugador.
- Los torneos, se dividirán por longitud de palabras, en donde todas las partidas del torneo se harán con esa longitud de palabra.

Nivel 3:

- Mejoras de rendimiento, optimización de código, minimizar la carga de la aplicación.
- Mejoras visuales y de diseño.
- Implementación de multilenguaje (Español/Inglés).
- Implementar un modo oscuro (*dark mode*).
- Desarrollar, además de una app web, una aplicación móvil, tanto para Android como para iOS, y poder jugar al juego mediante un smartphone.

Aunque el proyecto está basado en Wordle, cabe destacar que la segunda funcionalidad del primer nivel (poder jugar la partida clásica) y la penúltima del nivel 3 (implementar un modo oscuro) son las únicas implementadas en el juego original.

I.3. Estado del arte

El estado del arte del proyecto abarca una amplia gama de aspectos relacionados con el desarrollo web, los juegos de adivinanzas de palabras y las tecnologías utilizadas en aplicaciones multiplataforma. En esta sección, se explorarán metodologías, enfoques y tecnologías relevantes para el desarrollo de la aplicación Wordle mejorada.

I.3.1. Metodologías y enfoques

En el desarrollo de aplicaciones web multiplataforma, se utilizan diferentes metodologías y enfoques, algunos de los cuales se describen a continuación:

1. Basado en componentes¹: se centra en la creación de componentes reutilizables para construir aplicaciones web multiplataforma. Los desarrolladores crean componentes independientes que se pueden usar en diferentes partes de la aplicación.
2. Basado en microservicios²: la aplicación se divide en pequeños servicios independientes que se comunican entre sí para ofrecer una funcionalidad completa. Cada microservicio se desarrolla, prueba e implementa de manera independiente, lo que permite una mayor flexibilidad y escalabilidad.
3. Metodología ágil⁵: la metodología ágil se basa en la colaboración, la iteración y la entrega continua. En lugar de seguir un plan rígido, los desarrolladores trabajan en ciclos cortos y entregan nuevas funcionalidades en cada iteración.
4. Basado en contenedores³: la aplicación se ejecuta dentro de contenedores aislados que se pueden mover fácilmente entre diferentes

plataformas. Los contenedores ofrecen una forma eficiente y segura de distribuir y ejecutar aplicaciones en diferentes entornos.

5. Basado en el modelo vista controlador (MVC)⁴: se divide la aplicación en tres partes: el modelo (los datos), la vista (la interfaz de usuario) y el controlador (la lógica de negocio). Esta división permite una mayor separación de preocupaciones y una mayor reutilización de código.

1.3.2. Proyectos similares

Hay varias aplicaciones y juegos en línea que son similares a Wordle en términos de su mecánica de juego y objetivos. Aquí hay algunas opciones:

- Hangman: Es un juego clásico que implica adivinar una palabra oculta letra por letra antes de que se dibuje un ahorcado completo. Hangman también es un juego de palabras simple y divertido que se puede jugar en línea o en papel.
- Typing.com: Este sitio web ofrece varios juegos de mecanografía en línea para mejorar tus habilidades de escritura y velocidad de escritura. Los juegos tienen diferentes niveles de dificultad y objetivos, pero todos están diseñados para mejorar la precisión y velocidad al escribir palabras.
- 7 Little Words: Este juego de palabras es una mezcla de crucigrama y búsqueda de palabras. Consiste en adivinar siete palabras que están relacionadas entre sí, pero se presentan como pistas en lugar de definiciones. Los jugadores tienen que encontrar las palabras a partir de las letras y pistas que se les dan.
- Wordament: Es un juego de búsqueda de palabras en línea que permite a los jugadores encontrar tantas palabras como sea posible en un tablero de 4x4 letras. Wordament también permite a los jugadores competir contra otros jugadores en línea y comparar sus puntajes.
- SpellTower: Es un juego de palabras que consiste en hacer palabras de letras dispuestas en una cuadrícula. Cada letra utilizada se elimina del tablero y el objetivo es eliminar todas las letras. El juego ofrece varios modos de juego y niveles de dificultad.

Por tanto, existen juegos similares a Wordle. Sin embargo, la propuesta es ampliar las funcionalidades y características de dicho juego, reimplementándolo y utilizando las tecnologías punteras en el desarrollo de aplicaciones multiplataforma utilizando las ventajas que ofrecen, permitiendo una mayor interacción entre los diferentes jugadores mediante las partidas 1 contra 1 y los torneos.

1.3.2. Tecnologías y herramientas

En cuanto a las tecnologías y frameworks para el desarrollo web, existen varias opciones populares como React²⁰, Angular²¹, Vue.js²⁹ y Django. Cada una de ellas tiene sus propias ventajas y desventajas en términos de rendimiento, escalabilidad, facilidad de uso y comunidad de desarrolladores. Evaluar estas tecnologías permitirá tomar decisiones fundamentadas sobre cuál utilizar en el proyecto.

Considerando la naturaleza del proyecto, que busca una solución multiplataforma, es necesario examinar las tecnologías y herramientas para el desarrollo de aplicaciones multiplataforma. En este contexto, destacan Ionic²⁸, React Native²⁴ y Flutter¹⁰. Estas tecnologías permiten desarrollar aplicaciones que se ejecutan en diferentes sistemas operativos y dispositivos, lo que brinda una amplia cobertura de usuarios. Comparar y contrastar estas tecnologías ayudará a seleccionar la más adecuada para el proyecto Wordle mejorado.

El diseño de interfaces de usuario es otro aspecto crucial a considerar. Investigar principios y mejores prácticas de diseño de interfaces de usuario, como el uso de patrones de diseño, la accesibilidad, la usabilidad y la estética, permitirá crear una experiencia de usuario atractiva y fácil de usar. La aplicación de estos principios en el diseño de la interfaz del juego Wordle mejorado mejorará la experiencia de juego y la satisfacción del usuario.

En cuanto a las soluciones de almacenamiento de datos, se deben considerar diferentes enfoques, como bases de datos relacionales, bases de datos NoSQL y sistemas de almacenamiento en la nube. Evaluar las ventajas y desventajas de cada opción en términos de escalabilidad, rendimiento y seguridad ayudará a seleccionar la solución más adecuada.

I.4. Objetivos

El presente proyecto tiene como objetivo principal la reimplementación del juego Wordle con la adición de diversas funcionalidades, así como el logro de metas específicas que contribuyan al enriquecimiento de la experiencia de juego y al aprendizaje de nuevas tecnologías. Los objetivos de este proyecto se dividen en las siguientes áreas:

1. Reimplementación del juego Wordle: El primer objetivo es desarrollar una versión mejorada del juego Wordle, que permita a los jugadores personalizar el tamaño de las palabras a adivinar. Esto implica adaptar la lógica del juego existente para manejar diferentes longitudes de palabras y ajustar la interfaz de usuario para mostrar correctamente las palabras seleccionadas. Además, se busca optimizar el rendimiento del juego para garantizar una experiencia fluida y ágil.
2. Implementación de funcionalidades adicionales: Un objetivo importante es añadir funcionalidades que mejoren la experiencia de juego. Esto incluye

la posibilidad de jugar partidas 1vs1, donde dos jugadores compiten para adivinar palabras en un tiempo limitado. Además, se busca desarrollar un sistema de amigos y peticiones de amistad, que permita a los jugadores conectarse y jugar entre sí. Esta funcionalidad requerirá la implementación de un sistema de gestión de usuarios y la integración de un sistema de notificaciones.

3. Asociación de valores a los jugadores: Otro objetivo consiste en asociar valores a los jugadores, como experiencia y número de Wordles ganados. Esto implica crear un sistema de seguimiento y almacenamiento de datos de jugadores, así como una interfaz que muestre estos valores de manera visible. La intención es proporcionar a los jugadores una forma de medir su progreso y compararse con otros jugadores.
4. Desarrollo de un sistema de torneos y clasificación: Se plantea la implementación de un sistema de torneos en el juego Wordle, donde los jugadores puedan inscribirse y competir entre sí. Además, se buscará desarrollar un sistema de clasificación que muestre los mejores jugadores según parámetros relevantes, como la puntuación total o la tasa de victorias. Este sistema permitirá a los jugadores participar en competiciones emocionantes y fomentará un sentido de competitividad y logro.
5. Divulgación del conocimiento y contribución a la comunidad: Como objetivo adicional, se busca compartir el conocimiento adquirido durante el desarrollo del proyecto. Esto implica alojar el código fuente y los recursos relacionados en un repositorio de GitHub, dando detalles de implementación en las *Issues* del repositorio, proporcionando así un acceso abierto para que otros desarrolladores puedan aprender, colaborar y construir sobre el proyecto. Se espera que esta divulgación contribuya al avance en el campo del desarrollo de juegos y aplicaciones, y promueva el intercambio de ideas y soluciones innovadoras.
6. Abstracción de la infraestructura del proyecto: Un objetivo adicional es la completa abstracción de la infraestructura del sistema sobre el que se aloja. Mediante el uso de contenedores, se busca proporcionar flexibilidad, portabilidad y escalabilidad al proyecto. Esto permitirá que el juego Wordle y sus componentes se ejecuten de manera consistente y reproducible en diferentes entornos, independientemente de la configuración de hardware o software subyacente. El uso de contenedores también facilitará la gestión de dependencias y actualizaciones del proyecto, simplificando así el despliegue y la puesta en marcha en distintos entornos.

I.5. Descripción de la memoria

El capítulo I de la memoria comienza con una introducción al proyecto, donde se detallará la motivación detrás del mismo y se proporcionará una descripción general de la aplicación a desarrollar. A continuación, se presenta el estado del

arte, donde se exploran las metodologías y enfoques utilizados en proyectos similares. Esto permitirá contextualizar el trabajo realizado y resaltar las contribuciones y novedades del proyecto.

Posteriormente, se plantean los objetivos del proyecto, que abarcan desde la reimplementación del juego Wordle con funcionalidades adicionales hasta la divulgación del conocimiento y la contribución a la comunidad de desarrolladores. Estos objetivos serán fundamentales para guiar el desarrollo y evaluar el éxito del proyecto.

A partir del capítulo II, se aborda la planificación y presupuesto del proyecto, comenzando con la descripción de la metodología utilizada y la presentación de las historias de usuario. Se detallará la división de los sprints, la planificación de las tareas y se establecerá un presupuesto (por completar) que permita estimar los recursos necesarios para llevar a cabo el proyecto de manera exitosa.

En el capítulo III, se describe el desarrollo del proyecto, comenzando con la elección de las tecnologías utilizadas tanto en el frontend como en el backend de la aplicación. Se aborda el diseño de la base de datos, desde el análisis del contexto hasta la normalización de esta. Posteriormente, se presenta la implementación de los Sprints, donde se detallará cómo se desarrollaron y se llevaron a cabo las diferentes funcionalidades y componentes del juego Wordle mejorado, además de insertar pruebas y figuras de los resultados obtenidos en cada Sprint.

En el capítulo IV, se presentan las conclusiones y los trabajos futuros del proyecto. En este apartado, se resumirán los resultados obtenidos y se realizará una evaluación de los objetivos alcanzados. Se discutirán las lecciones aprendidas durante el desarrollo del proyecto y se destacarán las contribuciones y las posibles mejoras que podrían realizarse en el futuro. Además, se propondrán posibles trabajos futuros que puedan surgir a partir de este proyecto, como la expansión de las funcionalidades, la optimización del rendimiento o la exploración de nuevas tecnologías.

CAPÍTULO II. HISTORIAS DE USUARIO. PLANIFICACIÓN Y PRESUPUESTO

II.1. Descripción de la metodología

A comienzos de la década de los 90 muchos desarrolladores software se percataron de que los ciclos de producción y los métodos que ofrecían las metodologías clásicas no estaban dando resultados satisfactorios. En un entorno lleno de incertidumbre, tanto empresarial como económico, muchos proyectos software se veían cancelados antes de lanzarse al mercado.

Este fue el origen de la creación de las metodologías ágiles, que florecieron a partir de la década de los 2000 con la creación del Manifiesto Ágil⁵. Los valores de la metodología ágil sostienen que:

- **Las personas y las interacciones** antes que los procesos y las herramientas
- **El software en funcionamiento** antes que la documentación exhaustiva
- **La colaboración con el cliente** antes que la negociación contractual
- **La respuesta ante el cambio** antes que el apego a un plan

En donde los puntos de la izquierda son más importantes que los de la derecha. Por tanto, lo que permite este tipo de metodologías es evitar desarrollar sistemas software en secuencia por fases y utilizar un proceso de desarrollo simultáneo y constante.

Con la filosofía asentada, se crearon diversos marcos ágiles enfocados al desarrollo del software, como Scrum⁶, Kanban⁷ o Programación Extrema (XP), que actualmente son piezas fundamentales del [DevOps](#) o la integración continua/implementación continua [CI/CD](#)⁸.

Scrum² es uno de los marcos de trabajo más utilizados en la actualidad, que permite el trabajo colaborativo entre equipos de desarrollo de software, aunque sus valores y principios se pueden aplicar a diversos tipos de trabajo colaborativo. Scrum² es un marco de trabajo heurístico, que se basa en el aprendizaje y continuo y en la readaptación de los factores fluctuantes del equipo. Supone que el equipo no sabe las variables iniciales de un proyecto y

que evolucionará con él. Scrum² hace uso de “artefactos”, que son herramientas para solucionar un problema. Esta metodología utiliza tres de ellos:

- **Product backlog:** es la pila o la lista inicial de las tareas que tienen que desarrollarse en el proyecto. Esta lista es creada por propietario del producto junto con el equipo de desarrollo, en donde se incluyen requisitos, funcionalidades y mejoras del producto. La lista no es absoluta, si no que se revisa y se cambia por el propietario del producto, ante las fluctuaciones del mercado y los requisitos del propietario.
- **Sprint backlog:** se trata de la lista de elementos, historias de usuario, seleccionada por el equipo de trabajo para su implementación en el sprint actual. Un sprint es un pequeño periodo en el cual el equipo de desarrollo implementa y distribuye un incremento del producto (una versión del producto). Los sprints suelen tener una duración de 2 semanas, aunque puede cambiar según el equipo o el trabajo a realizar. Esta lista de elementos del sprint es flexible y puede evolucionar con el sprint.
- **Incremento:** es el producto final utilizable de un sprint, es decir, el resultado del sprint. La definición tampoco es rígida y depende del contexto y el entorno, puede ser que un incremento no suponga una versión del producto final, si no una parte de él.

Una característica fundamental en la metodología Scrum² es el **scrum diario** o reunión rápida, de unos 10-15 minutos de duración, en donde el equipo hace hincapié en lo que se debe hacer ese día, además de comentar qué se hizo el día anterior. También se utiliza para expresar inquietudes e impedimentos del sprint actual.

Al finalizar un sprint se suele hacer una **revisión del sprint** en donde el equipo demuestra e inspecciona el incremento, resultado de haber desarrollado los elementos del backlog del sprint. Esta revisión también es útil para que el propietario vea el incremento y decida si lanzarlo al mercado o no.

Otro aspecto importante que destacar es la **retrospectiva del sprint** que sirve para que el equipo documente y analice los aspectos del sprint que han funcionado y los que no. En esta etapa el equipo puede analizar lo que salió mal y evolucionar para un mejor desarrollo software.

En esta metodología existen tres figuras importantes resaltables:

- El propietario del producto o *Product Owner* (PO): son los que más conocen el producto y lo proponen. Se centran en los aspectos empresariales, de los clientes y del mercado. Básicamente define el backlog del producto, está en contacto constante con el equipo de desarrollo y decide cuándo lanzar el producto.

- El experto en scrum o *Scrum Master* (SM): son los que proporcionan formación al equipo y a los propietarios del producto para refinar su práctica. Básicamente planifica los recursos necesarios para realizar los sprints, las reuniones, las revisiones y las retrospectivas.
- El equipo de desarrollo: son los que realizan el trabajo y desarrollan el producto en varios sprints. Los miembros se reorganizan y aprenden entre ellos para tener una actitud positiva colectiva.

Por tanto, para este proyecto, se utilizará una metodología Scrum² adaptado a 1 sola persona, es decir, tanto el dueño del producto, el Scrum Master como el equipo de desarrollo lo conformaré yo mismo, tomando los diferentes roles y adoptando diferentes perspectivas según el rol.

Se seguirán las etapas de planificación (product backlog), ejecución (sprints, pila de sprint, probablemente cortos, de 2 semanas) y control (revisiones con el profesor y retrospectiva). El profesor podría tomar de forma provisional el rol de cliente y juzgar de forma objetiva los resultados obtenidos.

Esta cuestión ha sido consultada a la profesora de la asignatura Metodologías de Desarrollo Ágil (mlra@ugr.es).

Al ser un proyecto largo, me beneficiaré de las diversas versiones que permite la metodología scrum, para así obtener un producto más refinado y de calidad. También haré toma de contacto con esta metodología, muy utilizada actualmente en el mercado y en multitud de empresas.

II.2. Historias de usuario

Los puntos de historia (PH) son una medida utilizada para estimar el esfuerzo relativo necesario para desarrollar una funcionalidad o característica específica del producto. Esta unidad de medida que ayuda al equipo de desarrollo a determinar la complejidad y el tamaño de las tareas que deben llevar a cabo.

Los puntos de historia son una forma de estimación relativa en la que se asigna un valor numérico a cada historia de usuario en función de su complejidad y esfuerzo percibidos en comparación con otras historias. La escala utilizada para los puntos de historia puede variar de un equipo a otro, pero generalmente se utiliza una escala de Fibonacci o una escala de números enteros.

Épica	Ident.	Título	Estimación	Prioridad
Registro y autenticación	HU.1	Como jugador quiero poder registrarme	2	1
	HU.2	Como administrador quiero poder registrarme	2	1
	HU.3	Como usuario (jugador y administrador) quiero poder iniciar sesión	2	1
	HU.4	Como usuario quiero poder cerrar sesión	1	1
Juego individual	HU.5	Como jugador quiero poder echar un Wordle clásico	8	1
	HU.6	Como jugador quiero poder seleccionar la longitud de la palabra de la partida clásica	2	2
Torneos y gestión de eventos	HU.7	Como administrador quiero crear una sala abierta de un torneo	5	2
	HU.8	Como administrador quiero poder seleccionar el tamaño del torneo (número de participantes)	2	2
	HU.9	Como administrador quiero poder seleccionar la longitud de la palabra del torneo	2	2
	HU.26	Como jugador quiero poder ver la lista de torneos	2	2
	HU.10	Como administrador quiero poder crear un torneo y preseleccionar los participantes	3	2
	HU.11	Como jugador quiero poder jugar una partida 1vs1 con un amigo	5	2

	HU.27	Como jugador quiero poder ver la lista de partidas pendientes	2	2
	HU.28	Como jugador quiero poder ver la lista de partidas completadas	2	2
	HU.29	Como jugador quiero poder ver los Wordles completados	2	2
	HU.12	Como jugador quiero poder seleccionar la longitud de la palabra del torneo (tipo de torneo)	2	2
	HU.13	Como jugador quiero poder unirme a una sala abierta de un torneo y jugar al torneo	4	2
Perfiles y configuración	HU.14	Como usuario quiero poder modificar mis datos personales	2	3
	HU.15	Como jugador quiero poder ver el ranking	4	3
	HU.16	Como jugador quiero poder seleccionar el filtro del ranking	2	3
	HU.17	Como jugador quiero poder ver mi perfil de jugador	2	3
	HU.19	Como usuario quiero poder cambiar el modo de visualización de la plataforma (modo predeterminado / modo oscuro)	3	5
	HU.20	Como jugador quiero poder ver mis notificaciones a través del buzón	2	2
Amigos	HU.21	Como jugador quiero poder ver mi lista de amigos	3	2
	HU.22	Como jugador quiero poder buscar a otro jugador por su nombre y mandarle una solicitud de amistad	3	2

HU.23	Como jugador quiero poder aceptar una solicitud de amistad	2	2
HU.30	Como jugador quiero poder rechazar una solicitud de amistad	2	4
HU.24	Como jugador quiero poder eliminar un jugador de mi lista de amigos	1	4
HU.25	Como jugador quiero poder ver el perfil de jugador de un amigo	2	4

Identificador: HU.1		Registro jugador
Descripción: Como jugador quiero poder registrarme		
Estimación 2	Prioridad 1	Entrega 1
Pruebas de aceptación: <ul style="list-style-type: none"> • Introducir un nickname ya existente en el sistema y comprobar que se indica un error • Introducir los campos vacíos y comprobar que se indica un error • Introducir caracteres inválidos y comprobar que se indica un error • Introducir todos los datos correctos y comprobar que se ha registrado un nuevo jugador en la base de datos 		
Observaciones:		

Identificador: HU.2		Registro administrador
Descripción: Como administrador quiero poder registrarme		
Estimación 2	Prioridad 1	Entrega 2
Pruebas de aceptación: <ul style="list-style-type: none"> • Introducir un nickname ya existente en el sistema y comprobar que se indica un error • Introducir un código de autorización incorrecto y comprobar que se indica un error. • Introducir los campos vacíos y comprobar que se indica un error • Introducir caracteres inválidos y comprobar que se indica un error • Introducir todos los datos correctos y comprobar que se ha registrado un nuevo jugador en la base de datos 		
Observaciones:		

Identificador: HU.3		Iniciar sesión
Descripción: Como usuario (jugador y administrador) quiero poder iniciar sesión		
Estimación 2	Prioridad 1	Entrega 2
Pruebas de aceptación: <ul style="list-style-type: none"> • Introducir un <i>nickname</i> no registrado en el sistema y comprobar que se indica un error • Introducir una contraseña incorrecta y comprobar que se indica un error • Introducir los campos vacíos y comprobar que se indica un error • Introducir caracteres inválidos y comprobar que se indica un error • Introducir todos los datos correctos y comprobar que el usuario ha accedido a las funcionalidades de la aplicación / se encuentra en la lista de usuarios conectados en el sistema 		
Observaciones:		

Identificador: HU.4		Cerrar sesión
Descripción: Como usuario quiero poder cerrar sesión		
Estimación 1	Prioridad 1	Entrega 2
Pruebas de aceptación: <ul style="list-style-type: none"> Comprobar que el usuario no se encuentra en la lista de usuarios conectados en el sistema 		
Observaciones:		

Identificador: HU.5		Wordle clásico
Descripción: Como jugador quiero poder jugar un Wordle clásico		
Estimación 8	Prioridad 1	Entrega 3
Pruebas de aceptación: <ul style="list-style-type: none"> Comprobar que el jugador introduce un intento de palabra y se indica la posición de cada letra (contiene, no contiene, y acierto) Comprobar que el jugador agota todos los intentos y se indica un mensaje de que ha perdido la partida Comprobar que el jugador adivina la palabra y se indica un mensaje de que ha ganado la partida 		
Observaciones:		

Identificador: HU.6		Longitud palabra Wordle clásico
Descripción: Como jugador quiero poder seleccionar la longitud de la palabra de la partida clásica		
Estimación 2	Prioridad 2	Entrega 4

Pruebas de aceptación: <ul style="list-style-type: none"> • Comprobar que el jugador selecciona la nueva longitud de palabra y se aplican los cambios en la aplicación • Comprobar que el jugador no puede escribir en el selector de longitud de palabra • Comprobar que el jugador, al escribir un intento, su longitud corresponde con el introducido en el selector de longitud
Observaciones:

Identificador: HU.7	Crear sala torneo	
Descripción: Como administrador quiero crear una sala abierta de un torneo		
Estimación 5	Prioridad 2	Entrega 5
Pruebas de aceptación: <ul style="list-style-type: none">• Introducir un valor no válido de número de participantes de la sala y comprobar que se muestra un mensaje de error• Introducir un tamaño de palabra no válido y comprobar que se muestra un mensaje de error• Introducir los campos vacíos y comprobar que se muestra un mensaje de error• Introducir todos los campos correctamente y que se crea un nuevo torneo en la lista de torneos disponibles en el sistema		
Observaciones:		

Identificador: HU.8	Tamaño torneo	
Descripción: Como administrador quiero poder seleccionar el tamaño del torneo (número de participantes)		
Estimación 2	Prioridad 2	Entrega 6
Pruebas de aceptación: <ul style="list-style-type: none">• Comprobar que el tamaño indicado coincide con el tamaño de la sala recién creada		
Observaciones:		

Identificador: HU.9	Tamaño palabra torneo	
Descripción: Como administrador quiero poder seleccionar la longitud de la palabra del torneo		
Estimación 2	Prioridad 2	Entrega 6
Pruebas de aceptación: <ul style="list-style-type: none">Comprobar que el tamaño de palabra indicado coincide con el tamaño de palabra de la sala recién creada		
Observaciones:		

Identificador: HU.10	Crear torneo fijo	
Descripción: Como administrador quiero poder crear un torneo y preseleccionar los participantes		
Estimación 3	Prioridad 2	Entrega 6
Pruebas de aceptación: <ul style="list-style-type: none">• Comprobar que se introducen usuarios registrados en el sistema• Comprobar que en número de usuarios preseleccionados no supera el tamaño máximo del torneo• Introducir un valor no válido de número de participantes de la sala y comprobar que se muestra un mensaje de error• Introducir un tamaño de palabra no válido y comprobar que se muestra un mensaje de error• Introducir los campos vacíos y comprobar que se muestra un mensaje de error• error• Introducir todos los campos correctamente y que se crea un nuevo torneo en la lista de torneos disponibles en el sistema• Comprobar que al jugador invitado le ha llegado la invitación		
Observaciones:		

Identificador: HU.11	Crear partida 1vs1	
Descripción: Como jugador quiero poder jugar una partida 1vs1 con un amigo		
Estimación 5	Prioridad 2	Entrega 7

Pruebas de aceptación: <ul style="list-style-type: none"> • Comprobar que la palabra asignada es la misma para ambos jugadores en la partida • Comprobar que, al terminar, la partida se ha introducido en el histórico de partidas • Comprobar que al jugador invitado le ha llegado la invitación de la partida
Observaciones:

Identificador: HU.12		Seleccionar longitud palabra torneo jugador
Descripción: Como jugador quiero poder seleccionar la longitud de la palabra del torneo (tipo de torneo)		
Estimación 2	Prioridad 2	Entrega 9
Pruebas de aceptación: <ul style="list-style-type: none">• Comprobar que al usuario se le muestran las salas abiertas de torneos que tienen la misma longitud de palabra que la seleccionada		
Observaciones:		

Identificador: HU.13		Unirse a sala	
Descripción: Como jugador quiero poder unirme a una sala abierta de un torneo y jugar al torneo			
Estimación 4		Prioridad 2	Entrega 9
Pruebas de aceptación: <ul style="list-style-type: none">• Comprobar que el jugador aparece en la lista de usuarios inscritos a la sala• Comprobar que el jugador puede jugar dentro del torneo• Comprobar que, si el jugador pierde, ya no forma parte del torneo y no puede seguir jugando			
Observaciones:			

Identificador: HU.14	Modificar datos personales	
Descripción: Como usuario quiero poder modificar mis datos personales		
Estimación 2	Prioridad 3	Entrega 4
Pruebas de aceptación: <ul style="list-style-type: none">• Introducir los campos vacíos y comprobar que se muestra un mensaje de error• Introducir los valores nuevos y comprobar que se han modificado en el sistema y en la interfaz del usuario		
Observaciones:		

Identificador: HU.15	Ver ranking	
Descripción: Como jugador quiero poder ver el ranking		
Estimación 4	Prioridad 3	Entrega 9
Pruebas de aceptación: <ul style="list-style-type: none">• Comprobar que al usuario se le muestra de forma ordenada por el criterio seleccionado los mejores jugadores• Comprobar que los jugadores mostrados están registrados en el sistema		
Observaciones:		

Identificador: HU.16		Cambiar filtro ranking
Descripción: Como jugador quiero poder seleccionar el filtro del ranking		
Estimación 2	Prioridad 3	Entrega 9
Pruebas de aceptación: <ul style="list-style-type: none">• Comprobar que al introducir el nuevo filtro el ranking cambia• Comprobar que el usuario no puede introducir un filtro no válido		

Identificador: HU.17		Ver perfil jugador
Descripción: Como jugador quiero poder ver mi perfil de jugador		
Estimación 2	Prioridad 3	Entrega 5
Pruebas de aceptación: <ul style="list-style-type: none"> Comprobar que al usuario se le muestran los valores almacenados en el sistema 		
Observaciones:		

Identificador: HU.19		Cambiar modo visualización
Descripción: Como usuario quiero poder cambiar el modo de visualización de la plataforma (modo predeterminado / modo oscuro)		
Estimación 3	Prioridad 5	Entrega 12
Pruebas de aceptación: <ul style="list-style-type: none"> Comprobar que al usuario se le muestra toda la aplicación en el modo seleccionado Comprobar que se ha cambiado el icono del modo de visualización al seleccionado. 		
Observaciones:		

Identificador: HU.20		Ver buzón
Descripción: Como jugador quiero poder ver mis notificaciones a través del buzón		
Estimación 2	Prioridad 2	Entrega 11
Pruebas de aceptación: <ul style="list-style-type: none"> Comprobar que al usuario se le muestra todas las notificaciones recientes Comprobar que el buzón no crece demasiado (no se eliminan antiguas notificaciones) 		
Observaciones: En el buzón se recibirán notificaciones de: solicitudes de amistad, solicitud de ingreso en un torneo, solicitudes de partidas 1vs1.		

Identificador: HU.21	Lista amigos	
Descripción: Como jugador quiero poder ver mi lista de amigos		
Estimación 3	Prioridad 2	Entrega 10
Pruebas de aceptación: <ul style="list-style-type: none">• Comprobar que, si el usuario borra un amigo, éste ya no aparece en su lista de amigos• Comprobar que, si se conecta un amigo, este aparece con algún símbolo de que está conectado en la lista de amigos• Comprobar que los amigos de la lista son los que están almacenados en el sistema		
Observaciones:		

Identificador: HU.22	Añadir amigo	
Descripción: Como jugador quiero poder buscar a otro jugador por su nombre y mandarle una solicitud de amistad		
Estimación 3	Prioridad 2	Entrega 10
Pruebas de aceptación: <ul style="list-style-type: none">• Comprobar que al amigo ha recibido la solicitud de amistad• Comprobar que el nuevo amigo aparece en su lista de amigos• Comprobar que la búsqueda introducida y los usuarios registrados están relacionados• Comprobar que si el amigo ya está añadido a la lista de amigos que no se pueda mandar una solicitud de amistad		
Observaciones:		

Identificador: HU.23	Aceptar solicitud amistad	
Descripción: Como jugador quiero poder aceptar una solicitud de amistad		
Estimación 2	Prioridad 2	Entrega 10

Pruebas de aceptación: <ul style="list-style-type: none"> Comprobar que el usuario tiene al nuevo amigo añadido a la lista de amigos
Observaciones:

Identificador: HU.24	Eliminar amigo	
Descripción: Como jugador quiero poder eliminar un jugador de mi lista de amigos		
Estimación 1	Prioridad 4	Entrega 11
<p>Pruebas de aceptación:</p> <ul style="list-style-type: none">Comprobar que el usuario eliminado ya no forma parte de la lista de amigosComprobar que a ese usuario se le puede enviar una solicitud de amistad		
Observaciones:		

Identificador: HU.25		Ver perfil amigo
Descripción: Como jugador quiero poder ver el perfil de jugador de un amigo		
Estimación 2	Prioridad 4	Entrega 11
Pruebas de aceptación: <ul style="list-style-type: none">Comprobar que al usuario se le muestran los atributos almacenados del jugador seleccionado		
Observaciones:		

Identificador: HU.26		Ver lista torneos
Descripción: Como jugador quiero poder ver la lista de torneos		
Estimación 2	Prioridad 2	Entrega 5

Pruebas de aceptación: <ul style="list-style-type: none"> Comprobar que el usuario puede ver todos los torneos disponibles
Observaciones:

Identificador: HU.27		Ver partidas pendientes
Descripción: Como jugador quiero poder ver la lista de partidas multijugador pendientes		
Estimación 2	Prioridad 2	Entrega 8
Pruebas de aceptación: <ul style="list-style-type: none">• Comprobar que el jugador puede ver todas sus partidas pendientes.• Comprobar que, al crear una nueva partida, esta aparece en la lista.• Comprobar que, al resolver la partida, esta desaparece de la lista.		
Observaciones:		

Identificador: HU.28		Ver partidas completadas
Descripción: Como jugador quiero poder ver la lista de partidas multijugador completadas		
Estimación 2	Prioridad 2	Entrega 8
Pruebas de aceptación: <ul style="list-style-type: none">• Comprobar que el jugador puede ver todas sus partidas completadas.• Comprobar que, al resolver la partida, esta aparece en la lista de completadas.		
Observaciones:		

Identificador: HU.29		Ver Wordles completados
Descripción: Como jugador quiero poder ver la lista Wordles completados		
Estimación 2	Prioridad 2	Entrega 10
Pruebas de aceptación: <ul style="list-style-type: none">• Comprobar que el jugador puede ver todos sus Wordles completados.• Comprobar que, al jugador al Wordle clásico, este aparece en la lista de Wordles completados.		

Observaciones:

Identificador: HU.30		Rechazar solicitud de amistad
Descripción: Como jugador quiero poder rechazar una solicitud de amistad		
Estimación 2	Prioridad 2	Entrega 11
Pruebas de aceptación: <ul style="list-style-type: none"> Comprobar que el jugador tiene una solicitud de amistad menos. 		
Observaciones:		

II.3. Sprints

Un aspecto que destacar de la metodología Scrum² es que el equipo de desarrollo tiene asociado una **velocidad de trabajo**. Esta magnitud representa la cantidad de trabajo realizado en un periodo de tiempo. Esto quiere decir que, si el equipo tiene una velocidad de X puntos, el equipo realizará X puntos en cada sprint.

Este valor es importante a la hora de realizar la planificación de Sprints, ya que ésta influye directamente en qué historias de usuario van a ser completadas en qué Sprint en concreto.

Normalmente, para estimar este valor, se tiene en cuenta los Puntos de Historia (PH) completados en el primer Sprint. Como estos datos no están disponibles actualmente, se realizará una planificación inicial de Sprints aproximada, con un valor de velocidad máxima de 8PH. Dicha planificación estará sujeta a cambios, dependiendo de la velocidad calculada y de la evolución de los Sprints.

II.3.1. Planificación de Sprints

La planificación de los Sprints se ha realizado según los siguientes criterios:

- Por orden de **prioridad**: las historias de usuario más prioritarias se realizarán en Sprints más tempranos.
- Por agrupamiento de similitud: las historias de usuario similares se realizarán en los mismos Sprints.

De esta forma, la planificación queda de la siguiente manera:

Nº	Objetivo	Fecha de entrega	
1	Infraestructura del proyecto. Registro de jugadores	27 de febrero del 2023	
Ident.	Título	Estimación	Prioridad
HT	Establecer la infraestructura del proyecto.	4	1
HU.1	Como jugador quiero poder registrarme	2	1

Nº	Objetivo	Fecha de entrega	
2	Registro de administrador. Inicio y cierre de sesión	13 de marzo del 2023	
Ident.	Título	Estimación	Prioridad
HU.2	Como administrador quiero poder registrarme	2	1
HU.3	Como usuario (jugador y administrador) quiero poder iniciar sesión	2	1
HU.4	Como usuario quiero poder cerrar sesión	1	1

Nº	Objetivo	Fecha de entrega	
3	Wordle clásico	27 de marzo del 2023	
Ident.	Título	Estimación	Prioridad
HU.5	Como jugador quiero poder jugar un Wordle clásico	8	1

Nº	Objetivo	Fecha de entrega	
4	Ampliación Wordle clásico. Modificación datos usuarios	10 de abril del 2023	

Ident.	Título	Estimación	Prioridad
HU.6	Como jugador quiero poder seleccionar la longitud de la palabra de la partida clásica	2	2
HU.14	Como usuario quiero poder modificar mis datos personales	2	3
HU.20	Como jugador quiero ver las notificaciones a través del buzón	2	2

Nº	Objetivo	Fecha de entrega	
5	Creación torneo	24 de abril del 2023	
Ident.	Título	Estimación	Prioridad
HU.7	Como administrador quiero crear una sala abierta de un torneo	2	2
HU.26	Como jugador quiero poder ver los torneos disponibles	2	2
HU.17	Como jugador quiero poder ver mi perfil de jugador	2	3

Nº	Objetivo	Fecha de entrega	
6	Configuración torneo	19 de junio del 2023	
Ident.	Título	Estimación	Prioridad
HU.8	Como administrador quiero poder seleccionar el tamaño del torneo (número de participantes)	2	2
HU.9	Como administrador quiero poder seleccionar la longitud de la palabra del torneo	2	2
HU.10	Como administrador quiero poder crear un torneo y preseleccionar los participantes	3	2

Nº	Objetivo	Fecha de entrega	
7	Perfil de jugador. Solicitudes de amistad. Lista de amigos.	8 de mayo del 2023	
Ident.	Título	Estimación	Prioridad
HU.21	Como jugador quiero poder ver mi lista de amigos	3	2
HU.22	Como jugador quiero poder buscar a otro jugador por su nombre y mandarle una solicitud de amistad	3	2
HU.23	Como jugador quiero poder aceptar una solicitud de amistad	2	2

Nº	Objetivo	Fecha de entrega	
8	Multijugador 1vs1. Histórico de partidas pendientes	22 de mayo del 2023	
Ident.	Título	Estimación	Prioridad
HU.11	Como jugador quiero poder jugar una partida 1vs1 con un amigo	5	2
HU.27	Como jugador quiero poder ver mis partidas multijugador pendientes	2	2

Nº	Objetivo	Fecha de entrega	
9	Histórico de partidas completadas. Inscripción al torneo	05 de junio del 2023	
Ident.	Título	Estimación	Prioridad
HU.28	Como jugador quiero poder ver mis partidas multijugador completadas	2	2
HU.12	Como jugador quiero poder seleccionar la longitud de la palabra del torneo (tipo de torneo)	2	2

HU.13	Como jugador quiero poder unirme a una sala abierta de un torneo y jugar al torneo	4	2
--------------	--	---	---

Nº	Objetivo	Fecha de entrega	
10	Visualización del ranking. Wordles completados	19 de junio del 2023	
Ident.	Título	Estimación	Prioridad
HU.29	Como jugador quiero poder ver la lista de los Wordles completados	2	2
HU.15	Como jugador quiero poder ver el ranking	4	3
HU.16	Como jugador quiero poder seleccionar el filtro del ranking	2	3

Nº	Objetivo	Fecha de entrega	
11	Eliminación de amigos. Perfiles de amigos.	02 de julio del 2023	
Ident.	Título	Estimación	Prioridad
HU.24	Como jugador quiero poder eliminar un jugador de mi lista de amigos	1	4
HU.25	Como jugador quiero poder ver el perfil de jugador de un amigo	2	4
HU.30	Como jugador quiero poder aceptar una solicitud de amistad	2	4

Nº	Objetivo	Fecha de entrega	
12	Modo oscuro. Cambio de idioma	16 de julio del 2023	
Ident.	Título	Estimación	Prioridad
HU.19	Como usuario quiero poder cambiar el modo de visualización de la plataforma (modo predeterminado / modo oscuro)	3	5

Esta planificación inicial muestra que el Sprint número 12 se quedaría fuera de plazo de la entrega del producto, siempre y cuando la velocidad del equipo no varíe. Si la velocidad de equipo se ve incrementada después de varios Sprints completados, éste último se incorporará al producto final.

II.3.2. División en tareas

Una vez definidas las Historias de Usuario, éstas se pueden dividir en tareas. Una tarea no es parte del resultado del proyecto, sino que es un medio para producir un resultado.

En este proyecto se seguirá un patrón *Modelo Vista Controlador*, y por ello, se pueden distinguir dos tipos de tareas:

- Tareas de **frontend**: son las tareas relacionadas con el diseño, implementación y desarrollo de las vistas de usuario e interfaces, es decir, con lo que el usuario podrá interactuar directamente.
- Tareas de **backend**: son las tareas relacionadas con la base de datos y el controlador. Son funcionalidades que el usuario no va a interactuar directamente con ellas, pero son necesarias para que el sistema, en su cómputo total, funcione correctamente.

Identificador: HU.1		Registro jugador
Identificador	Título de la tarea	
Tarea 1-1	Definir el modelo de jugador en la BBDD.	
Tarea 1-2	Implementación la inserción automática del jugador en la BBDD.	
Tarea 1-3	Definir el modelo de usuario en la BBDD.	
Tarea 1-4	Implementación la inserción automática del usuario en la BBDD.	
Tarea 1-5	Definir el controlador asociado al registro del jugador.	
Tarea 1-6	Diseño e implementación de la IU asociada a la creación del jugador.	
Observaciones:		

Identificador: HU.2	Registro del administrador										
<table> <tr> <th>Identificador</th><th>Título de la tarea</th></tr> <tr> <td>Tarea 2-1</td><td>Definir el modelo del administrador en la BBDD.</td></tr> <tr> <td>Tarea 2-2</td><td>Implementación la inserción automática del administrador en la BBDD.</td></tr> <tr> <td>Tarea 2-3</td><td>Definir el controlador asociado al registro del administrador.</td></tr> <tr> <td>Tarea 2-4</td><td>Diseño e implementación de la IU asociada a la creación del administrador.</td></tr> </table>		Identificador	Título de la tarea	Tarea 2-1	Definir el modelo del administrador en la BBDD.	Tarea 2-2	Implementación la inserción automática del administrador en la BBDD.	Tarea 2-3	Definir el controlador asociado al registro del administrador.	Tarea 2-4	Diseño e implementación de la IU asociada a la creación del administrador.
Identificador	Título de la tarea										
Tarea 2-1	Definir el modelo del administrador en la BBDD.										
Tarea 2-2	Implementación la inserción automática del administrador en la BBDD.										
Tarea 2-3	Definir el controlador asociado al registro del administrador.										
Tarea 2-4	Diseño e implementación de la IU asociada a la creación del administrador.										
Observaciones: El modelo de usuarios ya está creado.											

Identificador: HU.3	Iniciar sesión						
<table> <tr> <th>Identificador</th><th>Título de la tarea</th></tr> <tr> <td>Tarea 3-1</td><td>Definir el controlador asociado al inicio de sesión.</td></tr> <tr> <td>Tarea 3-2</td><td>Diseño e implementación de la IU asociada al inicio de sesión.</td></tr> </table>		Identificador	Título de la tarea	Tarea 3-1	Definir el controlador asociado al inicio de sesión.	Tarea 3-2	Diseño e implementación de la IU asociada al inicio de sesión.
Identificador	Título de la tarea						
Tarea 3-1	Definir el controlador asociado al inicio de sesión.						
Tarea 3-2	Diseño e implementación de la IU asociada al inicio de sesión.						
Observaciones: El modelo de usuario está creado.							

Identificador: HU.4	Cerrar sesión						
<table> <tr> <th>Identificador</th><th>Título de la tarea</th></tr> <tr> <td>Tarea 4-1</td><td>Diseño de la IU asociada al botón de cierre de sesión.</td></tr> <tr> <td>Tarea 4-2</td><td>Definir el controlador asociado al cierre de sesión.</td></tr> </table>		Identificador	Título de la tarea	Tarea 4-1	Diseño de la IU asociada al botón de cierre de sesión.	Tarea 4-2	Definir el controlador asociado al cierre de sesión.
Identificador	Título de la tarea						
Tarea 4-1	Diseño de la IU asociada al botón de cierre de sesión.						
Tarea 4-2	Definir el controlador asociado al cierre de sesión.						
Observaciones: El modelo de usuario está creado.							

Identificador: HU.5	Wordle clásico								
<table> <tr> <th>Identificador</th><th>Título de la tarea</th></tr> <tr> <td>Tarea 5-1</td><td>Implementación de la lógica del Wordle clásico.</td></tr> <tr> <td>Tarea 5-2</td><td>Diseño e implementación de la IU asociada al Wordle clásico.</td></tr> <tr> <td>Tarea 5-3</td><td>Definir el controlador asociado a la partida.</td></tr> </table>		Identificador	Título de la tarea	Tarea 5-1	Implementación de la lógica del Wordle clásico.	Tarea 5-2	Diseño e implementación de la IU asociada al Wordle clásico.	Tarea 5-3	Definir el controlador asociado a la partida.
Identificador	Título de la tarea								
Tarea 5-1	Implementación de la lógica del Wordle clásico.								
Tarea 5-2	Diseño e implementación de la IU asociada al Wordle clásico.								
Tarea 5-3	Definir el controlador asociado a la partida.								
Observaciones:									

Identificador: HU.6	Longitud palabra Wordle clásico				
<table> <tr> <th>Identificador</th><th>Título de la tarea</th></tr> <tr> <td>Tarea 6-1</td><td>Definir el controlador asociado al cambio de longitud de la palabra.</td></tr> </table>		Identificador	Título de la tarea	Tarea 6-1	Definir el controlador asociado al cambio de longitud de la palabra.
Identificador	Título de la tarea				
Tarea 6-1	Definir el controlador asociado al cambio de longitud de la palabra.				

Tarea 6-2	Diseño e implementación de la IU asociada a la selección de la longitud de la palabra.
Observaciones: El usuario, al querer jugar a un Wordle clásico, tendrá una elección previa que será el tamaño de la palabra del Wordle.	

Identificador: HU.7	Crear sala torneo
Identificador	Título de la tarea
Tarea 7-1	Definir el modelo de la sala torneo en la BBDD.
Tarea 7-2	Implementación la inserción automática de la sala del torneo en la BBDD.
Tarea 7-3	Definir el controlador asociado a la creación de la sala del torneo.
Tarea 7-4	Diseño e implementación de la IU asociada a la creación de la sala del torneo.
Observaciones: Vista solo apta para administradores.	

Identificador: HU.8	Tamaño torneo
Identificador	Título de la tarea
Tarea 8-1	Definir el controlador asociado al cambio de longitud de la palabra.
Tarea 8-2	Diseño e implementación de la IU asociada a la selección de la longitud de la palabra.

Observaciones:

Vista apta solamente para administradores.

Identificador: HU.9	Tamaño palabra torneo
Identificador	Título de la tarea
Tarea 9-1	Definir el controlador asociado al cambio de tamaño de la palabra del torneo.
Tarea 9-2	Diseño e implementación de la IU asociada a la selección de la longitud de la palabra del torneo.
Observaciones: Vista apta solo para administradores.	

Identificador: HU.10	Crear torneo fijo
Identificador	Título de la tarea
Tarea 10-1	Definir el controlador asociado a la creación del torneo.
Tarea 10-2	Enviar notificación correspondiente a los jugadores seleccionados.
Tarea 10-3	Diseño e implementación de la IU asociada a la creación del torneo fijo.
Observaciones: <ul style="list-style-type: none">- El modelo del torneo ya está creado.- Vista apta solo para administradores.	

Identificador: HU.11		Crear partida 1vs1
Identificador	Título de la tarea	
Tarea 11-1	Definir el modelo de las partidas multijugador.	
Tarea 11-2	Definir el controlador asociado a la creación de la partida 1vs1.	
Tarea 11-3	Diseño e implementación de la IU asociada a la partida 1vs1.	
Tarea 11-4	Enviar notificación correspondiente al jugador.	
Observaciones: <ul style="list-style-type: none">- Se completará cuando se implemente la lista de amigos.- El modelo del jugador ya está creado.		

Identificador: HU.12		Seleccionar longitud palabra torneo jugador
Identificador	Título de la tarea	
Tarea 12-1	Diseño e implementación de la IU asociada a la selección de la longitud de palabra en el torneo.	
Tarea 12-2	Implementación del controlador asociado a la selección de la longitud de palabra en el torneo.	
Observaciones: El usuario, al querer ingresar a un torneo, tendrá una elección previa que será el tamaño de la palabra de dicho torneo (tipo).		

Identificador: HU.13		Jugar un torneo
Identificador	Título de la tarea	
Tarea 13-1	Definir el controlador asociado a la añadir al usuario a la sala seleccionada.	
Tarea 13-2	Diseño e implementación de la IU asociada a la unirse a la sala.	

Tarea 13-3	Implementación de la creación automática de rondas y partidas.
Tarea 13-4	Diseño e implementación de la IU asociada a las rondas del torneo.
Tarea 13-5	Diseño e implementación de la IU asociada la partida del torneo.
Observaciones: <ul style="list-style-type: none"> - El modelo de las salas ya está creado. - El modelo de los jugadores ya está creado. 	

Identificador: HU.14	Modificar datos personales
Identificador	Título de la tarea
Tarea 14-1	Definir el controlador asociado a la modificación de los datos del usuario.
Tarea 14-2	Diseño e implementación de la IU asociada a la ficha del usuario.
Observaciones: El modelo de los usuarios ya está creado.	

Identificador: HU.15	Ver ranking
Identificador	Título de la tarea
Tarea 15-1	Definir el controlador asociado a obtener los usuarios del ranking.
Tarea 15-2	Diseño e implementación de la IU asociada al ranking.

Observaciones:

El modelo de los jugadores ya está creado.

Identificador: HU.16	Cambiar filtro ranking
Identificador	Título de la tarea
Tarea 16-1	Definir el controlador asociado a obtener los usuarios del ranking según el filtro escogido.
Tarea 16-2	Definir la IU asociada al filtro del ranking.
Observaciones:	

Identificador: HU.17	Ver perfil jugador
Identificador	Título de la tarea
Tarea 17-1	Definir el controlador asociado a obtener la información del jugador.
Tarea 17-2	Diseño e implementación de la IU asociada al perfil del jugador.
Observaciones:	
El modelo de los jugadores ya está creado.	

Identificador: HU.19	Cambiar modo visualización
Identificador	Título de la tarea

Tarea 19-1	Definir el controlador asociado al cambio de visualización.
Tarea 19-2	Diseño e implementación de la IU asociada al cambio de visualización (botón).
Observaciones:	

Identificador: HU.20	Ver buzón										
<table> <tr> <th>Identificador</th><th>Título de la tarea</th></tr> <tr> <td>Tarea 20-1</td><td>Definir el modelo de las Notificaciones en la BBDD.</td></tr> <tr> <td>Tarea 20-2</td><td>Implementación la inserción automática de las notificaciones en la BBDD.</td></tr> <tr> <td>Tarea 20-3</td><td>Definir el controlador asociado al buzón.</td></tr> <tr> <td>Tarea 20-4</td><td>Diseño e implementación de la IU asociada al buzón.</td></tr> </table>		Identificador	Título de la tarea	Tarea 20-1	Definir el modelo de las Notificaciones en la BBDD.	Tarea 20-2	Implementación la inserción automática de las notificaciones en la BBDD.	Tarea 20-3	Definir el controlador asociado al buzón.	Tarea 20-4	Diseño e implementación de la IU asociada al buzón.
Identificador	Título de la tarea										
Tarea 20-1	Definir el modelo de las Notificaciones en la BBDD.										
Tarea 20-2	Implementación la inserción automática de las notificaciones en la BBDD.										
Tarea 20-3	Definir el controlador asociado al buzón.										
Tarea 20-4	Diseño e implementación de la IU asociada al buzón.										
Observaciones:											

Identificador: HU.21	Lista amigos								
<table> <tr> <th>Identificador</th><th>Título de la tarea</th></tr> <tr> <td>Tarea 21-1</td><td>Definir el modelo de la lista de amigos.</td></tr> <tr> <td>Tarea 21-2</td><td>Definir el controlador asociado a obtener la lista de amigos.</td></tr> <tr> <td>Tarea 21-3</td><td>Diseño e implementación de la IU asociada a la lista de amigos.</td></tr> </table>		Identificador	Título de la tarea	Tarea 21-1	Definir el modelo de la lista de amigos.	Tarea 21-2	Definir el controlador asociado a obtener la lista de amigos.	Tarea 21-3	Diseño e implementación de la IU asociada a la lista de amigos.
Identificador	Título de la tarea								
Tarea 21-1	Definir el modelo de la lista de amigos.								
Tarea 21-2	Definir el controlador asociado a obtener la lista de amigos.								
Tarea 21-3	Diseño e implementación de la IU asociada a la lista de amigos.								

Observaciones:

El modelo de los jugadores ya está creado.

Identificador: HU.22

Añadir amigo

Identificador	Título de la tarea
Tarea 22-1	Definir el modelo de la lista de peticiones de amistad.
Tarea 22-2	Definir el controlador asociado a añadir un amigo.
Tarea 22-3	Diseño e implementación de la IU asociada a añadir a un amigo (botón en lista de amigos).
Tarea 22-4	Enviar notificación correspondiente al jugador.

Observaciones:

El modelo de los jugadores ya está creado.

Identificador: HU.23

Aceptar solicitud amistad

Identificador	Título de la tarea
Tarea 23-1	Definir el controlador asociado a aceptar a un amigo.
Tarea 23-2	Diseño e implementación de la IU asociada a aceptar a un amigo (botón).
Tarea 23-3	Enviar notificación correspondiente al jugador.

Observaciones:

El modelo de los jugadores ya está creado.

Identificador: HU.24	Eliminar amigo
Identificador	Título de la tarea
Tarea 24-1	Definir el controlador asociado a eliminar a un amigo.
Tarea 24-2	Diseño e implementación de la IU asociada a eliminar a un amigo (botón).
Observaciones: El modelo de los jugadores ya está creado.	

Identificador: HU.25	Ver perfil amigo
Identificador	Título de la tarea
Tarea 25-1	Definir el controlador asociado a ver el perfil de un amigo.
Tarea 25-2	Diseño e implementación de la IU asociada a ver el perfil de un amigo (botón).
Observaciones: <ul style="list-style-type: none"> - El modelo de los jugadores ya está creado. - La IU asociada a ver el perfil de un jugador ya está creada. 	

Identificador: HU.26	Lista de torneos
Identificador	Título de la tarea
Tarea 26-1	Definir el controlador asociado a la vista de los torneos.
Tarea 26-2	Diseño e implementación de la IU asociada a la vista de los torneos.
Observaciones: <ul style="list-style-type: none"> - El modelo de los torneos ya está creado 	

Identificador: HU.27	Ver partidas pendientes
Identificador	Título de la tarea
Tarea 27-1	Definir el controlador asociado a ver las partidas pendientes.
Tarea 27-2	Diseño e implementación de la IU asociada a ver las partidas pendientes .
Observaciones: - El modelo de las partidas ya está creado.	

Identificador: HU.28	Ver partidas completadas
Identificador	Título de la tarea
Tarea 28-1	Definir el controlador asociado a ver las partidas completadas.
Tarea 28-2	Diseño e implementación de la IU asociada a ver las partidas completadas.
Observaciones: - El modelo de las partidas ya está creado.	

Identificador: HU.29	Ver Wordles completados
Identificador	Título de la tarea
Tarea 29-1	Definir el controlador asociado a ver los Wordles completados.
Tarea 29-2	Diseño e implementación de la IU asociada a ver los Wordles completados.
Observaciones: - El modelo de las partidas ya está creado.	

Identificador: HU.29	Ver Wordles completados
Identificador	Título de la tarea
Tarea 29-1	Definir el controlador asociado a ver los Wordles completados.
Tarea 29-2	Diseño e implementación de la IU asociada a ver los Wordles completados.
Observaciones: <ul style="list-style-type: none"> - El modelo de las partidas ya está creado. 	

Identificador: HU.30	Rechazar solicitud amistad
Identificador	Título de la tarea
Tarea 30-1	Definir el controlador asociado a rechazar a un amigo.
Tarea 30-2	Diseño e implementación de la IU asociada a rechazar la solicitud.
Observaciones: <ul style="list-style-type: none"> - El modelo de los jugadores ya está creado. - El modelo de las solicitudes de amistad ya está creado. 	

II.4. Presupuesto

El presupuesto del proyecto se basará en el cálculo de las horas invertidas en el desarrollo y análisis, considerando el precio por hora de un desarrollador de software en España, así como los costos asociados al uso de un ordenador personal y la conexión a Internet. Cabe mencionar que se buscarán tecnologías accesibles y gratuitas, lo que ayudará a minimizar los gastos relacionados con software y licencias.

Desarrollo

Para el desarrollo del proyecto, que representa aproximadamente 180 horas de las 300 horas asociadas, se aplicará un precio por hora correspondiente a un desarrollador de software en España. Supongamos un precio de 40 euros por hora para estas tareas.

Presupuesto de desarrollo: $180 \text{ horas} \times 40 \text{ euros/hora} = 7,200 \text{ euros}$

Costos del equipo y conexión a Internet:

En cuanto al dispositivo utilizado y que aloja el proyecto, se ha utilizado mi ordenador personal, con un valor de 700 euros.

En cuanto a la conexión a Internet, consideremos una tarifa mensual de 30 euros durante el período de desarrollo de 6 meses.

Presupuesto de conexión a Internet: $6 \text{ meses} \times 30 \text{ euros/mes} = 180 \text{ euros}$

Total del presupuesto:

Sumando los costos de desarrollo, equipo y conexión a Internet, obtenemos el presupuesto total del proyecto:

Presupuesto total = Presupuesto de desarrollo + Presupuesto de equipo + Presupuesto de conexión a Internet

Presupuesto total = $7,200 \text{ euros} + 700 \text{ euros} + 180 \text{ euros} = 8,080 \text{ euros}$

CAPÍTULO III. DESARROLLO

III.1. Elección de las tecnologías

El desarrollo de aplicaciones web multiplataforma tiene una importancia destacable en la actualidad por las siguientes razones:

1. Mayor alcance de audiencia: Al desarrollar una aplicación web multiplataforma, se puede llegar a una audiencia más amplia, ya que se puede ejecutar en diferentes sistemas operativos y dispositivos, como computadoras de escritorio, laptops, tabletas y teléfonos inteligentes.
2. Reducción de costos: Al desarrollar una aplicación web multiplataforma, se puede reducir el costo de desarrollo y mantenimiento, ya que se puede utilizar un código base común en todas las plataformas en lugar de tener que desarrollar aplicaciones específicas para cada sistema operativo o dispositivo.
3. Mayor flexibilidad: Al desarrollar este tipo de aplicaciones, permite que los usuarios accedan a la aplicación desde cualquier lugar y en cualquier momento, lo que aumenta la flexibilidad y la comodidad.

Para lograr esto, los ingenieros informáticos deben tener una comprensión profunda de los diferentes sistemas operativos y dispositivos, así como de los lenguajes de programación y herramientas necesarias para crear aplicaciones web multiplataforma. Además, también deben considerar las mejores prácticas de seguridad y privacidad para garantizar que la aplicación sea segura y proteja los datos del usuario en todas las plataformas y dispositivos.

III.1.1. Frontend

Una vez planteadas las HUs, su división en tareas, y la planificación de los Sprints, es necesario realizar un análisis de las tecnologías y herramientas candidatas para realizar este proyecto.

Antes de profundizar más en la cuestión, me gustaría aclarar que uno de los grandes objetivos de este proyecto es el aprendizaje y la adquisición de nuevo conocimiento, al igual que todo mi transcurso por la carrera, por lo que le daré mayor peso a aquellas herramientas con las que no tengo experiencia y/o demasiado conocimiento. A priori, sería más sencillo utilizar herramientas ya conocidas, pero estoy más interesado en ampliar mi perfil académico e invertir cierto tiempo en aprender ámbitos nuevos.

Como ya se ha mencionado varias veces, Wordle+ consiste en realizar un proyecto *fullstack*, es decir, tiene componentes tanto *backend* como *frontend*. Además, uno de los principales objetivos del proyecto es conseguir una aplicación web multiplataforma, por lo que tecnologías dedicadas exclusivamente para desarrollo de aplicaciones para móviles (como Android Studio⁹, Flutter¹⁰, etc) quedarán descartadas. Con esto, el dominio queda reducido a tecnologías web, por lo que las alternativas principales son:

- **Pila LAMP¹¹**: consiste en una infraestructura que utiliza Linux¹², Apache¹³, MySQL/MariaDB¹⁴ y PHP¹⁵, también utilizando JavaScript¹⁶ como lenguaje de *scripting*. Es una infraestructura ya ampliamente utilizada en otros proyectos de la carrera, por lo que quedan descartados. Además, puede resultar complicado diseñar completamente el proyecto de forma adaptativa o *responsive*.
- **Pila MERN/MEAN¹⁷**: similar a la pila LAMP⁷, pero o se utiliza MongoDB¹⁸ como gestor de base de datos; Express¹⁹ como gestor de peticiones HTTP; React²⁰ o Angular²¹ como herramienta principal para el *frontend*; y Node.js²² como plataforma de *backend*. El principal lenguaje a utilizar es JavaScript¹². Esta alternativa resulta interesante y se presenta como una renovación de la pila LAMP⁷. Sin embargo, personalmente ya he utilizado esta infraestructura (MERN) en un proyecto de la asignatura “Dirección y Gestión de Proyectos”. En enlace al repositorio público es: <https://github.com/davidcr01/Class4All>. Además, presenta el mismo problema del diseño *responsive* que presentaba la pila LAMP⁷.

Con lo anterior expuesto, se podría cambiar el foco a frameworks que permitan el desarrollo de aplicaciones multiplataforma o híbridas. A pesar de que hay una extensa lista de alternativas, se comentarán las principales de ellas:

- **Flutter⁶**: si bien es cierto que ha sido mencionado anteriormente, Flutter es uno de los sistemas más utilizados para el desarrollo de este tipo de aplicaciones. Sin embargo, se suele utilizar en mayor medida para el desarrollo de aplicaciones móviles, además de utilizar el lenguaje de programación Dart²³, por lo que se aleja del uso de tecnologías web, que es uno de los objetivos de este proyecto.
- **React Native²⁴**: se presenta como otra alternativa interesante, en donde el lenguaje principal es JavaScript, con posibilidad de escribir módulos en Objective-C²⁵, Swift²⁶ o Java²⁷. Sin embargo, como ya he comentado anteriormente, he realizado un proyecto utilizando React²⁰, por lo que el aprendizaje que me aportaría esta alternativa se vería reducido.
- **Ionic Framework²⁸**: al igual que las demás, permite crear aplicaciones híbridas, pero en este caso utilizando HTML, CSS y JavaScript¹², con otra notación distinta. Este framework permite un desarrollo ágil para diseñar aplicaciones híbridas, pudiendo utilizar elementos web ya construidos. Cabe destacar que es uno de los framework más utilizado actualmente, además de tener una gran comunidad y documentación.

Finalmente he escogido de **Ionic Framework**²⁸, por lo comentado anteriormente. Sigue la línea de las tecnologías web, que es uno de los objetivos de este proyecto, permitiendo desarrollar Wordle+ tanto en web como en móvil de forma **simultánea**. Además, esta alternativa permite utilizar tres de los grandes lenguajes de programación *frontend* actual a elegir, es decir, solamente se puede utilizar una de ellas: React²⁰, Angular²¹ y Vue²⁹. En mi caso, descarto la primera por haberla utilizado en otros proyectos, y respecto a las dos restantes, ambas son similares, pero **Angular**¹⁶ tiene una comunidad y tasa de mercado mucho más significativa.

Por tanto, y a modo de resumen, se utilizará el framework Ionic²⁸ utilizando el lenguaje de programación Angular²¹.

III.1.2. Backend

El groso de las tecnologías comentadas anteriormente son herramientas de desarrollo de *frontend*. Respecto al *backend*, podemos categorizar las alternativas siguiendo diferentes criterios:

- Por **cómo** se gestiona la información: las principales alternativas en este campo son modelos **relacionales** y modelos **no relacionales**. Personalmente he utilizado ambos exhaustivamente, con herramientas como MySQL¹⁰ y MongoDB¹⁴. Sin embargo, tengo cierto interés en utilizar **PostgreSQL**³⁰, uno de los sistemas de gestión de bases de datos relacional más utilizado actualmente. En este caso, tengo preferencia en utilizar modelos relacionales, ya que gran parte de la información que va a alojar la base de datos tiene bastante conexión mediante claves externas, y tendría poco sentido utilizar modelos no relacionales.
- Por **dónde** se ubica la información: en este caso, las principales alternativas son utilizar servicios en la **nube** o alojar la base de datos de forma **local**. La nube es interesante ya que los datos están almacenados en internet, y hay poca probabilidad de pérdida. Sin embargo, se requiere de un proveedor externo, de un ancho de banda considerable para obtener la información, y evidentemente de conexión a internet. Por otro lado, la alternativa local no presenta las desventajas descritas anteriormente, pero la información solamente estaría alojada en el sistema local. Finalmente, he preferido escoger la alternativa **local**, aunque para paliar sus desventajas me gustaría realizar un despliegue del backend en un contenedor Docker³⁴, punto que se tratará más adelante.

Este proyecto presenta una arquitectura del tipo Modelo-Vista-Controlador. Ya se han mencionado los aspectos del “Modelo” y de la “Vista”. Respecto al controlador, se propone desarrollar una API REST³¹, que permite la

estandarización de intercambio de datos entre los servicios web, abstrayéndose del lenguaje de programación y de los sistemas operativos.

Utilizar una API REST³¹ tiene las siguientes ventajas:

- Escalabilidad: se optimizan las interacciones entre el cliente y el servidor.
- Flexibilidad: existe un desacoplo entre los componentes de manera que éstos pueden evolucionar de forma independiente
- Independencia: no importa la tecnología utilizada, no afecta al diseño de la API.

En cuanto a tecnologías, existen dos alternativas ampliamente conocidas:

- **Node.js**¹⁷: junto con la herramienta Express¹⁹, es uno de los métodos más utilizados para crear APIs. Sin embargo, esta alternativa ya ha sido utilizada en el proyecto anteriormente mencionado.
- **Python**³²: uno de los lenguajes multiparadigma más utilizados en el panorama actual, permite construir APIs de forma sencilla mediante el framework **Django REST framework**³³. Esta opción me parece interesante para incorporar al proyecto una potente tecnología como es Python²⁶, y hacer uso de otro de los frameworks más utilizados actualmente, como es el citado anteriormente.

III.1.3. Despliegue

Actualmente, la tecnología está evolucionando continuamente y a una gran velocidad. Ya se han mencionado anteriormente aspectos como la integración y distribución continuas (CICD⁸) que aportan nuevo software rápidamente, con actualizaciones muy frecuentes.

Ante este panorama, no es recomendable hacer un despliegue en un sistema en concreto, utilizando un sistema operativo y un hardware específico. Esta metodología haría el proyecto muy rígido, permitiendo no poder migrarlo en caso de que fuera necesario. Ante esto, surgen varios proyectos como **Docker**³⁴ que permiten realizar un despliegue de aplicaciones dentro de contenedores software. Esto, a su vez, abstrae y automatiza la virtualización de aplicaciones en distintos sistemas operativos.

Por ello, lo ideal sería hacer el despliegue de todo el proyecto utilizando la herramienta *docker-compose* que permite orquestar varios contenedores y realizar una comunicación interna entre ellos. Si esto no es posible, se pretende como mínimo desplegar el *backend* en contenedores, para así abstraer toda la información e instalación de esta parte del proyecto del sistema sobre el que se construya. Cabe destacar que *docker-compose* no soporta *hot-reload* con *ionic*²⁸, por lo que se servirá el servidor en local, y cuando se apliquen los cambios se reconstruirá la imagen del *frontend*.

III.1.4. Control de versiones y seguimiento del Sprint

GitHub³⁵ es uno de los sitios web en donde más desarrolladores alojan sus proyectos software. Gracias a su integración con la herramienta *Git* para el control de cambios, y sus características como las *Issues*, *Pull Request*, *Milestones*, la convierte en una herramienta idónea para este tipo de proyectos. Por tanto, el proyecto se alojará en un repositorio de GitHub³⁵, haciendo uso de *Git* para el control de cambios.

Respecto al seguimiento de los Sprints, existen varias conocidas alternativas como Trello o Jira, en donde plasmar todo el Product Backlog, planificación de los Sprints y su seguimiento. Sin embargo, esta tarea se puede también realizar el GitHub³⁵ de formas distintas.

En el proceso de desarrollo de la plataforma, se ha adoptado una metodología que involucra la documentación detallada y el control de versiones utilizando GitHub³⁵. Esta elección se basa en el deseo de proporcionar una visión completa y didáctica del progreso del desarrollo, permitiendo que otros usuarios puedan aprender, examinar distintas perspectivas y obtener soluciones a problemas comunes.

GitHub³⁵ se ha convertido en una herramienta fundamental para mantener un registro exhaustivo de todo el proceso de desarrollo. Cada historia de usuario se representa mediante una *issue*, la cual puede incluir varias tareas específicas relacionadas con esa historia. Además, las *issues* también se utilizan para reportar errores o bugs encontrados en la aplicación, permitiendo un seguimiento y solución eficiente de los mismos.

Con el fin de facilitar la planificación y el seguimiento del progreso, las historias de usuario se agrupan en milestones, que representan de manera aproximada cada *sprint* del proyecto. Estos milestones ayudan a organizar y priorizar las tareas de desarrollo.

En cuanto a las ramas utilizadas en el repositorio, se tiene la rama *main* que contiene una versión final y estable del producto. Las ramas de *desarrollo* se crean a partir de *main* y se enumeran, como por ejemplo 1.2, con *major* y *minor*. Cada rama enumerada se asocia a una *issue* específica, permitiendo un enfoque más focalizado en cada funcionalidad o problema a resolver. También existirá una rama *documentation* en donde se alojará el progreso y cambios de este documento.

Cuando el desarrollo de una funcionalidad o solución de *bug* ha finalizado, la rama de desarrollo se fusiona con su rama numerada, y cuando el Sprint ha finalizado, la rama numerada se fusiona con *main* si el incremento es suficiente para generar una nueva *release*. Este enfoque de liberación de versiones permite mantener un flujo constante de mejoras y actualizaciones del producto, brindando a los usuarios una experiencia más completa y actualizada.

Un ejemplo de esta metodología podría ser la siguiente:

1. Se crea una *issue* que reporte el desarrollo o la historia de usuario a implementar. Se describe brevemente su objetivo y se listan las tareas relacionadas, asociadas con la división en tareas. Nótese que se añaden las *labels* correspondientes, que dan más información acerca del desarrollo; se añade al proyecto Wordle+, donde se muestra el estado de la *issue* y su sprint asociado; el *milestone* asociado y la rama de desarrollo relacionada (Figura 1):

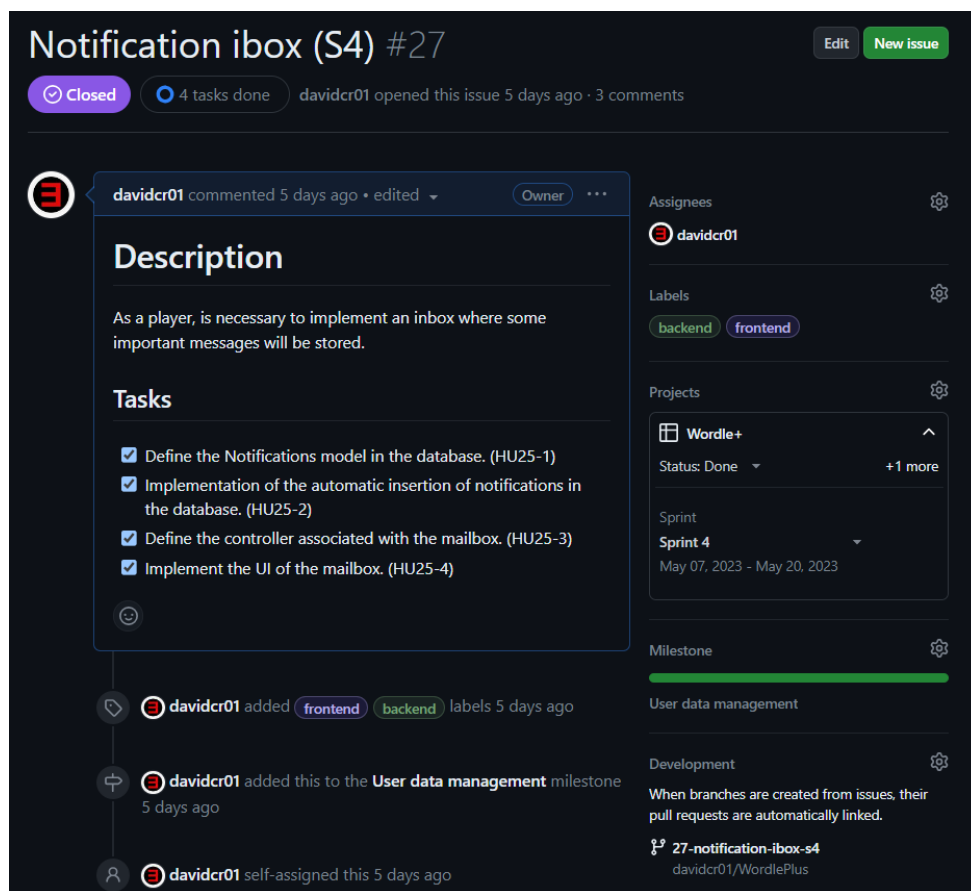


Figura 1. Información de las *issues*.

2. A lo largo del desarrollo, se van añadiendo nuevos comentarios en la *issues*, denominados reportes, en donde se explica el progreso y las soluciones alcanzadas para implementar la funcionalidad con cierto nivel de profundidad, además de mostrar resultados o pruebas (Figura 2):

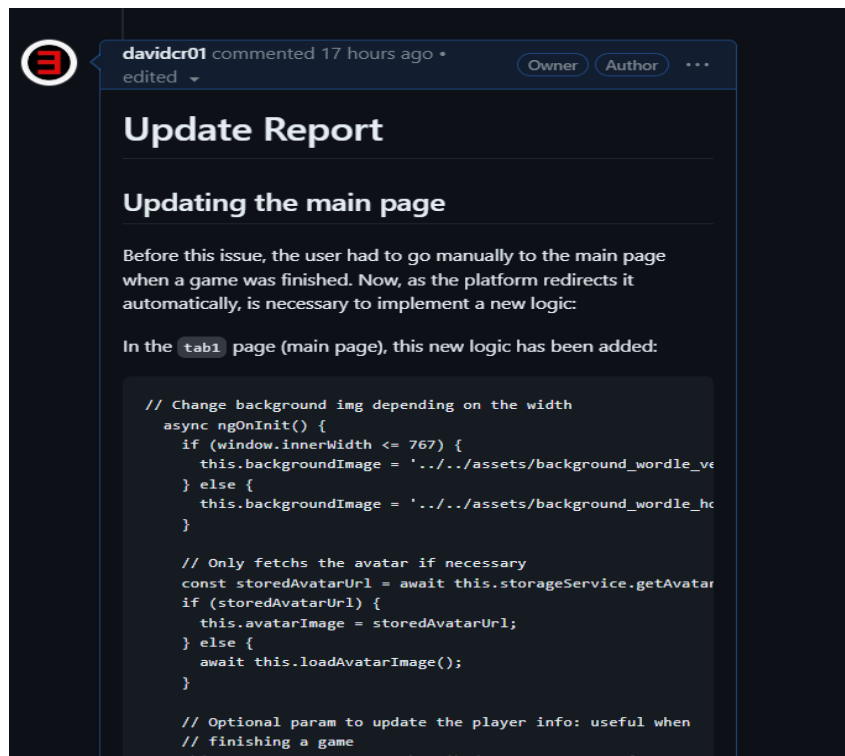


Figura 2. Reportes.

3. Cuando se finaliza el desarrollo y se han realizado las correspondientes pruebas de calidad, se hace un *Pull Request* de la rama de desarrollo a la rama enumerada (Figura 3):

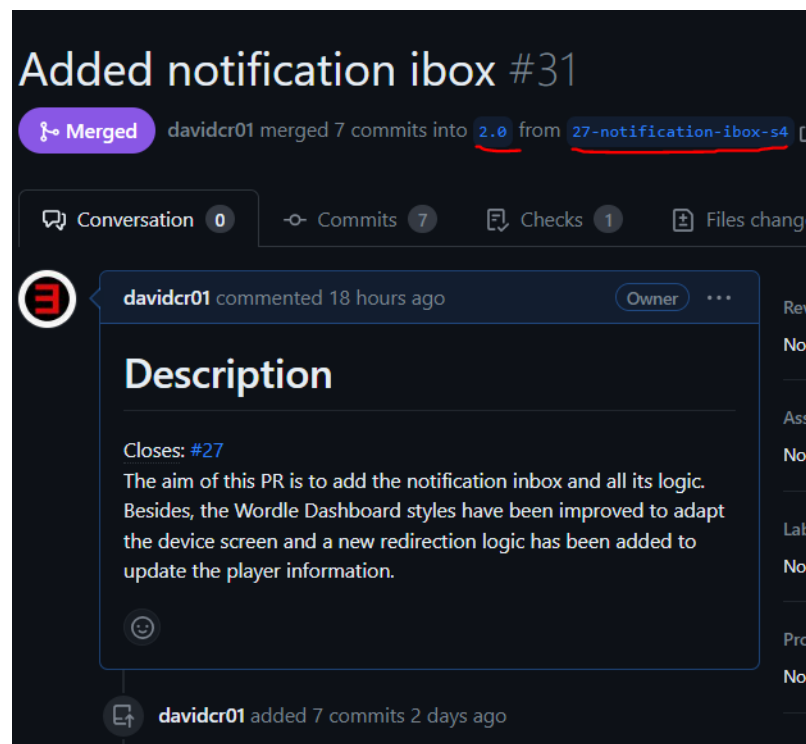


Figura 3. *Pull Request* al finalizar desarrollo.

4. Cuando todos los desarrollos del Sprint se han finalizado, la rama enumerada será incorporada a la rama principal, añadiendo todas las nuevas funcionalidades (Figura 4). Estos *PR* son los asociados a nuevas *releases*:

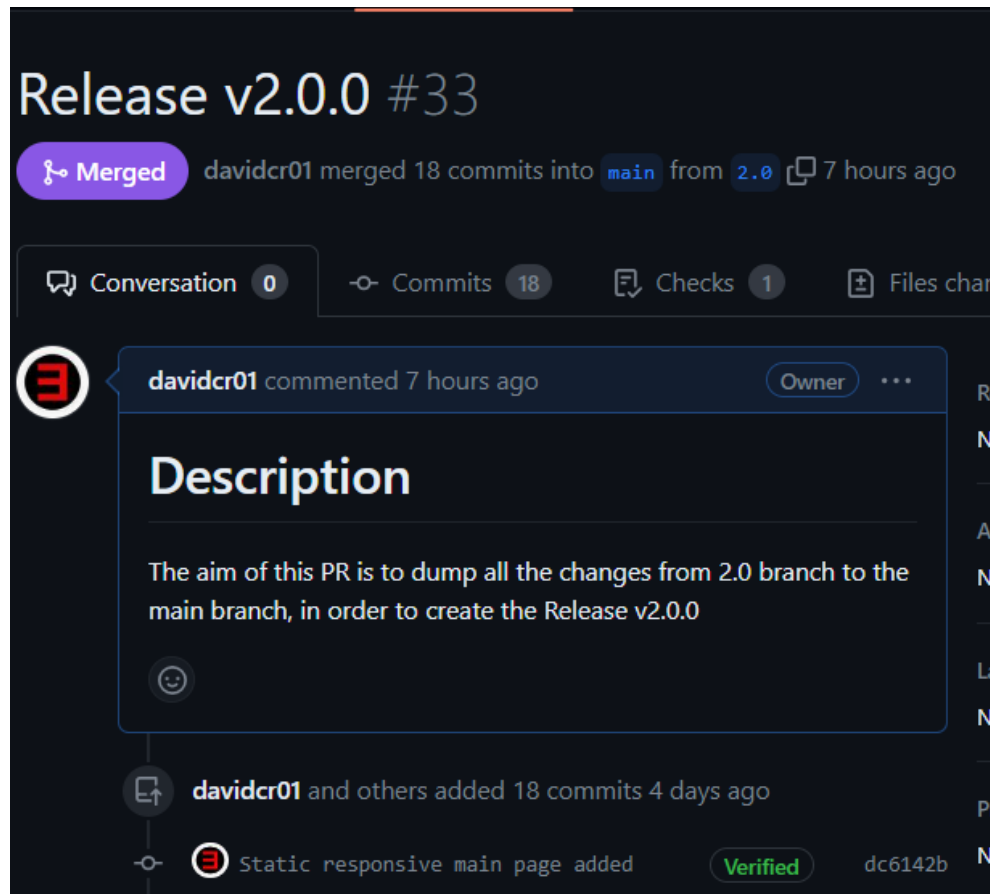


Figura 4. Incorporación de rama enumerada a principal.

5. Finalmente, se publica la *reléase* relacionada. Nótese que el *commit* de referencia de la *reléase* es el *Pull Request* mencionado anteriormente. En esta *reléase* se detallan los cambios más sustanciales al proyecto (Figura 5).

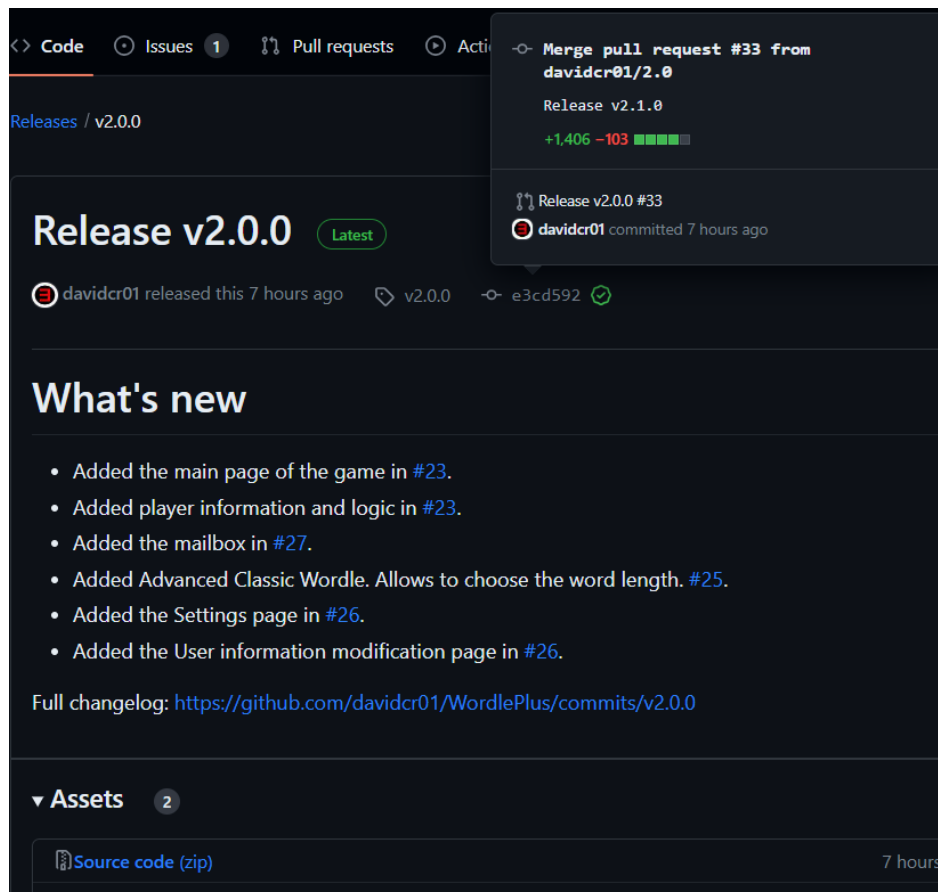


Figura 5. Publicando la *rélease*.

Como justificación, se indican las ventajas de este enfoque:

1. **Transparencia y accesibilidad:** permite a los usuarios y colaboradores acceder de manera fácil y transparente a toda la documentación, código fuente y registros de desarrollo en un solo lugar.
2. **Aprendizaje y colaboración:** facilita el aprendizaje de otros usuarios al proporcionar una visión detallada del progreso del desarrollo, permitiendo que examinen distintas perspectivas, problemas y soluciones. También fomenta la colaboración al permitir la participación y contribución de la comunidad de desarrolladores.
3. **Seguimiento y gestión eficiente:** el uso de issues y tareas asociadas permite un seguimiento sistemático del progreso del desarrollo, identificando las historias de usuario, errores y tareas pendientes de manera clara. Esto facilita la asignación de responsabilidades, la priorización de tareas y la gestión del flujo de trabajo.
4. **Control de versiones y liberaciones:** la estructura de ramas y la fusión con la rama principal (*main*) permiten un control de versiones sólido, lo que facilita la gestión de múltiples funcionalidades, la solución de errores y la implementación de nuevas versiones del producto.
5. **Registro detallado de cambios:** la documentación en GitHub proporciona un registro detallado de todos los cambios realizados en el

desarrollo, lo que facilita la revisión y la comprensión de las actualizaciones y mejoras implementadas en cada versión.

6. **Mejora continua:** La posibilidad de crear *milestones* y establecer hitos de desarrollo permite una planificación efectiva y una mejora continua del producto, manteniendo un flujo constante de actualizaciones y nuevas funcionalidades.

El repositorio es el siguiente: <https://github.com/davidcr01/WordlePlus/tree/main>

III.2. Diseño de la base de datos

III.2.1. Análisis del contexto

Para realizar un correcto diseño de la base de datos, previamente a caracterizar éste a la tecnología a utilizar, es necesario hacer un análisis de toda la información necesaria a almacenar y sus respectivas relaciones.

Para ello, y recapitulando todas las características principales del proyecto, existen varias entidades básicas de las que tenemos que almacenar información:

- Usuarios
- Partidas
- Torneos

De los usuarios derivan: los jugadores, los administradores, la lista de amigos, las peticiones de amistad y las notificaciones.

De los torneos derivan: las participaciones de los jugadores en los torneos y las rondas de cada torneo.

Con esta información, podemos identificar algunos elementos que se darán en la base de datos, añadiendo también los campos asociados:

- **Usuarios:** son los usuarios que utilizarán la aplicación. Sus campos asociados son: nombre de usuario, correo, nombre, apellidos, fecha de registro, si es mánager o no):
 - o **Jugadores:** son aquellos usuarios que utilizarán la plataforma para realizar las actividades relacionadas con el juego. Sus campos relacionados son: nombre de usuario, email, wordles completados, partidas 1vs1 ganadas, torneos ganados, experiencia, categoría y avatar)
 - o **Administradores (Gestores de Eventos):** son aquellos usuarios que se encargan de la gestión de los torneos de la plataforma. Sus campos asociados son: nombre de usuario, correo, nombre, apellidos, fecha de registro

- **Lista de amigos:** representa la lista de amigos de un jugador. Sus campos asociados son: nombre de usuario de un jugador, nombre de usuario de otro jugador.
- **Solicitudes de amigos:** representa las solicitudes de amistad de un jugador. Sus campos asociados son: nombre de usuario del remitente, nombre de usuario del destinatario, la fecha de creación y si es válida o no.
- **Notificaciones:** representa todas las notificaciones relacionadas con un jugador. Sus campos asociados son: identificador de la notificación, nombre de usuario asociado a la notificación, texto asociado y *link* asociado.
- **Partidas:** representa las partidas 1 contra 1 entre jugadores. Sus campos asociados son: identificador de la partida, nombre de usuario del jugador 1, nombre de usuario del jugador 2, si el jugador 1 ha jugado, si el jugador 2 ha jugado, y quién es el ganador.

Nota: las partidas serán asíncronas, es decir, no es necesario que ambos jugadores estén al mismo tiempo jugando la partida, por lo que es necesario los campos de si cada jugador ha jugado la partida.

- **Torneos:** representa los eventos entre jugadores cuya competición está relacionada con partidas de Wordle 1 contra 1. Sus campos asociados son: identificador del torneo, nombre, descripción, número máximo de jugadores y si está cerrado o no. Un torneo se considerará cerrado si:
 - o De por sí, es un torneo cerrado, es decir, si es un torneo el cual los jugadores de la plataforma no se pueden unir por voluntad propia. Los participantes de este tipo de torneo estarán seleccionados por los administradores.
 - o El número de jugadores inscritos en el torneo es igual a su número máximo permitido. Este caso se da en los torneos abiertos, en los que cualquier jugador puede participar.
- **Rondas:** los torneos se subdividen en rondas, en donde en cada ronda se obtiene el jugador ganador de cada partida 1 contra 1. Sus campos asociados son: identificador de la ronda, identificador del torneo asociado, número de la ronda, partida asociada a dicha ronda

En el análisis podemos notar lo siguiente:

- Las entidades **Jugadores** y **Gestores de eventos** son entidades heredadas de **Usuarios**, por lo que tendrán una relación de herencia *es-un*.
- La entidad **Gestores de eventos** resulta innecesaria al tener un subconjunto (y solamente dicho subconjunto) de los campos de la entidad **Usuarios**.
- Existen varias entidades que dependen existencialmente de otras, es decir, la existencia de una entidad es obligatoria para la existencia de otra

entidad. Estas restricciones se detallarán más adelante, una vez diseñado el diagrama Entidad-Relación.

III.2.2. Diagrama Entidad-Relación

Un modelo de datos entidad-relación está basado en una percepción del mundo real que consta de una colección de objetos básicos, llamados entidades, los cuales pueden tener propiedades o atributos asociados; y de relaciones, que son conexiones entre las entidades.³⁶

Teniendo en cuenta todos los aspectos destacados en el previo análisis, un posible diagrama Entidad-Relación sería el siguiente (Figura 6):

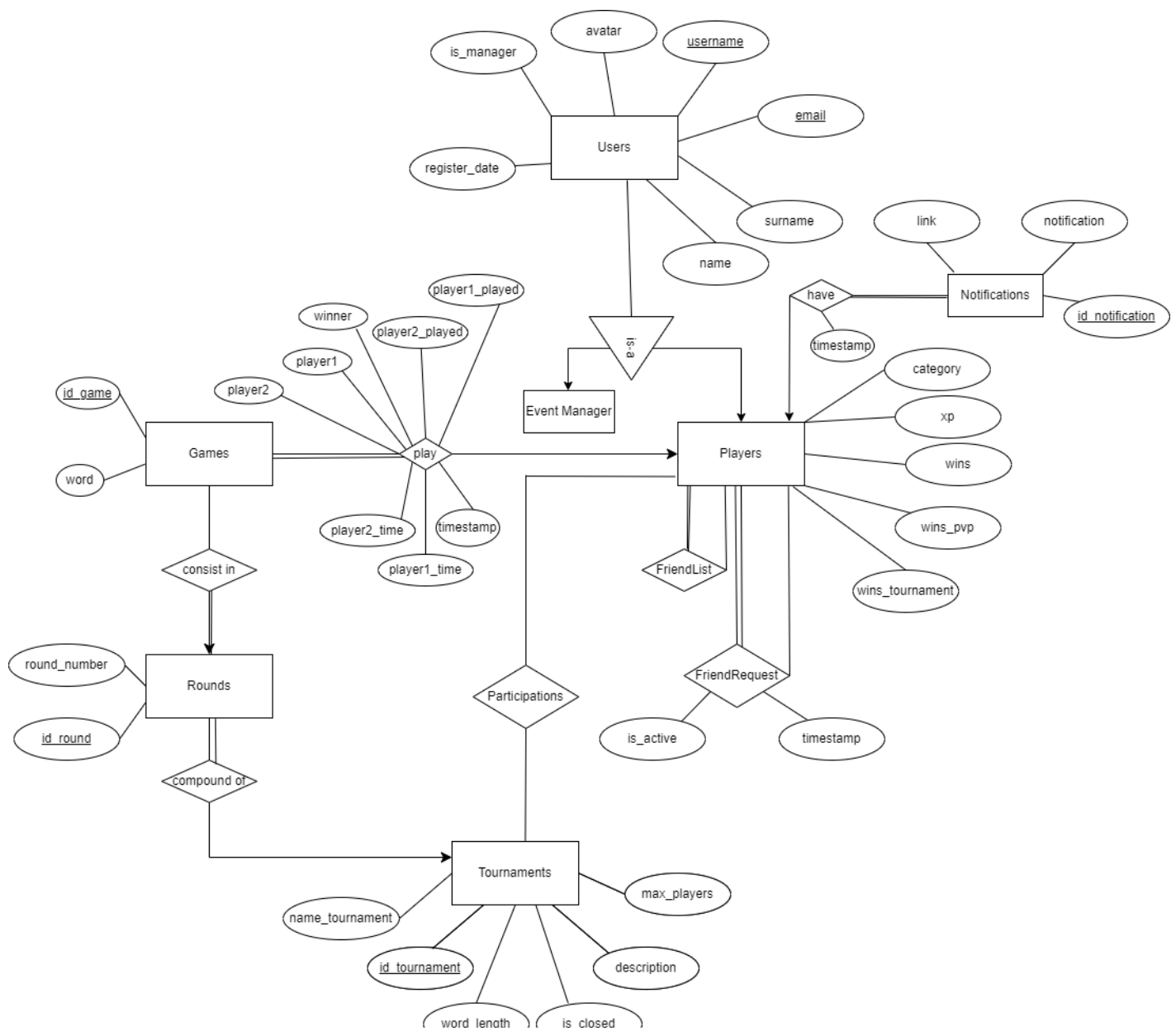


Figura 6. Diagrama entidad-relación inicial.

Algunas aclaraciones a tener en cuenta son:

1. Como se ha comentado anteriormente, la entidad *Event Manager* es una especialización de la entidad *Users* sin ningún atributo extra, por lo que añadiendo el campo *is_manager* a la entidad *Users* no es necesario crear una tabla nueva para *Event Manager*. La representación de los administradores en la base de datos puede hacerse con el campo *is_staff* de la tabla *Users*. En cambio, la entidad *Players*, al tener muchos más campos, si requiere una tabla aparte en donde almacenar dicha información.
2. Respecto a la entidad *Notifications* y la relación *Have*, un jugador puede tener muchas notificaciones, pero esa notificación solamente puede pertenecer a un jugador, por lo que la relación *have* entre *Players* y *Notifications* es de 1:N. Además, la entidad de notificaciones depende existencialmente de la entidad *Players*, ya que no tiene sentido una notificación que no tenga asociado un jugador. La entidad *Notifications* heredará como clave externa la clave de la entidad *Player*.
3. Respecto a la relación recursiva *FriendList* entre la entidad *Players*, un jugador puede ser amigo de muchos jugadores, y también viceversa, por lo que presenta una cardinalidad N:M. Además, la tabla resultante de esta relación presenta una obligatoriedad con la entidad *Players* ya que las relaciones de amistad solamente tienen sentido entre instancias de la entidad *Players*. Esta relación heredará las claves primarias de la entidad *Players*.
4. Respecto a la relación *FriendRequest*, presenta ciertas similitudes con la relación anterior, pero con ciertos atributos extra. Un jugador puede tener varias peticiones de amistad, y una misma petición de amistad puede pertenecer a varios jugadores, por lo que presenta una cardinalidad N:M, pero en este caso solamente puede pertenecer a uno de los dos jugadores, pero no de forma simultánea. Esta condición deberá comprobarse antes de introducir valores a la tabla resultante. La tabla resultante de esta relación exige una obligatoriedad con la entidad *Players* con la misma justificación que el punto anterior.
5. Respecto a la relación *Play* entre las entidades *Players* y *Games*, un jugador puede jugar muchas partidas, y esa partida puede ser jugada exactamente por dos jugadores, por lo que la relación presenta una cardinalidad N:2, con obligatoriedad de las partidas hacia los jugadores, ya que sin jugadores las partidas no pueden ser jugadas.
6. Respecto a la relación *Consist in* entre las entidades *Games* y *Rounds*, una ronda consiste en varias partidas, pero una partida de un torneo en concreto solamente puede pertenecer a una ronda, ya que en un torneo no se repiten enfrentamientos entre los mismos jugadores en diferentes

momentos (en torneos eliminatorios), por lo que la relación presenta una cardinalidad 1:N. Existe obligatoriedad de las rondas hacia las partidas, es decir, una ronda debe estar compuesta por partidas, ya que no tiene sentido que una ronda no tenga partidas.

7. Respecto a la relación *Compound of* entre las entidades *Tournaments* y *Rounds*, un torneo se compone de varias rondas, pero dicha ronda solamente puede pertenecer a un torneo en concreto, por lo que la relación presenta una cardinalidad 1:N. Además, existe obligatoriedad entre las rondas y los torneos, ya que una ronda no tiene sentido sin antes existir un torneo que la contenga.
8. Respecto a la relación *Participations* entre las entidades *Tournaments* y *Players*, un jugador puede participar en muchos torneos, y en un mismo torneo pueden participar muchos jugadores, por lo que la relación presenta una cardinalidad N:M. No existe obligatoriedad entre estas entidades, ya que la existencia de los torneos y los jugadores son independientes.

III.2.3. Paso a tablas

Una vez está diseñado el esquema conceptual de la base de datos, podemos realizar un modelo lógico de la base de datos realizando el “paso a tablas”. En este proceso, se deben tener las siguientes consideraciones:

- Los atributos de las **entidades fuertes** se representarán por medio de una tabla en donde cada tupla es una ocurrencia del conjunto de entidades y está caracterizada por n columnas distintas, una por cada atributo. La clave primaria de la tabla está constituida por los atributos que forman la clave primaria en el conjunto de entidades.
- Los atributos de las **entidades débiles** se representarán de forma similar a las entidades fuertes, pero considerando que la clave primaria de la tabla estará constituida por los atributos que forman la clave primaria de la entidad de la que depende, más los atributos marcados como discriminadores o claves parciales de la entidad débil si existen. Además, hay que generar una clave externa que referencia a la entidad de la que depende. En este caso, no existen entidades débiles.
- Los atributos de las **relaciones** se representarán en tablas en donde la clave primaria dependerá de la cardinalidad de dicha relación:
 - o Si la relación es de muchos a muchos, la clave primaria estará formada por la unión de las claves primarias de los conjuntos de entidades que intervienen en la relación, con posibilidad de añadir algunos atributos de la relación.

- o Si la relación es de muchos a uno, la clave primaria estará formada por la clave primaria de la entidad con la cardinalidad de muchos.
- o Si la relación es de uno a uno, existirán dos claves candidatas de las cuales una de ellas será la clave primaria.

- Los atributos de las relaciones de **herencia** se representarán en tablas en donde el conjunto de entidades más general pasa a ser una tabla según su tipo de entidad, y cada conjunto de entidades de nivel inferior será una tabla constituida por los atributos propios más la clave primaria de la entidad superior.

En cualquier caso, los atributos que identifican a las claves de otras entidades hay que establecerlos como claves externas a las claves primarias de dichas entidades.

Con esto, las tablas resultantes del diagrama Entidad-Relación son las siguientes (Figura 7 y 8):

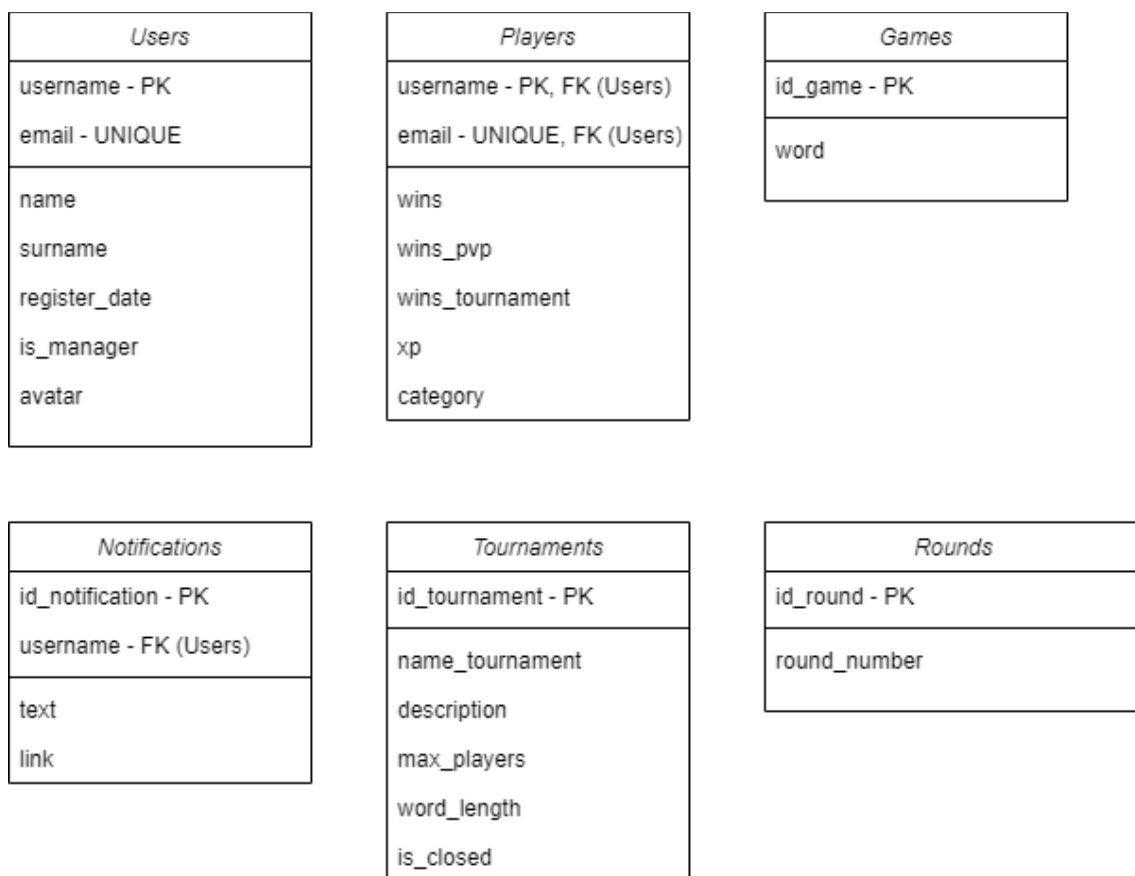


Figura 7. Entidades del paso a tablas.

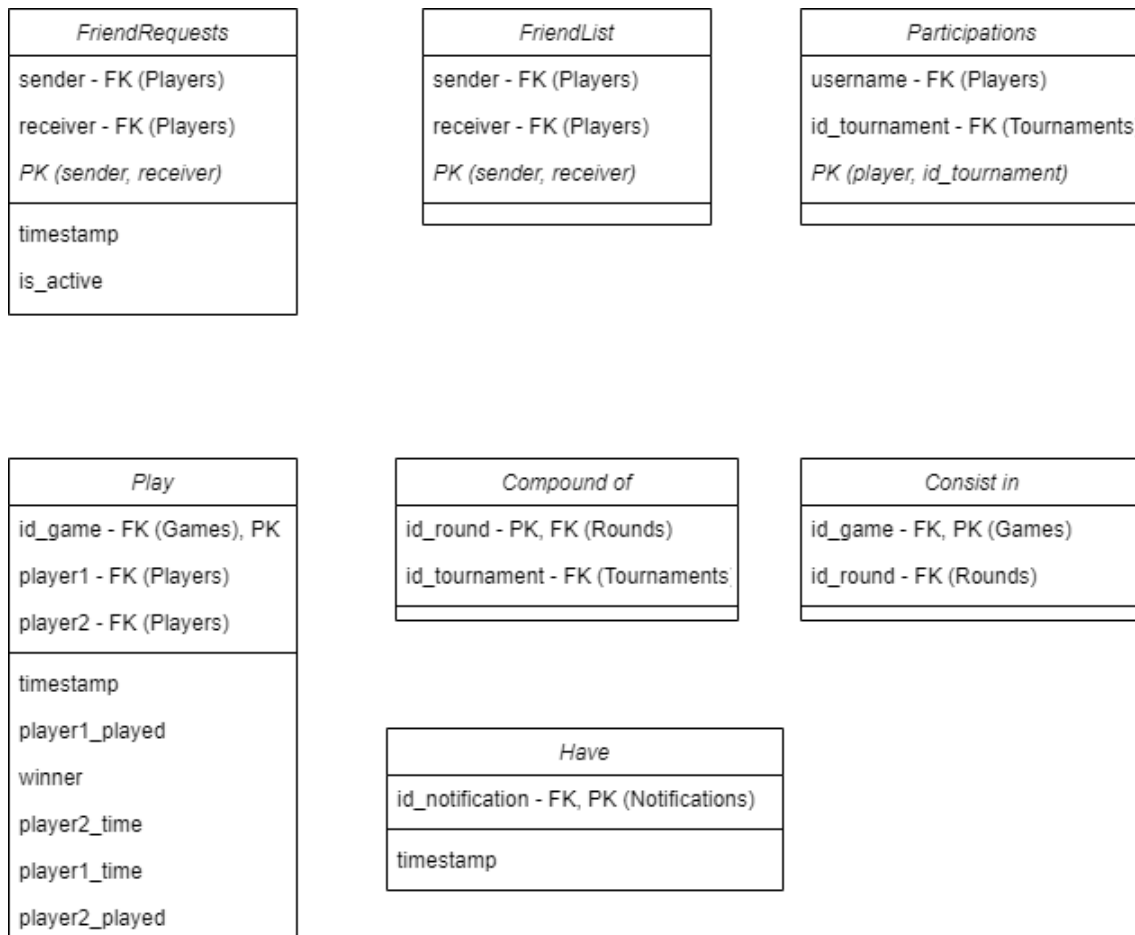


Figura 8. Relaciones del paso a tablas.

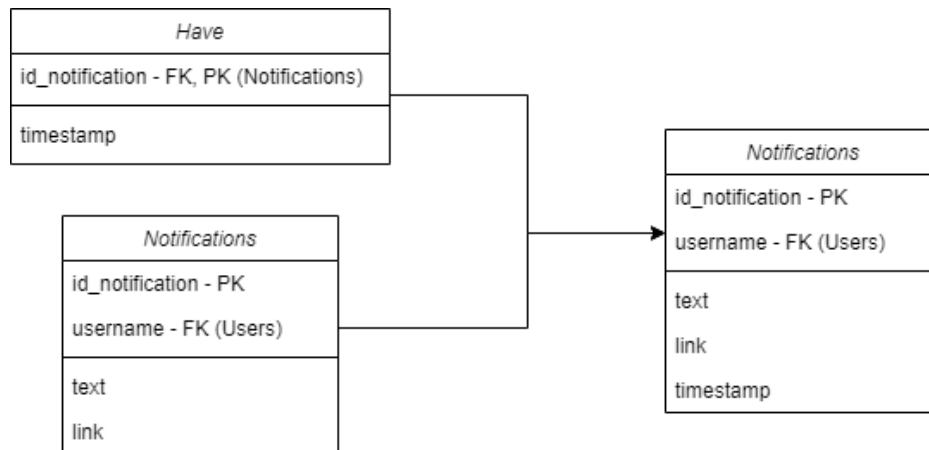
III.2.4. Fusiones

Las fusiones consisten en combinar varias tablas, permitiendo una reducción del número de estas, siempre y cuando no se pierda información (tanto de datos como de restricciones). Las fusiones también permiten mejoras de eficiencia a nivel de almacenamiento, es decir, se ahorra espacio de almacenamiento; y rendimiento del sistema, ya que las búsquedas pueden verse aceleradas al compactar la información.

Las condiciones necesarias que se deben dar para realizar una fusión son:

- Que las tablas tengan la misma clave primaria o candidata.
 - Que ninguna de las tablas proceda de herencia.
 - Que semánticamente la fusión tenga sentido.
1. La fusión entre las tablas *Users* y *Players* no puede darse, al pertenecer la tabla *Players* de una herencia de la Tabla *Users*.
 2. La fusión entre las tablas *Have* y *Notifications* puede darse, al tener la misma clave primaria (*id_notification* de *Notifications*) y ser

semánticamente compatibles, ya que una notificación siempre va a estar asociada a un jugador. Cabe mencionar que no se puede dar la fusión entre *Have* y *Players* ya que un jugador puede que no tenga notificaciones en ningún momento.



3. La fusión entre las tablas *Players* y *FriendList* no puede darse, ya que un jugador no tiene por qué tener una lista de amigos. De forma similar, tampoco puede darse entre *Players* y *FriendRequest*.
4. La fusión entre las tablas *Players* y *Participations* no puede darse, ya que no siempre un jugador va a participar en un torneo. De forma similar, tampoco puede darse entre *Tournaments* y *Participations* ya que pueden existir torneos sin participantes. Este es el caso temporal de los torneos abiertos, en los que en un momento determinado (en su creación) los torneos abiertos no tienen participantes hasta que éstos no ingresan en el torneo.
5. La fusión entre las tablas *Games* y *Play* puede darse, al tener la misma clave primaria (*id_game* de *Games*) y ser semánticamente compatibles, ya que una partida siempre va a estar relacionada directamente con dos jugadores. Cabe mencionar que no se puede dar la fusión entre *Players* y *Play*, ya que podrían existir jugadores que nunca han jugado a ninguna partida (Figura 9).

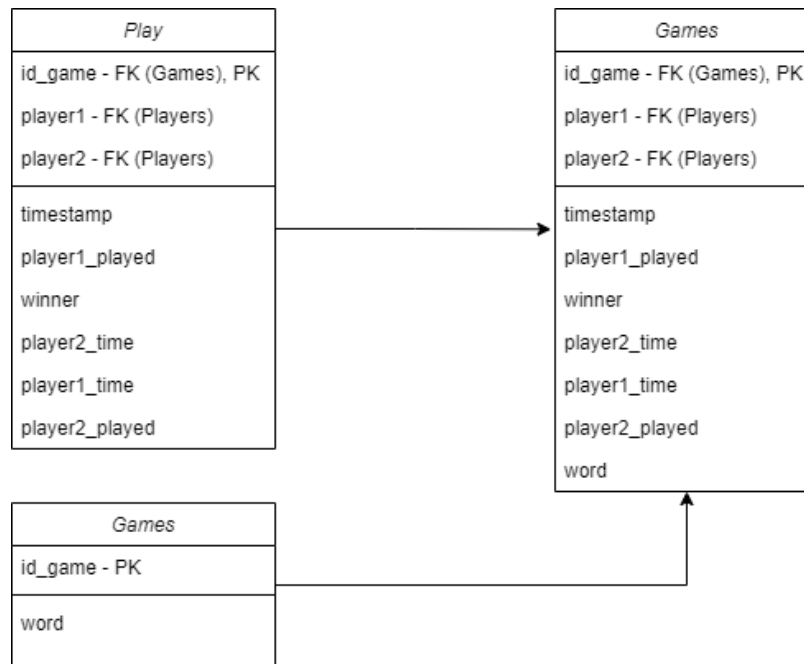


Figura 9. Fusion de *Play* y *Games*.

6. La fusión entre las tablas *Rounds* y *compound of* puede darse, al tener la misma clave primaria (*id_round* de *Rounds*) y ser semánticamente compatibles, ya que una ronda siempre estará relacionada con un torneo existente. Cabe mencionar que no se puede dar la fusión entre *Rounds* y *consist in* ya que no tienen la misma clave primaria (Figura 10).

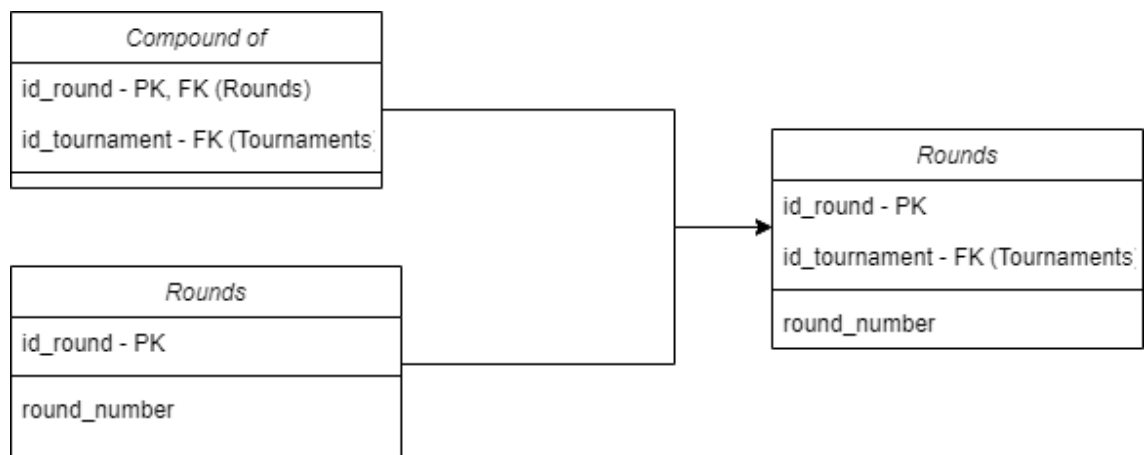


Figura 10. Fusion de *Compound of* y *Rounds*.

Por tanto, y como resultado de las fusiones anteriores, el modelado de tablas de la base de datos queda de la siguiente manera (Figura 11):

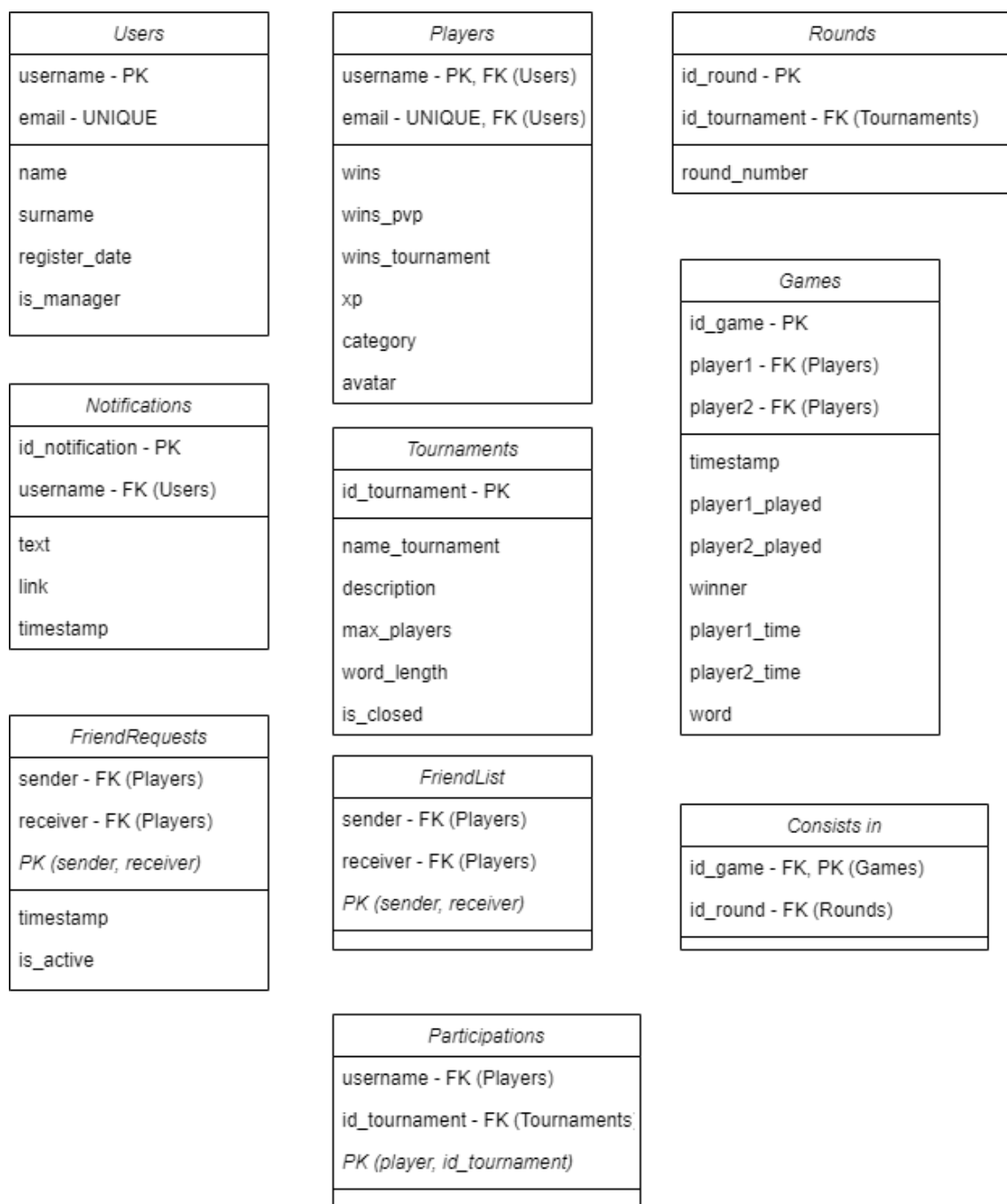


Figura 11. Fusiones sin normalización.

III.2.5. Normalización

El proceso de normalización³⁷ de base de datos consiste en aplicar una serie de procedimientos a las tablas obtenidas tras el paso del modelo entidad-relación al modelo relacional (o tablas) para conseguir minimizar la redundancia de los datos.

Los objetivos de la normalización son:

- Minimizar la redundancia de los datos.
- Reducir los problemas de actualización de los datos en las tablas.
- Proteger la integridad de los datos.

Existen varios tipos de normalización, que incluyen la Primera Forma Normal (1FN), Segunda Forma Normal (2FN), Tercera Forma Normal (3FN), Forma normal de Boyce-Codd (FNBC), Cuarta Forma Normal (4FN) y Quinta Forma Normal (5FN). En general, las primeras tres formas normales son el mínimo que deben cubrir todas las bases de datos, y es recomendable normalizarlas hasta la Forma Normal de Boyce-Codd. Se dice que una base de datos está normalizada en la forma normal N si todas sus tablas están en la forma normal N.

Es necesario mencionar que un atributo no primo (o no primario) es aquel que no pertenece a ninguna clave candidata.

De esta manera, las tablas obtenidas en el punto anterior serán normalizadas hasta FNBC.

III.2.5.1. Primera Forma Normal (1FN)

Las reglas de la 1FN son:

- 1) Todos los atributos son atómicos, es decir, los elementos del dominio son simples e indivisibles.
- 2) No existe variación entre el número de columnas.
- 3) Los campos no clave se identifican por la clave.
- 4) Hay una independencia del orden tanto de las filas como de las columnas.

Si analizamos todas las tablas resultantes del paso a tablas, observamos que todas las tablas están en 1FN:

- **Users:**
 - 1) Sus atributos asociados (*username*, *email*, *name*, *surname*, *register_date*, *is_manager*) son de dominios simples.
 - 2) El número de columnas no varía.
 - 3) Todos los campos que no son clave están identificados por la clave.
 - 4) Si los datos cambian de orden no cambian sus significados.
- **Players:**
 - 1) Sus atributos asociados (*username*, *wins*, *wins_pvp*, *wins_tournament*, *xp*, *category*, *avatar*) son de dominios simples.

- 2) El número de columnas no varía.
 - 3) Todos los campos que no son clave están identificados por la clave.
 - 4) Si los datos cambian de orden no cambian sus significados.
- **Rounds:**
 - 1) Sus atributos asociados (*id_round, id_tournament, round_number*) son de dominios simples.
 - 2) El número de columnas no varía.
 - 3) Todos los campos que no son clave están identificados por la clave.
 - 4) Si los datos cambian de orden no cambian sus significados.
 - **Notifications:**
 - 1) Sus atributos asociados (*id_notification, username, text, link, timestamp*) son de dominios simples.
 - 2) El número de columnas no varía.
 - 3) Todos los campos que no son clave están identificados por la clave.
 - 4) Si los datos cambian de orden no cambian sus significados.
 - **Tournaments:**
 - 1) Sus atributos asociados (*id_tournament, name_tournament, description, max_players, word_length, is_closed*) son de dominios simples.
 - 2) El número de columnas no varía.
 - 3) Todos los campos que no son clave están identificados por la clave.
 - 4) Si los datos cambian de orden no cambian sus significados.
 - **Games:**
 - 1) Sus atributos asociados (*id_game, player1, player2, timestamp, player1_played, player2_played, winner, player1_time, player2_time, word*) son de dominios simples.
 - 2) El número de columnas no varía.
 - 3) Todos los campos que no son clave están identificados por la clave.
 - 4) Si los datos cambian de orden no cambian sus significados.
 - **FriendRequests**
 - 1) Sus atributos asociados (*sender, receiver, timestamp, is_active*) son de dominios simples.
 - 2) El número de columnas no varía.
 - 3) Todos los campos que no son clave están identificados por la clave.
 - 4) Si los datos cambian de orden no cambian sus significados.
 - **FriendList**
 - 1) Sus atributos asociados (*sender, receiver*) son de dominios simples.
 - 2) El número de columnas no varía.
 - 3) Todos los campos que no son clave están identificados por la clave.
 - 4) Si los datos cambian de orden no cambian sus significados.

- **Consist in**
 - 1) Sus atributos asociados (*id_game*, *id_round*) son de dominios simples.
 - 2) El número de columnas no varía.
 - 3) Todos los campos que no son clave están identificados por la clave.
 - 4) Si los datos cambian de orden no cambian sus significados.
- **Participations**
 - 1) Sus atributos asociados (*username*, *id_tournament*) son de dominios simples.
 - 2) El número de columnas no varía.
 - 3) Todos los campos que no son clave están identificados por la clave.
 - 4) Si los datos cambian de orden no cambian sus significados.

Al encontrarse todas las tablas en 1FN, se concluye que la base de datos se encuentra en 1FN.

III.2.5.2. Segunda Formal Normal (2FN)

La condición necesaria de la 2FN es que la base de datos esté en 1FN, y, además, que los atributos que no forman parte de ninguna clave dependan de forma completa de la clave principal, es decir, que no haya dependencias parciales. Todos los atributos que no pertenezcan a la clave deben depender únicamente de la clave. Este requisito está basado en el concepto de dependencia funcional.³⁷

Cabe recalcar que las tablas que están en 1FN, y que además disponen de una clave primaria formada por una única columna con valor indivisible, cumple con la 2FN.

Con esta aclaración, las tablas que cumplen dicha condición son: *Users*, *Players*, *Rounds*, *Notifications*, *Tournaments*, *Games*, y *Consist in*, ya que su clave primaria solamente está formada por un único campo.

Las tablas restantes, las cuales son necesarias analizar si están en 2FN son: *FriendRequests*, *FriendList* y *Participations*.

- **FriendRequests:**
 - 1) La clave primaria de esta tabla está formada por la combinación de los usuarios involucrados en la petición de amistad (*sender*, *receiver*).
 - 2) El atributo *timestamp* representa la fecha de creación de la petición de amistad, y es totalmente dependiente de toda la clave primaria. Este campo no puede depender solamente del campo *sender* ya que no se conocería quién es el que recibe la petición; y tampoco puede depender solamente de *receiver* ya que no se conocería quién es el que la manda.

- 3) El atributo *is_active* representa si la solicitud de amistad está activa o no. Controla si la petición se ha cancelado, rechazado o aceptado. Este campo es totalmente dependiente de la clave primaria por la misma razón que el atributo *timestamp*, ya que es un atributo que relaciona a ambos jugadores.

Por tanto, esta tabla se encuentra en 2FN.

- **FriendList:**

- 1) La clave primaria de esta tabla está formada por la combinación de los usuarios involucrados en la relación de amistad (*sender*, *receiver*).
- 2) Esta tabla no tiene más atributos asociados, por lo que no existen dependencias parciales.

Por tanto, esta tabla se encuentra en 2FN.

- **Participations:**

- 1) La clave primaria de esta tabla está formada por la combinación del nombre de usuario del jugador y el identificador de torneo (*username*, *id_tournament*).
- 2) Esta tabla no tiene más atributos asociados, por lo que no existen dependencias parciales.

Por tanto, esta tabla se encuentra en 2FN.

Al encontrarse todas las tablas en 2FN, se concluye que la base de datos se encuentra en 2FN.

III.2.5.3. Tercera Formal Normal (3FN)

La condición necesaria para que una base de datos se encuentre en 3FN es que se encuentre en 2FN y, además, no contenga ninguna dependencia transitiva entre los atributos que no son clave.

Una dependencia transitiva es una dependencia funcional $X \rightarrow Z$ en la cual Z no es inmediatamente dependiente de X , pero sí de un tercer conjunto de atributos Y , que a su vez depende de X (y siempre que no ocurra que X sea también dependiente de Y). Es decir, $X \rightarrow Z$ por virtud de $X \rightarrow Y$ e $Y \rightarrow Z$ (y no ocurre que $Y \rightarrow X$). Dicho de otra manera, las columnas que no pertenezcan a la clave primaria deben depender solamente de la clave, y no de otra columna que no sea clave.³⁸

A continuación, se expone un análisis de cada tabla de la base de datos, comprobando si se cumplen las condiciones de la 3FN:

- **Users:**
 - 1) Sus atributos no primos (*name*, *surname*, *register_date*, *is_manager*) solamente dependen de la clave primaria (*username*) o de la clave candidata (*email*), y no existen dependencias entre ellos.
- **Players:**
 - 1) Sus atributos no primos (*wins*, *wins_pvp*, *wins_tournament*, *xp*, *category*, *avatar*) solamente dependen de la clave primaria (*username*) o de la clave candidata (*email*), y no existen dependencias entre ellos, excepto el campo *category*.
 - 2) El campo *category* depende del campo *xp*, y el campo *xp* depende de la clave primaria *username*. Por tanto, existe una dependencia transitiva que es necesario eliminar para cumplir las condiciones de la 3FN.
 - 3) Para ello, esta tabla se divide en dos, eliminando la dependencia transitiva (Figura 12). De esta manera, el campo *category* de la tabla *Players* se elimina, y se crea una nueva tabla *Categories* que contiene la información asociada (el identificador, el nombre de la categoría y el umbral de experiencia de dicha categoría)
 - 4) Con esta división, la dependencia transitiva ya no existe.

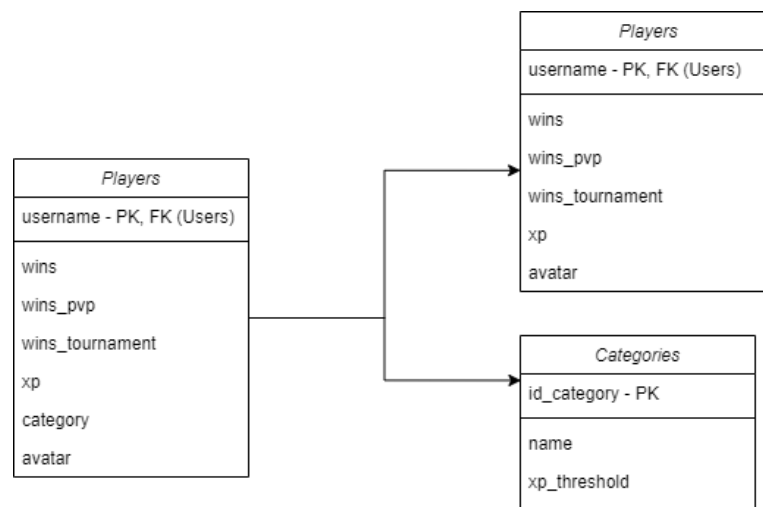


Figura 12. Eliminación de dependencia transitiva de *Players*.

- **Rounds:**
 - 1) Sus atributos no primos (*id_tournament*, *round_number*) solamente dependen de la clave primaria (*id_round*), y no existen dependencias entre ellos.
- **Notifications:**
 - 1) Sus atributos no primos (*username*, *text*, *link*, *timestamp*) solamente dependen de la clave primaria (*id_notification*), y no existen dependencias entre ellos.

- **Tournaments:**
 - 1) Sus atributos no primos (*name_tournament*, *description*, *max_players*, *word_length*, *is_closed*) solamente dependen de la clave primaria (*id_tournament*), y no existen dependencias entre ellos.
- **Games:**
 - 1) Sus atributos no primos (*player1*, *player2*, *timestamp*, *player1_played*, *player2_played*, *winner*, *player1_time*, *player2_time*, *word*) solamente dependen de la clave primaria (*id_game*), y no existen dependencias entre ellos.
- **FriendRequests**
 - 1) Sus atributos no primos (*timestamp*, *is_active*) solamente dependen de la clave primaria (*sender*, *receiver*), y no existen dependencias entre ellos.
- **FriendList**
 - 1) No contiene atributos no primos, solamente contiene su clave primaria (*sender*, *receiver*).
- **Consist in**
 - 1) El único atributo no primo (*id_round*) depende de la clave primaria (*id_game*).
- **Participations**
 - 1) No contiene atributos no primos, solamente contiene su clave primaria (*username*, *id_tournament*).

Al encontrarse todas las tablas en 2FN, se concluye que la base de datos se encuentra en 3FN.

III.2.5.4. Forma Normal de Boyce-Codd (FNBC)

La FNBC es una versión ligeramente más restrictiva que la 3FN, que impone que una tabla estará en FNBC si está en 3FN, y además no existen dependencias funcionales no triviales de los atributos que no sean un conjunto de la clave candidata. Dicho de otra manera, se cumple la 3FN si y sólo si todo determinante es una clave candidata, donde determinante de una relación es todo conjunto de atributos del cual depende de forma completa otro atributo de la relación.

Una forma sencilla de comprobar si una tabla se encuentra en FNBC es que no tenga clave candidatas compuestas, es decir, con varios atributos.

En esta base de datos, las únicas tablas con claves candidatas son *Users* y *Players* con el atributo *email*, que no es una clave candidata compuesta, sino simple. Por lo que se concluye que la base de datos se encuentra en FNBC.

Como conclusión, y tras el paso a tablas y la normalización, las tablas resultantes de dichos procesos son las siguientes (Figura 13):

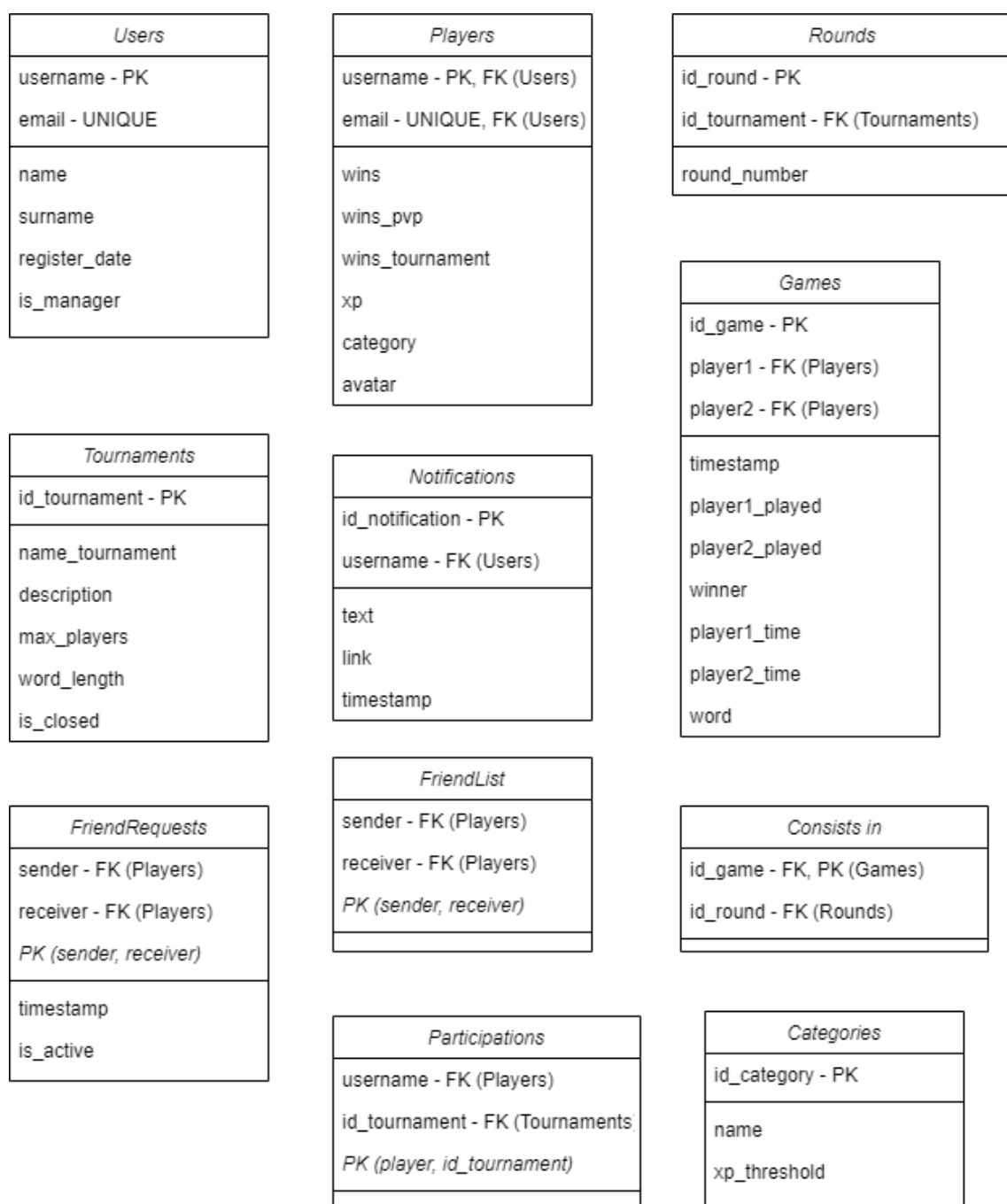


Figura 13. Tablas tras normalización.

III.3. Implementación de los Sprints

III.3.1. Sprint 1

Nº	Objetivo	Fecha de entrega	
1	Infraestructura del proyecto. Registro de jugadores	13 de febrero del 2023	
Ident.	Título	Estimación	Prioridad
HT	Establecer la infraestructura del proyecto.	4	1
HU.1	Como jugador quiero poder registrarme	2	1

El objetivo de este sprint es construir la infraestructura del proyecto y hacer una primera toma de contacto con las tecnologías escogidas desarrollando la creación de jugadores.

En sintonía con el proyecto de GitHub³⁵, este Sprint está estrechamente relacionado con los siguientes *milestones*. Se pueden consultar las *issues* relacionadas de cada *milestone*:

- <https://github.com/davidcr01/WordlePlus/milestone/2>
- <https://github.com/davidcr01/WordlePlus/milestone/1>

III.3.1.1. Infraestructura

Como se comentó en el capítulo [III.1.3. Despliegue](#), se utilizaría la herramienta de gestión de contenedores *Docker*³⁴ para albergar el proyecto, concretamente la utilidad *docker-compose*, que permite orquestar distintos contenedores y mantener una comunicación entre ellos.

Cabe mencionar que la estructura de directorios será la siguiente:

- **Data:** este directorio albergará todos los ficheros de la base de datos gestionados por el motor de *PostgreSQL*³⁰
- **Django:** este directorio alberga el proyecto y la aplicación de *Django Rest Framework*³³ que representa el backend del proyecto junto con la carpeta *data*. Cabe destacar que la carpeta *data* solamente es la información de la base de datos, pero su configuración y gestión se realiza desde el proyecto de *Django Rest Framework*³³, de ahí que sean directorios al mismo nivel.

- **ionic**: este directorio alberga la aplicación de Ionic Framework²⁸, que representa el *frontend* del proyecto.

Para orquestar las distintas partes, es necesario construir un *Dockerfile* para el *backend* y otro para el *frontend*. Estos ficheros definen cómo se crearán las imágenes Docker para posteriormente ser utilizadas por contenedores. Para gestionar y orquestar estos contenedores, es necesario construir un fichero *docker-compose.yaml*.

Dentro de la carpeta *django*, se crea un *Dockerfile* con el siguiente contenido:

```
FROM python:3

# Set environment variables
ENV PYTHONUNBUFFERED 1

COPY requirements.txt /

# Install dependencies.
RUN pip install -r /requirements.txt

# Set work directory.
RUN mkdir /code
WORKDIR /code

# Copy project code.
COPY . /code/

EXPOSE 80
```

Figura 14. Dockerfile de Django.

Este *Dockerfile* (Figura 14):

- Define la imagen base utilizada para construir el contenedor. En este caso, se utiliza la imagen oficial de *Python*³² versión 3.
- Establece una variable de entorno que evita que la salida se almacene en búfer, lo que permite que se muestre inmediatamente en la consola
- Copia el archivo *requirements.txt* del directorio local al directorio raíz del contenedor.
- Ejecuta el comando *pip install* dentro del contenedor para instalar las dependencias especificadas en el archivo *requirements.txt*. Estas dependencias son los paquetes de *Python*³² necesarios para que la aplicación funcione correctamente.
- Crea un directorio llamado */code* dentro del contenedor.
- Establece el directorio de trabajo actual dentro del contenedor como */code*. A partir de ahora, todos los comandos se ejecutarán en este directorio.
- Copia el código del proyecto actual, incluidos todos los archivos y directorios, al directorio */code* del contenedor.

- Expone el puerto 80 del contenedor, lo que permite que la aplicación se ejecute en ese puerto y esté accesible desde fuera del contenedor.

En resumen, se configura el entorno del contenedor con *Python3*³², se instalan las dependencias necesarias y se copia el código al contenedor.

La issue relacionada en *GitHub*³⁵ es [Dockerize the backend \(S1\)](#).

Respecto al *frontend*, dentro de la carpeta *ionic* se crea un *Dockerfile* con el siguiente contenido:

```
FROM node:18-alpine as build
RUN mkdir /app
WORKDIR /app
COPY ionic-app/package*.json /app/ionic-app/
RUN npm install --prefix ionic-app
COPY ./ /app/
RUN npm run-script build --prefix ionic-app -- --output-path=./dist/out
FROM nginx:alpine
RUN rm -rf /usr/share/nginx/html/*
COPY --from=build /app/ionic-app/dist/out /usr/share/nginx/html/
COPY ./nginx/nginx.conf /etc/nginx/conf.d/default.conf
```

Figura 15. Dockerfile de Ionic.

Este *Dockerfile* (Figura 15):

- Define la imagen base utilizada para la etapa de construcción del contenedor. En este caso, se utiliza la imagen de *Node.js* versión 18 basada en *Alpine Linux*.
- Crea un directorio llamado */app* dentro del contenedor.
- Establece el directorio de trabajo actual dentro del contenedor como */app*. A partir de ahora, todos los comandos se ejecutarán en este directorio.
- Copia los archivos *package.json* y *package-lock.json* del directorio *ionic-app* del proyecto al directorio */app/ionic-app* del contenedor.
- Ejecuta el comando *npm install* dentro del directorio */app/ionic-app* del contenedor para instalar las dependencias necesarias para la aplicación de *Ionic*²⁸.
- Copia todo el contenido del directorio raíz del proyecto al directorio */app* del contenedor.
- Ejecuta el comando *npm run build* dentro del directorio */app/ionic-app* del contenedor. Esto compila la aplicación de *Ionic* y genera los archivos de salida en el directorio */app/ionic-app/dist/out*.
- Define la imagen base utilizada para la etapa de producción del contenedor. En este caso, se utiliza la imagen de *NGINX*³⁶ basada en *Alpine Linux*.

- Elimina todos los archivos existentes en el directorio `/usr/share/nginx/html/` dentro del contenedor de *NGINX*³⁶.
- Copia los archivos generados durante la etapa de construcción del contenedor desde la etapa de compilación anterior (usando `--from=build`) al directorio `/usr/share/nginx/html/` del contenedor de *NGINX*³⁶. Estos archivos son los archivos de salida de la aplicación de Ionic compilada.
- Copia el archivo de configuración *nginx.conf* del directorio `nginx` del proyecto al directorio `/etc/nginx/conf.d/` del contenedor de *NGINX*³⁶. Esto reemplaza la configuración predeterminada de *NGINX*³⁶ con una configuración personalizada. El fichero *nginx.conf* es un fichero de configuración que establece el puerto y la ruta de los ficheros web a usar.

En resumen, se construye una imagen que primero compila una aplicación de *Ionic*²⁸ utilizando *Node.js* en la etapa de construcción, y luego utiliza una imagen de *NGINX*³⁹ en la etapa de producción para servir los archivos generados por la compilación de la aplicación de *Ionic*²⁸. Cabe destacar que podría utilizarse otro servidor web como *Apache*¹³, pero *NGINX*³⁶ ofrece un mayor rendimiento, menor consumo de recursos y mayor flexibilidad.

La issue de *GitHub*³⁵ relacionada es [Dockerize the frontend \(S1\)](#).

Ambos *Dockerfiles* serán utilizados por el fichero *docker-compose.yaml*. (Figura 16)

```

version: "3"

services:
  db:
    image: postgres
    volumes:
      - ./data/db:/var/lib/postgresql/data
    healthcheck:
      test: ["CMD-SHELL", "pg_isready -U postgres"]
      interval: 5s
      timeout: 5s
      retries: 5
    environment:
      - POSTGRES_DB=postgres
      - POSTGRES_USER=postgres
      - POSTGRES_PASSWORD=postgres
  dj:
    container_name: dj
    build: django
    command: python manage.py runserver 0.0.0.0:80
    volumes:
      - ./django:/code
    ports:
      - "80:80"
    environment:
      - POSTGRES_NAME=postgres
      - POSTGRES_USER=postgres
      - POSTGRES_PASSWORD=postgres
    depends_on:
      db:
        condition: service_healthy
  ng:
    container_name: ng
    build: ionic
    ports:
      - "8080:80"

```

Figura 16. docker-compose.yaml

Este fichero define 3 servicios a orquestar: db, dj y ng.

- **db:** representa la base de datos. Se define su imagen a utilizar *postgres* que se descargará automáticamente de *DockerHub*, se define el volumen a copiar en el contenedor, se define un comando de verificación de salud para asegurarse de que esté lista antes de que otros servicios dependan de ella, y por último se definen ciertas variables de entorno. La creación de la base de datos la hará automáticamente el servicio *dj* usando las variables de entorno.

- **dj**: un servicio que se construye a partir del *Dockerfile* definido en la carpeta *django*. Se define el nombre del contenedor como "dj". El comando especificado es *python manage.py runserver 0.0.0.0:80*, que ejecuta el servidor en el puerto 80 dentro del contenedor. Se monta un volumen desde *./django* al directorio */code* dentro del contenedor. Se mapea el puerto 80 del contenedor al puerto 80 del *host*. También se definen variables de entorno para la conexión a la base de datos *PostgreSQL*.
- **ng**: un servicio que se construye a partir del *Dockerfile* definido en la carpeta *ionic*. Se define el nombre del contenedor como "ng" y se mapea el puerto 8080 del *host* al puerto 80 del contenedor.

De esta manera, tenemos todo el proyecto encapsulado en contenedores y orquestados de forma uniforme. Esto permite una mayor escalabilidad y flexibilidad. Permitiría realizar modificaciones en su estructura sin tener que reconstruirlo de nuevo como, por ejemplo, migrar el proyecto a otra máquina con mayores prestaciones, cambiar el gestor de la base de datos, cambiar el servidor web, dividir el proyecto en varias partes y ser ejecutado en máquinas distintas, entre otros.

Este entorno puede gestionarse con el comando *docker-compose* (Figura 17) o con la aplicación *Docker Desktop*.


















<input type="checkbox"/>	Name	Image	Status	Port(s)	Last started	Actions
<input type="checkbox"/>	 wordle	-	Exited		1 day ago	 
<input type="checkbox"/>	 ng fd2b9615aba6 	wordle-ng	Exited	8080:80 	3 days ago	 
<input type="checkbox"/>	 dj 42d69814e256 	wordle-dj	Exited	80:80 	1 day ago	 
<input type="checkbox"/>	 db-1 365d710864da 	postgres	Exited		1 day ago	 

Figura 17. Contenedores orquestados por *docker-compose*.

Respecto a GitHub³⁵, un proceso más detallado está redactado en el fichero *README.md* de la carpeta principal del proyecto. Puede consultarse en:

<https://github.com/davidcr01/WordlePlus/tree/main/Wordle%2B>

Este fichero podría considerarse un tutorial para construir toda la infraestructura desde cero.

III.3.1.2. Registro de usuarios

Backend

Esta tarea requiere los siguientes desarrollos en el *backend*:

- Definir los modelos para los usuarios y los jugadores.
- Definir los serializadores para dichos modelos.
- Definir las vistas para dichos serializadores.
- Crear la página web para el registro.

En *Django REST Framework*³³:

- **Modelos**: son clases que definen la estructura y comportamiento de los datos en la base de datos. Estas clases se utilizan para crear las tablas y realizar operaciones relacionadas con la persistencia de datos, como la creación, lectura, actualización y eliminación (CRUD). Los modelos en DRF³³ son similares a los modelos en Django, pero con funcionalidad adicional específica para la creación de API web. Se definen en el fichero *models.py*. Cuando se realizan los modelos, se hacen las migraciones ejecutando *python manage.py makemigrations*, creando automáticamente las nuevas tablas sin acceder directamente a la base de datos.
- **Serializadores**: son clases que se utilizan para convertir los objetos de Django en representaciones de datos que se pueden enviar a través de la red, como JSON. Los serializadores también se utilizan para convertir las representaciones de datos recibidas en objetos de Django. Se definen en el fichero *serializers.py*.
- **Vistas**: son clases o funciones que definen cómo se procesan las solicitudes de la API y cómo se devuelven las respuestas. Las vistas reciben las solicitudes HTTP (GET, POST, PUT, DELETE, etc.) y se encargan de realizar las operaciones correspondientes en los modelos y serializadores. Las vistas determinan qué datos se envían como respuesta y en qué formato. Se definen en el fichero *views.py*

Un ejemplo sencillo que muestre lo expuesto puede ser el siguiente:

Se define el modelo del Jugador (Figura 18), de tal manera que se especifica qué tipo de dato es cada atributo y qué propiedades tiene (como un valor por defecto). Algo a destacar es el atributo *user* que es de tipo *OneToOneField*. Este tipo de dato representa las relaciones 1:1. Por tanto, cada jugador estará estrictamente relacionado con un usuario.

```

class Player(models.Model):
    user = models.OneToOneField(CustomUser, on_delete=models.CASCADE,
related_name='player')
    wins = models.PositiveIntegerField(default=0)
    wins_pvp = models.PositiveIntegerField(default=0)
    wins_tournament = models.PositiveIntegerField(default=0)
    xp = models.PositiveIntegerField(default=0)

    def __str__(self):
        return self.user.username

```

Figura 18. Modelo del jugador.

Se define el serializador para el modelo *Player* (Figura 19). En este caso, se especifica que el atributo *user* utilizará el serializador definido para el modelo *CustomUser*, se define su modelo relacionado (*Player*) y los atributos que se deseen.

Además, se pueden redefinir ciertos métodos. En este caso, se ha modificado el comportamiento de la creación de nuevos jugadores sobrescribiendo el método *create*, que crea el jugador siempre y cuando se pase en el cuerpo de la petición los datos del usuario al que va a estar relacionado.

```

class PlayerSerializer(serializers.ModelSerializer):
    user = CustomUserSerializer()

    class Meta:
        model = Player
        fields = ('user', 'wins', 'wins_pvp', 'wins_tournament', 'xp')

    def create(self, validated_data):
        user_data = validated_data.pop('user', None)
        user = None

        if user_data:
            # Create the user only if user_data is provided
            user = CustomUser.objects.create_user(**user_data)

        player = Player.objects.create(user=user, **validated_data)
        return player

```

Figura 19. Serializador del jugador.

Posteriormente, se define la vista (Figura 20) que hará uso del serializador para los jugadores. En este caso, la petición a la base de datos será obtener todos los jugadores ordenados por su número de victorias. De la misma forma que en el serializador, se han sobrescrito ciertos métodos para modificar el comportamiento de la vista.

- En el caso del método *get_permissions*, se controla los permisos de los usuarios que utilicen dicha vista. La creación está disponible para todo el mundo; la modificación para los administradores; la destrucción para los administradores y el propietario, y el resto para usuarios autenticados.

De esta manera, por ejemplo, un jugador no podría modificar su número de victorias a través de la API.

Cabe destacar que *Django Rest Framework*³³ proporciona al desarrollador ciertos permisos, como *permissions.IsAuthenticated*, y también permite crear nuevos permisos, como es el caso de *IsAdminUser()* o *IsOwnerOrAdminPermission()*. Este punto es crucial para garantizar la seguridad de la API, ya que permitir su uso abierto puede comprometer los datos de los usuarios y la integridad del proyecto.

- En el caso del método *destroy*, describe cómo se va a destruir la información del jugador, eliminando también la información del usuario asociada para que no queden datos inconsistentes.

```
class PlayerViewSet(viewsets.ModelViewSet):
    """
    API endpoint that allows players to be viewed or edited.
    """
    queryset = Player.objects.all().order_by('wins')
    serializer_class = PlayerSerializer

    def get_permissions(self):
        """
        Overwrites the get_permissions method to include the validation
        of the is_staff field
        """
        if self.action == 'create':
            # Player creation is available to every user
            return []
        elif self.action in ['update', 'partial_update']:
            # Edition only for the owner or event managers.
            return [IsAdminUser()]
        elif self.action == 'destroy':
            # Destruction available for Admins and the owner
            return [IsOwnerOrAdminPermission()]
        else:
            # Authentication is needed for the rest of the operations.
            return [permissions.IsAuthenticated()]

    # It is necessary to override the destroy method to destroy the
    # related user
    def destroy(self, request, *args, **kwargs):
        instance = self.get_object()

        # Delete the related user
        user = instance.user
        user.delete()

        # Delete the player
        self.perform_destroy(instance)
        return Response(status=status.HTTP_204_NO_CONTENT)
```

Figura 20. Vista del jugador.

Por último, las vistas son habilitadas mediante rutas (Figura 21), definidas en el archivo *urls.py*. Por ejemplo, en el caso de los jugadores, se añade al router de la aplicación la ruta para los jugadores:

```
router = routers.DefaultRouter()
router.register(r'api/players', views.PlayerViewSet)
```

Figura 21. Rutas para el jugador.

La issue relacionada con GitHub³⁵ es [Customize the User and Players model \(S1\)](#). Además, en esta issue se realizaron otras tareas adicionales, como evitar que la contraseña pudiera actualizarse mediante la API, lo que supondría un problema severo de seguridad; y almacenar en la base de datos la contraseña encriptada para una mayor seguridad de los datos de los usuarios.

Con esto, la API *backend* proporcionará la ruta <http://localhost:80/api/players>, y procesará las peticiones dependiendo de los permisos definidos.

Por último, y sobre este aspecto, también se ha programado la posibilidad de usar el *Admin site* (Figura 22) de Django definida por el fichero *admin.py*. Esta interfaz de usuario es solamente accesible para los administradores, y permite una gestión modelos y sus datos de forma amigable para los gestores de la aplicación (Figura 23). Esta interesante opción de Django ahorra la necesidad de hacer otra aplicación exclusiva para los administradores.

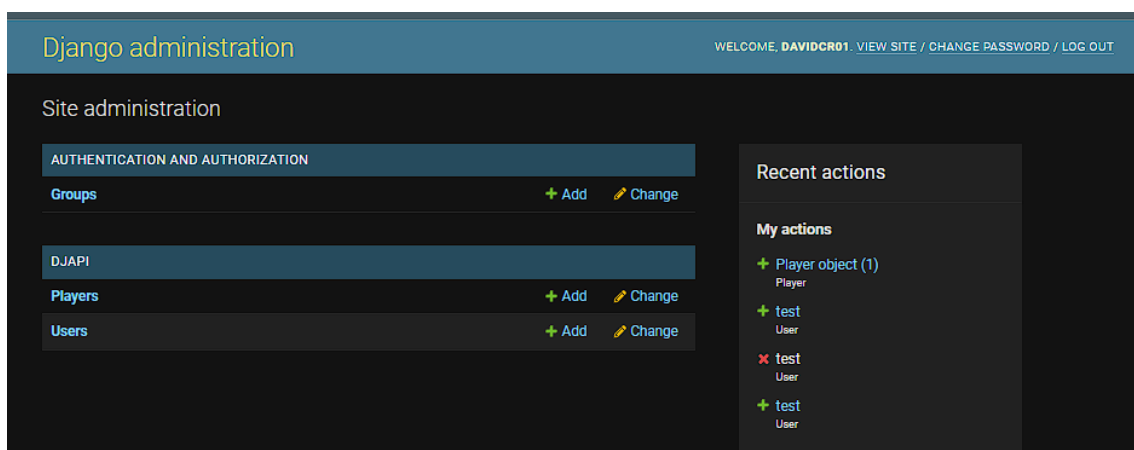


Figura 22. Admin Site.

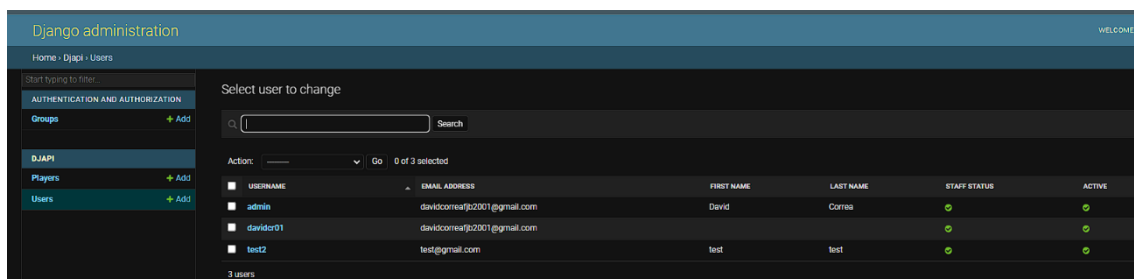


Figura 23. Admin site, usuarios almacenados.

Cabe destacar que este proyecto se ha configurado con autenticación con tokens para la seguridad de la API. De esta manera, toda interacción con la API deberá ser realizada con un token como autenticación, excepto la creación de nuevos usuarios.

Este desarrollo más en profundidad está plasmado en la issue de *GitHub*³⁵ [*Customize the Users authentication \(S1\)*](#).

Frontend

En cuanto al *frontend*, *Ionic Framework*²⁸ permite el uso de componentes. Los componentes son elementos reutilizables de código que encapsulan estructura, comportamiento y apariencia de una parte de la interfaz de usuario. Estos componentes se crean usando HTML5, CSS3 y TypeScript⁴⁰. El entorno proporciona componentes predefinidos y permite crear componentes personalizados.

En este Sprint, la única vista a implementar era el formulario de registro (Figura 24) de usuarios. El correspondiente boceto contiene el logo y el título en la parte superior, para recalcar el nombre de la plataforma; un color de fondo similar al color del logo, para establecer el tema principal de la aplicación; un cajón con todos los campos necesarios para el registro, para recalcar la información necesaria; y el botón de registro en el mismo color principal de la aplicación, resaltando entre el resto de los elementos.

Realizado en la herramienta *Figma*, queda de la siguiente manera:

El boceto muestra una interfaz de usuario para el registro en Wordle+. En la parte superior, hay un logo con la letra 'W' y el texto 'WORDLE+' a su derecha. Debajo, hay un formulario con los siguientes campos de entrada: 'username', 'email', 'first name', 'last name', 'password' y 'staff code (optional)'. Cada campo tiene un icono de ojo para alternar la visibilidad. Al final del formulario, hay un botón rectangular con el texto 'REGISTER'.

Figura 24. Boceto del formulario de registro

*Ionic Framework*²⁸ permite la generación de páginas de forma sencilla, con el comando *ionic generate page* se crean una serie de archivos y carpetas de forma automática, además de crear la nueva ruta de la aplicación:

- **nombre.module.ts:** Este archivo define un módulo específico para la página, que puede contener configuraciones de importación y

exportación, así como definiciones de rutas y otros elementos relacionados.

- **nombre.page.ts:** Este archivo contiene la lógica y la funcionalidad asociadas a la página, en donde se definen propiedades, métodos y eventos que se utilizan para controlar su comportamiento.
- **nombre.page.html:** Este archivo es la plantilla HTML de la página y define su estructura y los elementos de la interfaz de usuario. Es posible utilizar etiquetas HTML, componentes de Ionic y vinculación de datos.
- **nombre.page.scss:** Este archivo contiene estilos específicos de la página escritos en lenguaje de hojas de estilos (CSS o SCSS). Cabe destacar que el proyecto tiene un fichero *global.scss* en donde se añaden estilos que pueden usar todos los componentes.

Un ejemplo práctico de esto podría ser el siguiente:

En *register.page.html* (Figura 25) se define un formulario y un campo de dicho formulario de la siguiente manera

```
<form [formGroup]="registerForm" (ngSubmit)="onSubmit()">
  <ion-card-content>
    <ion-item>
      <ion-label position="floating">Username</ion-label>
      <ion-input type="text" formControlName="username" required></ion-input>
    </ion-item>
    <div *ngIf="registerForm.get('username')?.invalid &&
registerForm.get('username')?.touched" >
      Username is required. Min 4 characters.
    </div>
  </ion-card-content>
</form>
```

Figura 25. Documento HTML del formulario de registro.

Nótese que se usan etiquetas primitivas de HTML (como *form* o *div*) además de etiquetas propias del *framework*. Este trozo de código define un formulario de tipo *formGroup* llamado *registerForm* que ejecutará el método *onSubmit()* cuando el formulario se envíe. Esta información deberá corresponder con la lógica definida en el fichero *register.page.ts*.

Profundizando un poco más, el código muestra una pareja *label-input* para el campo *username* que es obligatorio y cuyo identificador en el formulario - controlado por el parámetro *formControlName*- es *username*. También se añade un *div* que se mostrará con la condición de que el valor del campo *username* sea inválido y el usuario haya modificado dicho campo. Esta condición es controlada por el parámetro *ngIf*.

Por otro lado, en el fichero *register.page.ts* (Figura 26) se define la lógica en este componente. Un aspecto interesante para recalcar de este ejemplo es la conexión con el *backend*.

Un aspecto para destacar en la creación de usuarios es que finalmente, tanto la creación de administradores como de jugadores se realizará en el mismo formulario expuesto anteriormente. Sin embargo, para distinguir ambos tipos de usuarios, el formulario pide un campo opcional *staff_code*, los cuales son códigos almacenados en el modelo *StaffCode*. Estos códigos solamente son creados por el superusuarios y proporcionados a los administradores para crear sus cuentas. Más información puede obtenerse en el siguiente enlace:

<https://github.com/davidcr01/WordlePlus/issues/5#issuecomment-1578536769>

```
const staffCode = this.registerForm.get('staff_code').value;

// Case of registering a player. The 'staff_code' field is not added
if (staffCode === '') {
  this.createUser('http://localhost/api/players/', { user: userData });
}
else { // Case of registering an admin. 'staff_code' field is added
  userData['staff_code'] = staffCode;
  this.createUser('http://localhost/api/users/', userData);
}
}
```

Figura 26. Lógica de creación de usuarios y administradores.

En este caso, es tan sencillo como comprobar si el valor del campo *is_staff* es vacío, si es el caso se creará un jugador con los datos proporcionados encapsulados en un objeto JSON *user*; en otro caso, se creará un administrador con la información proporcionada. La creación de usuarios está implementada en el método *createUser*, que hace una llamada POST a la *url* e informa al usuario de éxito o error.

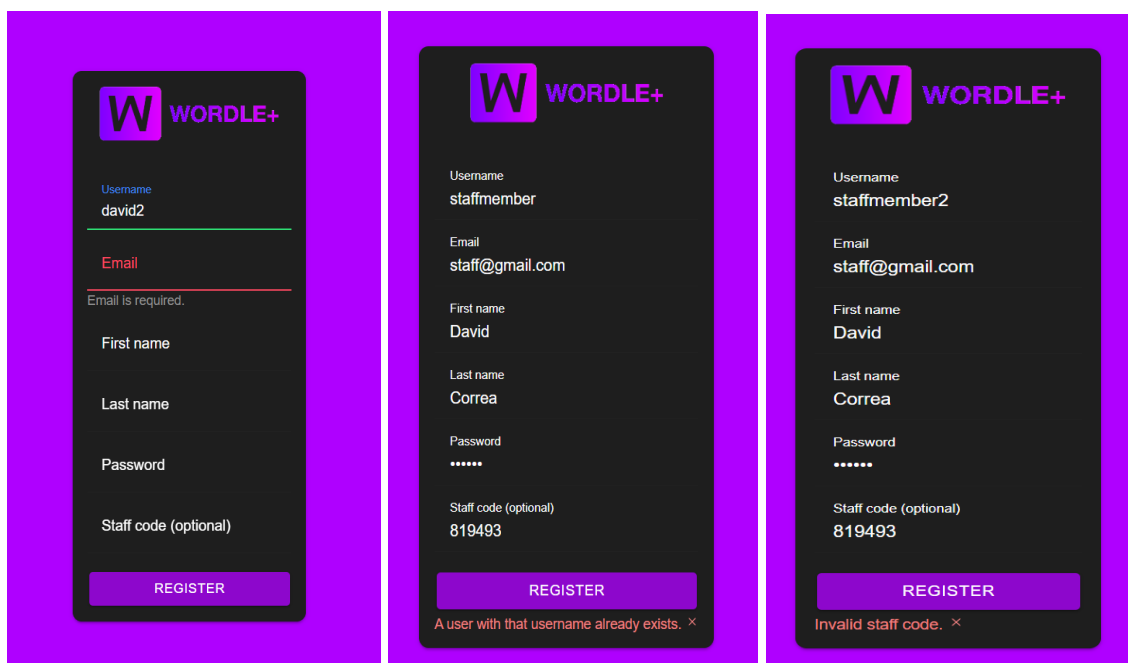
Este desarrollo se ha realizado en la issue de GitHub³⁵ [Register Page \(S1/S2\)](#). En dicha issue, se explica de forma más profunda el funcionamiento del formulario de registro y su lógica asociada.

Para finalizar, algunos desarrollos extras fueron realizados durante este Sprint. Estos desarrollos son mejoras y bugs reportados:

- [Staff members should not manage superuser account \(S1\)](#): protección de la cuenta del superusuario. No es modificable por otros administradores.
- [Administrators can not see anything in the Admin site \(S1\)](#): los administradores no tenían acceso a la vista *Admin* de Django³³. Supuso crear un nuevo grupo *Staff* con los permisos necesarios y programar la lógica de asignación de dicho grupo a los nuevos administradores creados.
- [Hide users information when a player list other players \(S1\)](#): como jugador se podía obtener toda la información de los usuarios. Supuso crear un nuevo serializador y vistas para filtrar la información cuando era obtenida por los jugadores.

III.3.1.3. Interfaz de usuario

En cuanto a las pruebas, se comprueba que el formulario contempla los posibles casos de error (Figuras 27, 28 y 29):

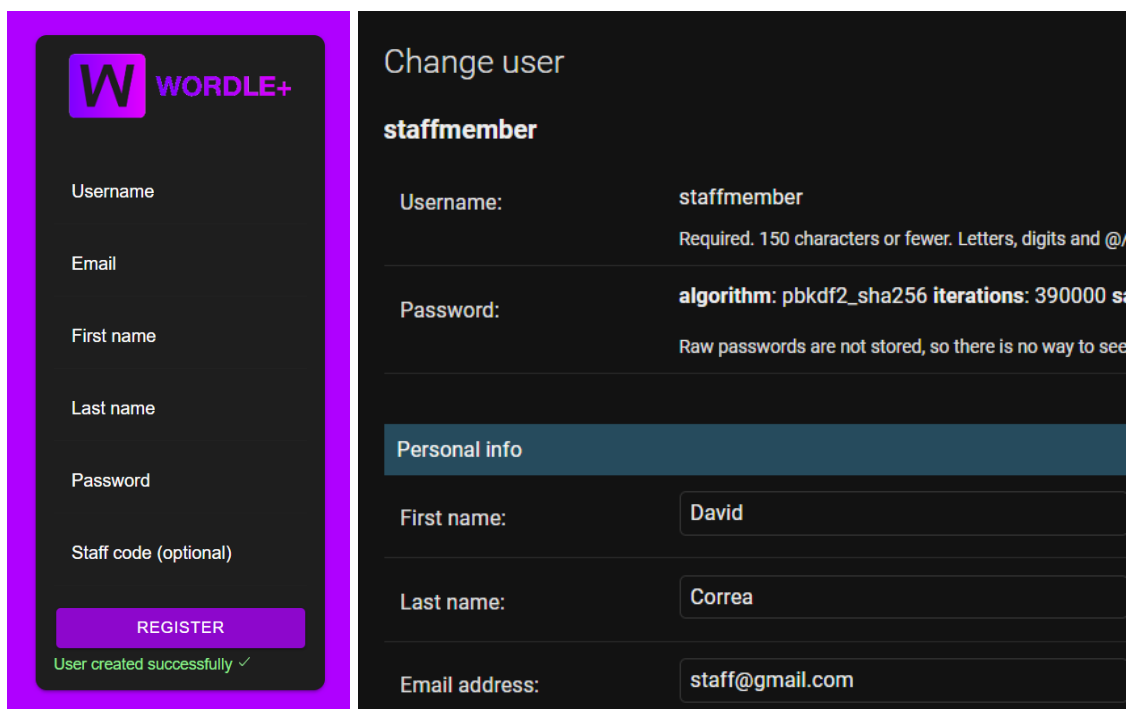


The figure consists of three side-by-side screenshots of the Wordle+ registration form, each showing a different error state. All three forms have a dark blue header with the Wordle+ logo. The form fields are: Username, Email, First name, Last name, Password, and Staff code (optional). A red 'REGISTER' button is at the bottom.

- Figure 27:** The 'Email' field is highlighted in red, and the text 'Email is required.' is displayed below it.
- Figure 28:** The 'Username' field is highlighted in red, and the text 'A user with that username already exists. ✕' is displayed at the bottom.
- Figure 29:** The 'Staff code' field is highlighted in red, and the text 'Invalid staff code. ✕' is displayed at the bottom.

Figuras 27, 28 y 29. Formulario de inicio de sesión. Casos de error.

Y, en caso de que el formulario sea correcto y cumpla las restricciones de la base de datos, el usuario se creará correctamente (Figuras 30 y 31):



The figure consists of two side-by-side screenshots of the Wordle+ interface. The left screenshot shows the registration form with a green 'REGISTER' button and a green message 'User created successfully ✓' at the bottom. The right screenshot shows the 'Change user' page for the 'staffmember' user, displaying their details and a 'Personal info' section.

Left Screenshot (Figura 30): The registration form is shown with the 'REGISTER' button highlighted in green. Below the button, a green message reads 'User created successfully ✓'.

Right Screenshot (Figura 31): The 'Change user' page for the 'staffmember' user is shown. It displays the following information:

- Username:** staffmember
- Password:** algorithm: pbkdf2_sha256 iterations: 390000 salt: ...
- Personal info:**
 - First name:** David
 - Last name:** Correa
 - Email address:** staff@gmail.com

Figuras 30 y 31. Caso de éxito y comprobación en Admin.

III.3.1.4. Revisión y retrospectiva

Como **revisión** del Sprint, los elementos del *backlog* que se plantearon en este Sprint se han cumplido correctamente, siguiendo los suficientes criterios de seguridad y calidad. Además, se han realizado otras tareas de otros Sprint que resultaron estar estrechamente relacionadas con el primer Sprint, debido al funcionamiento de las tecnologías. El resultado del incremento es satisfactorio, habiendo realizado un incremento atractivo y funcional. Sin embargo, este incremento se presenta incompleto para ser incorporado al producto final.

Como **retrospectiva** del Sprint, éste ha durado más tiempo de lo esperado. Es cierto que se han implementado ciertas funcionalidades que estaban planteadas para el futuro, y que ha sido el primer contacto con las herramientas, por lo que el proceso de aprendizaje ha sido lento.

Como aspecto a modificar, se debería de practicar más frente a estudiar y analizar la tarea a realizar de forma exhaustiva. Esto retrasa el progreso del Sprint y se daría con una solución más rápidamente si se hicieran pruebas de la solución pensada frente a sobre analizarla demasiado.

También se observa bastante documentación en el repositorio de GitHub, lo que también retrasa la creación de nuevo código. Se debe de buscar un equilibrio entre ambos como sugiere la filosofía Scrum.

III.3.2. Sprint 2

Nº	Objetivo	Fecha de entrega	
2	Registro de administrador. Inicio y cierre de sesión	27 de febrero del 2023	
Ident.	Título	Estimación	Prioridad
HU.2	Como administrador quiero poder registrarme	2	1
HU.3	Como usuario (jugador y administrador) quiero poder iniciar sesión	2	1
HU.4	Como usuario quiero poder cerrar sesión	1	1

El objetivo de este sprint es completar el registro de usuarios, en este caso de los administradores, y desarrollar el acceso a la plataforma mediante el inicio de sesión, además del cierre de sesión.

Como ya se mencionó en el Sprint anterior, la HU.2 se desarrolló de forma concurrente con la HU.1. Más detalles del registro de administradores se encuentra en la issue [Register Page \(S1/S2\)](#). El desarrollo relacionado es la creación y gestión del modelo *StaffCode*, así como la lógica del formulario de registro controlando el campo *staff_code*.

III.3.2.1. Inicio y cierre de sesión

En este Sprint, la única vista a implementar era el formulario de inicio de sesión de usuarios (Figura 32). Contiene, al igual que el formulario de registro, el logo y título de la plataforma recalcando su importancia; una caja con los campos necesarios, además del botón de inicio de sesión del mismo color de fondo para destacarlo, y un botón de registro en un color menos destacado, al no tener tanta relevancia en este formulario. El correspondiente boceto, realizado en la herramienta Figma, queda de la siguiente manera:



Figura 32. Boceto de inicio de sesión.

El formulario contendrá los campos necesarios para acceder a la plataforma, un botón de registro para redirigir al formulario de creación de usuarios, y un icono en la esquina inferior derecha que redirigirá al formulario de inicio de sesión del sitio *Admin* de *DRF*³³ para los administradores.

La maquetación HTML y la validación del formulario de inicio de sesión es muy similar al formulario de registro. El desarrollo y más detalles al respecto se encuentran en la issue [Login page \(S2\)](#).

Un aspecto importante a resaltar es la función *login()* (Figura 33) que se ejecuta al pulsar el botón de inicio de sesión:


```

login() {
  const credentials = {
    username: this.loginForm.get('username').value,
    password: this.loginForm.get('password').value,
  }

  this.http.post<any>('http://localhost:8080/api-token-auth/',
credentials).subscribe(
  async (response) => {
    // Store the token in the local storage
    this.errorMessage = ''
    const encryptedToken =
this.encryptionService.encryptData(response.token);
    await this.storageService.setAccessToken(encryptedToken);

    this.router.navigateByUrl('');
  },
  (error) => {
    console.error('Log in error', error);
    this.errorMessage = 'Provided credentials are not correct'
  }
);
}

```

Figura 33. Función *login()* del formulario de inicio de sesión.

En este caso, se obtienen las credenciales introducidas y se hace una llamada a la API con la URL *api-token-auth*. Esta ruta es proporcionada por el propio DRF³³, y dado un *username* y un *password*, devuelve un token de acceso si dicho usuario existe. Si no existe, un error se mostrará en el formulario, en otro caso, este token se encripta y se almacena en un almacenamiento local de la plataforma. Respecto a esto último, se explicará más detalle a continuación.

Seguridad y almacenamiento

Al haber desarrollado una API REST con autenticación por tokens, es necesario que su creación y gestión sean lo suficientemente seguras para garantizar la seguridad e integridad de la plataforma.

DRF³³ proporciona una generación de token lo suficientemente aleatorios seguros y no es necesario alterar la generación de éstos. Sin embargo, es necesario configurar una expiración de tokens para que estos se renueven cada cierto tiempo. Para ello, se desarrolló la *issue* [Improve Token generation \(S2\)](#) que define un nuevo *middleware* que comprueba si el token ha expirado, y en caso afirmativo lo elimina. Los *middlewares* en DRF³³ se ejecutan cuando se realiza una petición HTTP, antes de ejecutar la correspondiente vista.

Sin embargo, quizá el aspecto más importante de los tokens es su gestión y administración. Para asegurar esto, dos nuevos servicios han sido creados en el *frontend*:

- **Servicio de almacenamiento** (*storage.service.ts*): Proporciona un almacenamiento seguro inicializado y define algunos métodos para almacenar el token. Este almacenamiento es útil para guardar cierta información sobre los usuarios. Hace uso del módulo [IonicStorage](#).
- **Servicio de encriptación** (*encryption.service.ts*): para información sensible, como el token de acceso, no es suficiente almacenarlo, ya que puede ser interceptado mediante ataques *Man-In-The-Middle*. Por tanto, este servicio permite cifrar la información necesaria mediante el algoritmo AES 256 usando una clave aleatoria generada con el mismo algoritmo. Respecto a la clave:
 - o No se ha introducido en texto plano en el código proporcionado por el servicio web. Esto supone un riesgo de seguridad ya que el código sería consultable a través del navegador.
 - o La clave de cifrado se encuentra en un fichero por separado, y es obtenida leyendo dicho fichero. Este archivo, *env.json* no es proporcionado por el servidor web, por lo que es inaccesible a través del navegador. Además, se ha añadido al fichero. *gitignore* para evitar subirlo al repositorio.

Más detalles acerca de la implementación de los servicios se encuentra en el siguiente enlace:
<https://github.com/davidcr01/WordlePlus/issues/12#issuecomment-1595331890>

De esta manera, la información sensible y susceptible a ataques es cifrada y almacenada. Existe una alternativa para empresas denominada [SecureStorage](#), compatible con el módulo ya utilizado *Ionic Storage*, que proporciona un almacenamiento de alto rendimiento y seguro, utilizado por aplicaciones de contenido altamente sensible.

Es una buena solución para un entorno de producción. Sin embargo, teniendo en cuenta que en este entorno de desarrollo se priorizan el uso de tecnologías gratuitas y accesibles, y que la plataforma no maneja información altamente sensible, se ha optado por el desarrollo de los servicios mencionados anteriormente.

Cierre de sesión y página de redirección

Respecto al cierre de sesión, se ha creado un nuevo componente *app-logout-button* que puede ser importado y usado por cualquier página de la aplicación. La lógica del componente sencillamente elimina el token de acceso almacenado en *Ionic Storage* y redirige al usuario a la página de inicio de sesión. Más información puede ser consultada en la *issue* [Logout \(S2\)](#)

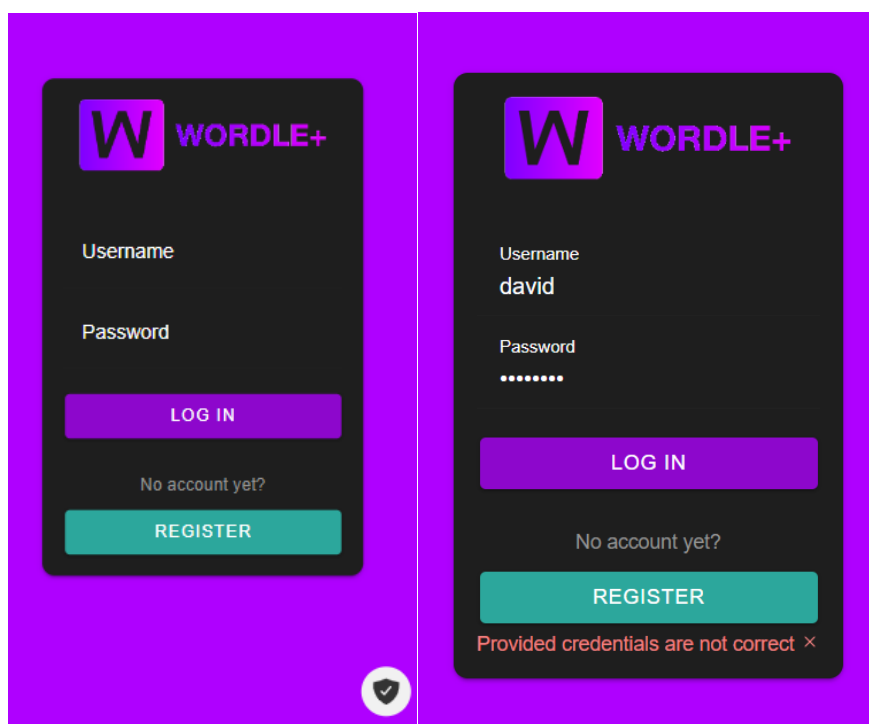
Además, se ha creado una nueva página denominada *home* que se encarga de redirigir al usuario cuando accede a la plataforma dependiendo de si tiene un token almacenado. Es similar a una pantalla de carga, y evita proporcionar al

usuario *feedback* de que la información está cargando y que la aplicación está operativa.

En caso afirmativo, se obvia la página de inicio de sesión y el usuario accede directamente a la plataforma. En otro caso, se redirecciona a la página de acceso. Más información está disponible en el siguiente enlace: <https://github.com/davidcr01/WordlePlus/issues/12#issuecomment-1596163741>

III.3.2.2. Interfaz de usuario

El resultado de las vistas han sido las siguientes (Figuras 34 y 35):



Figuras 34 y 35. Formulario de inicio de sesión. Caso de error.

La validación del formulario se realiza correctamente, y el usuario accede a la plataforma si las credenciales son correctas. Se comprueba que su *token* de acceso es almacenado correctamente y encriptado (Figura 36):

Entradas totales: 1		
#	Clave	Valor
0	"access_token"	"U2FsdGVkX1+cr1R05shNoUSy2G44t8cwg9RtzcmIYnJJmeV857B1t!"

Figura 36. Token encriptado y almacenado.

Cuando se pulsa el botón de *logout* se comprueba que el token es eliminado del almacenamiento correctamente (Figura 37).

Entradas totales: 0		
#	Clave	Valor

Figura 37. Token eliminado.

III.3.2.3. Revisión y retrospectiva

Como **revisión** del Sprint, los elementos del *backlog* que se plantearon en este Sprint se han cumplido correctamente, siguiendo los suficientes criterios de seguridad y calidad. Además, se ha realizado un análisis, evaluación e implementación de las necesidades de integridad de la plataforma, proporcionando las garantías de seguridad para los clientes. El resultado del incremento es satisfactorio, habiendo realizado un incremento atractivo y funcional. En este caso, el incremento junto el del anterior Sprint se incorporará a la aplicación final (Release v1.0.0).

Como **retrospectiva** del Sprint, éste ha durado menos tiempo de lo planificado, habiendo sido mucho más eficientes en el desarrollo. La implementación breve pero efectiva de este Sprint ha compensado en cierta medida la excesiva longitud del anterior. En este caso, se aprecia un mejor manejo de las tecnologías y una resolución de errores más eficaz.

Además, se ha equilibrado la documentación en el repositorio de GitHub³⁵, pero sin dejar de profundizar en los detalles e implementación desarrollados. Estas buenas prácticas deben de seguir realizándose en los siguientes Sprints.

III.3.3. Sprint 3

Nº	Objetivo	Fecha de entrega	
3	Wordle clásico	13 de marzo del 2023	
Ident.	Título	Estimación	Prioridad
HU.5	Como jugador quiero poder jugar un Wordle clásico	8	1

El objetivo de este Sprint es construir y desarrollar la esencia del juego, y de donde se ramifican el resto de las funcionalidades: el clásico juego de Wordle. Al ser una HU con una estimación considerable, podría clasificarse como una épica, al no poder dividirse en otras historias de usuario independientes. En este caso, al ser la interfaz de usuario lo más similar al juego original, no se ha

implementado un boceto, teniendo en cuenta que la intención es hacer un diseño muy similar para conservar la esencia del juego.

Aunque la historia de usuario se basa en el juego existente, cabe destacar que en juego original no existe registro de jugadores, por lo que esta historia de usuario también abarca el registro de las partidas asociadas a los jugadores.

III.3.3.2. Registro de partidas

Para registrar las partidas jugadas por los diferentes usuarios, es necesario añadir un nuevo modelo, serializador, vista y ruta. En el caso del modelo, este almacenará: una referencia al jugador, la palabra a adivinar, el tiempo e intentos consumidos, la experiencia ganada, la fecha de juego y si ha ganado el juego o no.

Respecto al serializador, este redefine el método *create()* para modificar su comportamiento cuando se cree nueva información. En este caso, el método incrementa en 1 las victorias del jugador si ha ganado, además de la experiencia conseguida. De esta forma, se evita tener que realizar otra petición PATCH para modificar la información relacionada con el jugador (Figura 38).

```
class ClassicWordleSerializer(serializers.ModelSerializer):
    class Meta:
        model = ClassicWordle
        fields = ['word', 'time_consumed', 'attempts', 'xp_gained',
' date_played', 'win']

    def create(self, validated_data):
        player = validated_data['player']
        is_winner = validated_data['win']
        xp = validated_data['xp_gained']

        # Increment the number of victories and add the xp_gained
        if is_winner:
            player.wins += 1
            player.xp += xp
            player.save()

        return ClassicWordle.objects.create(**validated_data)
```

Figura 38. Serializador de *ClassicWordle*.

En el caso de la vista, como solo es necesario definir los métodos para las solicitudes GET y POST, el serializador heredará de la clase *GenericViewSet* y definirá los métodos *list* y *create*. Para evitar pasar el identificador de usuario por parámetros, el jugador se obtiene comprobando el valor de *request.user*, relacionado con el token de acceso (Figura 39). Por último, se habilita la nueva

ruta añadiéndola al conjunto de rutas. Un ejemplo de uso de la nueva ruta podría ser (Figura 40):

```
class ClassicWordleViewSet(viewsets.GenericViewSet):
    permission_classes = [IsOwnerOrAdminPermission]
    queryset = ClassicWordle.objects.all()
    serializer_class = ClassicWordleSerializer

    def list(self, request):
        player = getattr(request.user, 'player', None)
        if not player:
            return Response({'error': 'Player not found'}, status=404)

        queryset =
ClassicWordle.objects.filter(player=player).order_by('-date_played')
        serializer = ClassicWordleSerializer(queryset, many=True)
        return Response(serializer.data)
```

Figura 39. Método *list* de la vista de *ClassicWordle*.

```
POST http://localhost:8080/api/classicwordles/
BODY:
{
  "word": "apple",
  "win": true,
  "time_consumed": 120,
  "attempts": 4,
  "xp_gained": 50
}

RESPONSE:
{
  "word": "apple",
  "time_consumed": 120,
  "attempts": 4,
  "xp_gained": 50,
  "date_played": "2023-06-26T20:19:13.732827+02:00",
  "win": true
}
```

Figura 40. Funcionamiento de la ruta *classicwordles*.

Respecto a la petición GET (asociado al método *list*) será útil para implementar el histórico de partidas del jugador.

Nótese que la fecha de la partida *date_played* se rellena automáticamente, por lo que no es necesario obtenerla en el *frontend*.

Más detalles de implementación pueden encontrarse en: <https://github.com/davidcr01/WordlePlus/issues/22#issuecomment-1608003716>

III.3.3.2. Maquetación HTML

Esta funcionalidad ha sido desarrollada como un componente, al igual que se hizo con el botón de cierre de sesión, para reutilizar el código del juego Wordle y no encasillarlo solamente en una página.

La estructura HTML del componente, en el fichero *wordle-dashboard.component.html*, se divide en dos partes, al igual que el juego original (Figura 41):

- La matriz de cajas: donde el usuario introduce las palabras, y las letras de dichas palabras se encasillan en cajas.
- El teclado virtual: para estandarizar el teclado y abstraerse de los teclados de los dispositivos, el juego ofrece un teclado en pantalla

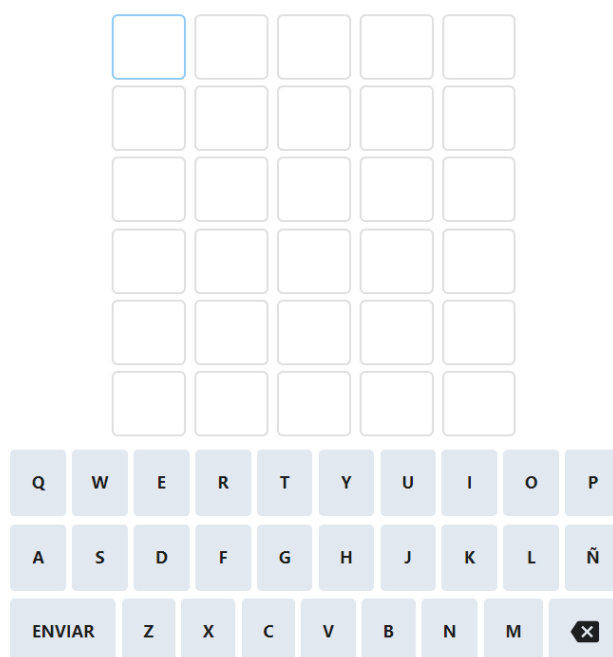


Figura 41. Interfaz del juego original.

En el caso de la matriz de cajas, *Ionic*²⁸ permite la construcción de código HTML en bucles, por lo que esta matriz se construye con dos bucles anidados, donde previamente se inicializa el número de filas y el tamaño de cada fila (Figura 42).

```
<ion-col size="9" class="game-board">
  <ion-row *ngFor="let row of letterRows">
    <ion-col *ngFor="let box of row" class="letter-box" [class.filled-box]="box.filled">
      {{ box.content }}
    </ion-col>
  </ion-row>
</ion-col>
```

Figura 42. Bucle de la matriz de cajas.

En el caso del teclado virtual, se construye de forma similar: se definen las listas de teclado de forma independiente y se crea una fila de teclas iterando sobre dichas listas, excepto los botones de borrado y *enter* que se definen de forma independiente. Cabe destacar que, aunque se proporcione un teclado virtual, el componente permite el uso de un teclado físico, en el caso de ordenadores o portátiles.

Más detalles pueden encontrarse en el siguiente enlace, en el apartado *HTML structure*: <https://github.com/davidcr01/WordlePlus/issues/22#issuecomment-1606055731>

III.3.3.3. Lógica del juego

La lógica del juego se define en el archivo *wodle-component.page.ts*. En este fichero se definen diversos métodos para su correcto funcionamiento y conexión con la base de datos. Los más importantes son:

- *generateWord()*: genera una palabra aleatoria dada su longitud. Esta palabra se obtiene de un fichero oculto *words.json* (no proporcionado por el servidor web ni subido al repositorio) que contiene todas las palabras definidas. Se eligió definir estas palabras en el *frontend* para no depender de una API externa que genere palabras aleatorias y para no sobrecargar al *backend* con peticiones de palabras.
- *initGame()*: inicializa variables importantes, como la matriz de cajas de la interfaz de usuario y las letras del teclado.
- *deleteLetter()*: elimina la última letra que se introdujo. Esto implica marcar la casilla actual como no rellenada, quitar la letra de la cadena que está construyendo el usuario y eliminar la letra de la matriz de cajas.
- *handleKeyboardButtonClick()*: maneja la letra introducida por el usuario, ya sea haciendo clic en el teclado virtual o el físico (este último también necesitará la función *handleKeyboardKeyUp*). Se encarga de los caracteres normales, teclas especiales (de introducción y borrado), y de asegurar los límites de la matriz, es decir, que el usuario no introduzca ni borre más letras de las permitidas.
- *handleKeyboardKeyUp()*: escucha un evento de teclado, obtiene la tecla presionada y envía dicha tecla a la función *handleKeyboardButtonClick*. Por ello, esta última función centraliza el uso de ambos teclados.
- *checkGuess()*: comprueba la palabra introducida cuando el usuario pulsa la tecla *Enter*. Comprueba si la palabra tiene la longitud suficiente y si se encuentra en la lista de palabras disponible (esto evita que el usuario introduzca letras aleatorias). Por último y a través de una animación, colorea cada caja dependiendo de si las letras introducidas se encuentran o no en la palabra.

- *handleEndgame()*: se encarga de gestionar el fin del juego si el usuario ha perdido o ganado. En ambos casos, se calcula el tiempo y los intentos consumidos, además de la experiencia, que depende del resto de estadísticas. Finalmente, se construye el cuerpo de la petición y se hace una llamada a la API explicada anteriormente.

Un ejemplo ilustrativo de este código puede ser la función *handleKeyboardButtonClick*, una de las más importantes del componente (Figura 43):

```
public handleKeyboardButtonClick(letter: string): void {
  if (letter === 'delete' || letter === 'backspace') {
    this.deleteLetter();
  } else if (letter === 'enter') {
    this.checkGuess();
  } else { // Controlling max number of letters in input
    if (this.currentGuess.length >= this.WORDS_LENGTH) {
      this.toastService.showToast('Max letters reached!', 2000, 'top');
      return;
    }
    const currentRow = this.MAX_GUESSES - this.guessesRemaining;
    const boxIndex = this.nextLetter;

    // Access to legal position of the matrix
    if (currentRow >= 0 && currentRow < this.letterRows.length &&
    boxIndex >= 0 && boxIndex < this.letterRows[currentRow].length) {
      const box = this.letterRows[currentRow][boxIndex];
      box.content = letter;
      box.filled = true;
    }

    this.currentGuess.push(letter);
    this.nextLetter++;
  }
}
```

Figura 43. Función *handleKeyboardButtonClick*.

Cabe destacar que el componente es modificable dinámicamente especificando un parámetro, *WORDS_LENGTH*, que define la longitud de la palabra. De esta manera, tanto la palabra escogida como la interfaz de usuario cambian. Por ejemplo, para definir un Wordle de 6 letras (Figura 44):

```
<ion-content>
  <app-wordle-dashboard [WORDS_LENGTH]="6"></app-wordle-dashboard>
</ion-content>
```

Figura 44. Componente Wordle de 6 letras.

Más información está disponible en el siguiente enlace, en el apartado *TS logic*: <https://github.com/davidcr01/WordlePlus/issues/22#issuecomment-1606055731>

Como **desarrollo adicional**, se ha añadido un guardián de autenticación. Hasta ahora, el usuario era redirigido a la aplicación principal a través del *login*, pero eso no garantizaba el no-acceso a la plataforma indicando explícitamente la URL sin iniciar sesión.

Para garantizar la protección de las vistas de la aplicación, se ha definido el fichero *auth.guard.ts*, que se encarga de la validez del token de acceso:

- Si no hay almacenado un token de acceso, el usuario es redireccionado a la página de inicio de sesión.
- Si lo hay, se comprueba si el token ha expirado mediante una nueva llamada a la API. Esta nueva vista en el *backend* es muy similar al *middleware* que ya se definió para los tokens, pero son conceptos distintos. El *middleware* verifica el token cuando se usa la API, pero el guardián verifica el token cuando accede a diferentes URL de la aplicación en el *frontend*.

Con ello, para proteger las vistas simplemente hay que añadir un parámetro a las rutas definidas en el fichero *app-routing.module.ts*. Por ejemplo (Figura 45):

```
{
  path: 'classic-wordle',
  loadChildren: () => import('./pages/classic-wordle/classic-
wordle.module').then( m => m.ClassicWordlePageModule),
  canActivate: [AuthGuard]
},
```

Figura 45. Protegiendo la vista *classic-wordle* usando *canActivate*.

Más información detallada se encuentra en el siguiente enlace, en el apartado *AuthGuard*: <https://github.com/davidcr01/WordlePlus/issues/22#issuecomment-1609080466>

III.3.3.4. Interfaz de usuario

Finalmente, el panel de juego, para palabras de 5 letras, queda de la siguiente manera (Figura 46 y 47):

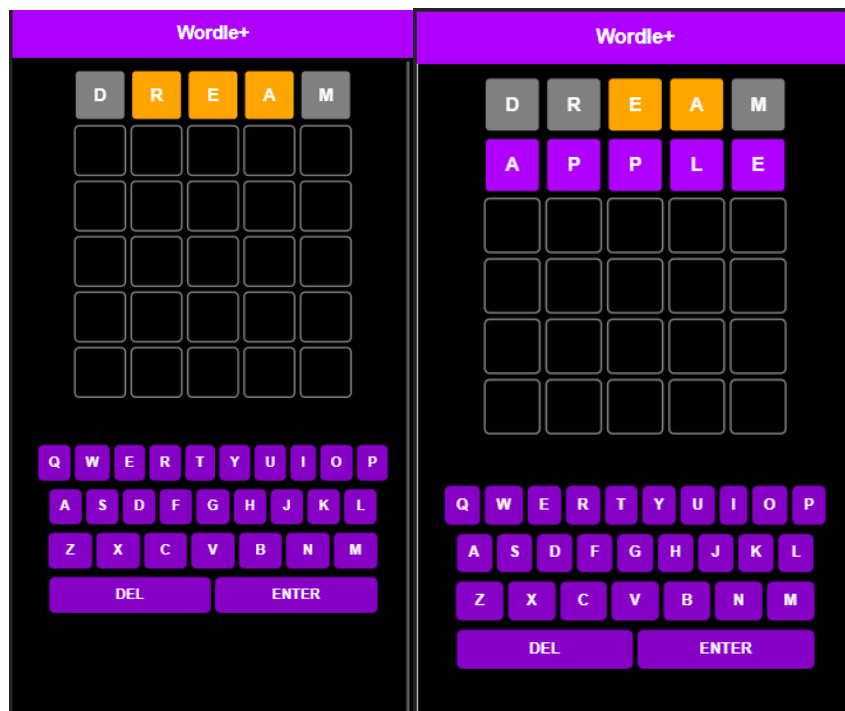


Figura 46 y 47. Panel Wordle de 5 letras.

Tanto la matriz de letras como el teclado se redimensionan según el tamaño de la pantalla y del tamaño de la letra. Por ejemplo, en una *NestHub* en horizontal con una palabra de 7 letras (Figura 48):

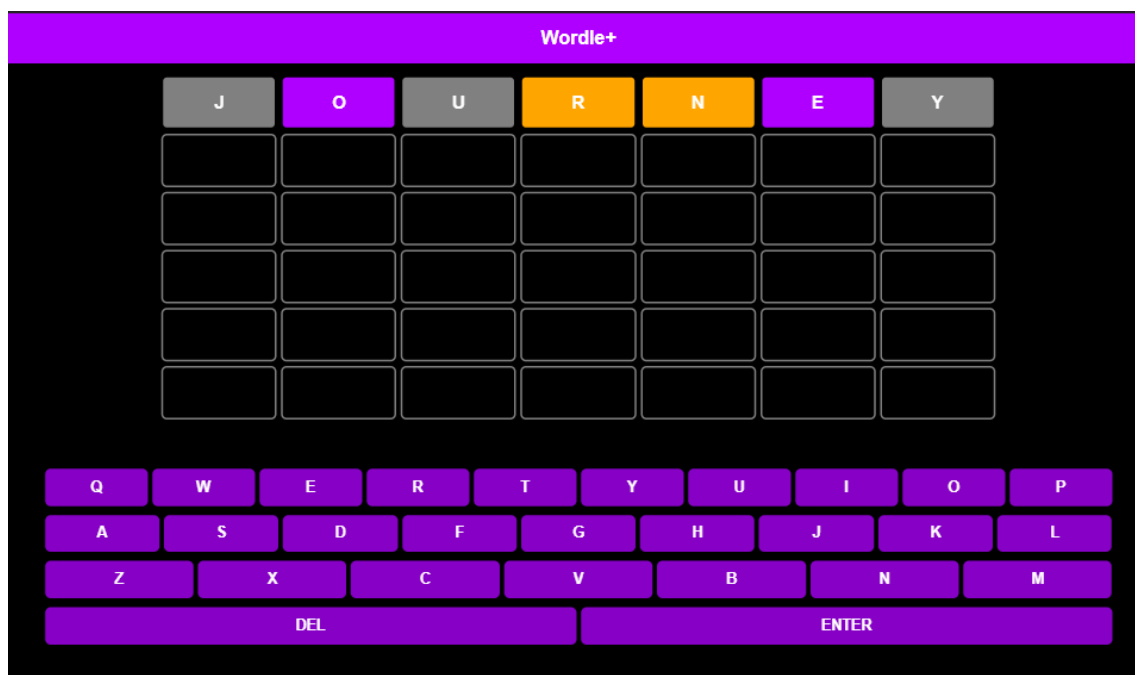


Figura 48. Wordle de 7 letras en horizontal.

III.3.3.5. Revisión y retrospectiva

Como **revisión** del Sprint, la épica del *backlog* que se planteó en este sprint se ha cumplido correctamente, siguiendo los suficientes criterios de seguridad y calidad. Además, se ha integrado la interfaz del juego con una correcta conexión a la base de datos, registrando las partidas del jugador. Esto también será útil cuando se implemente el registro de partidas. El resultado del incremento es satisfactorio, habiendo realizado un incremento atractivo y funcional. En este caso, el incremento se incorporará a la aplicación final (Release v1.1.0).

Como **retrospectiva** del Sprint, éste ha durado menos tiempo de lo planificado, habiendo sido mucho más eficientes en el desarrollo. El desarrollo de la interfaz ha presentado ciertas complicaciones, sobre todo en el diseño adaptable a la resolución de la pantalla.

Por otro lado, se ha implementado el mecanismo de seguridad *AuthGuard* en este Sprint, desarrollo que no estaba planificado y quizás hubiera sido conveniente implementarlo en el anterior Sprint. Es necesario evaluar y analizar todos los requisitos de las historias de usuario a implementar antes de realizar una implementación definitiva.

III.3.4. Sprint 4

Nº	Objetivo	Fecha de entrega	
4	Ampliación Wordle clásico. Modificación datos usuarios	27 de marzo del 2023	
Ident.	Título	Estimación	Prioridad
HU.6	Como jugador quiero poder seleccionar la longitud de la palabra de la partida clásica	2	2
HU.14	Como usuario quiero poder modificar mis datos personales	2	3
HU.25	Como jugador quiero ver las notificaciones a través del buzón	2	2

El objetivo de este Sprint es proporcionarle al jugador diversas funcionalidades e interacciones directamente con la plataforma, como personalizar la partida clásica, modificar su información personal y ver sus notificaciones. Para que el usuario pueda acceder cómodamente a dichas funcionalidades, es necesario

implementar una de las pestañas de la aplicación, en este caso *Main*, que será la pantalla principal de esta aplicación.

El boceto de esta vista, desarrollado con la herramienta Figma, es el siguiente (Figura 49):

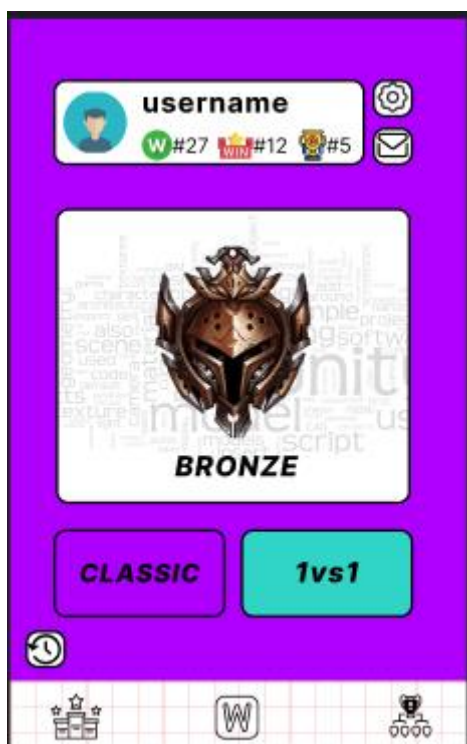


Figura 49. Boceto de la página principal

Respecto a este boceto, se ha especificado el color de fondo como el color principal de la aplicación, se añade una barra de información de jugador; a su derecha, un par de iconos para redirigir a los ajustes y al buzón; debajo del *banner* se muestra una carta con la imagen y nombre del rango del jugador; en la parte inferior se proporcionarán dos botones para jugar a la partida clásica y multijugador; y en la esquina inferior izquierda un icono para ver el histórico de partidas. El objetivo es construir una página limpia, que muestre la información necesaria, pero sin sobrecargarla demasiado.

III.3.4.1. Página principal (Main)

Al ser la vista a implementar bastante específica y personalizada, se han utilizado principalmente elementos HTML básicos para su implementación. Por ello, un aspecto crucial es que los elementos de la vista sean totalmente adaptables al tamaño de la pantalla del dispositivo. Para lograr esto, se han utilizado en su mayoría *flexbox* para organizar la disposición de elementos, y unidades de tamaño variables, como *vh* (alto de la pantalla), *vw* (ancho de la pantalla) y porcentajes (*respecto al contenedor padre*).

La página HTML se divide en:

- Zona de información del jugador: corresponde al banner superior, en donde también se encuentran un par de botones. Es necesario definir las variables correspondientes a la información del jugador en la lógica de la página (Figura 50).
- Carta del rango: donde se muestra la imagen del rango del jugador, su título y una imagen de fondo. La imagen de fondo cambia dependiendo del ancho de la pantalla para no deformarla demasiado.
- Botones de juego: contiene los botones para jugar al juego clásico y al multijugador.

```
<div class="victory">
  
  <span>{{ victoriesClassic }}</span>
</div>
<div class="victory">
  
  <span>{{ victoriesPvp }}</span>
</div>
```

Figura 50. Mostrando información del jugador.

Para almacenar esta información en primera instancia, se ha modificado la función *login* de la página de inicio de sesión, para que almacene esta información en el servicio *StorageService* (número de victorias, experiencia, identificador del usuario, etc). Siempre que se cargue la página principal, se recuperan estos valores del servicio. Esto se consigue con la función *ionViewWillEnter* que se ejecuta cada vez que la vista va a ser cargada (Figura 51):

```
async ionViewWillEnter() {
  this.username = await this.storageService.getUsername();
  this.victoriesClassic = await this.storageService.getWins();
  this.victoriesPvp = await this.storageService.getWinsPVP();
  this.victoriesTournaments = await
this.storageService.getWinsTournament();
  this.xP = await this.storageService.getXP();
  this.rank = await this.storageService.getRank();
  this.rankImage = await this.getRankImage(this.rank);
}
```

Figura 51. Función *ionViewWillEnter*.

Más información acerca de este desarrollo puede encontrarse en: <https://github.com/davidcr01/WordlePlus/issues/23#issuecomment-1614654829>

Un aspecto importante para recalcar en este apartado es el avatar del jugador. La idea es almacenar los avatares de todos los jugadores en el *backend*, pero el *frontend* necesita el avatar del jugador que ha iniciado sesión.

Por tanto, la metodología que se ha desarrollado ha sido la siguiente:

- Se ha definido una nueva vista en el *backend* para obtener y guardar imágenes de avatares. (Figura 52).
- En el frontend, la imagen se guarda de forma temporal en el StorageService. De esta manera, el avatar solamente se solicita una sola vez, y se recupera del almacenamiento cuando sea necesario. Esto evita sobrecargar al *backend* de peticiones.

```
class AvatarView(APIView):
    permission_classes = [permissions.IsAuthenticated]

    def get(self, request, user_id):
        try:
            user = get_object_or_404(CustomUser, id=user_id)
            if request.user == user:
                if user.avatar:
                    avatar_data = user.avatar.read()
                    return JsonResponse({'avatar': avatar_data.decode('utf-8')}, status=200, safe=False)
```

Figura 52. Obtención del avatar en el *backend*.

Se ha obtenido el siguiente resultado (Figura 53):



Figura 53. Página principal.

Algunos detalles a tener en cuenta son:

- Los iconos superiores se han incluido en el banner, al ser elementos relacionados con el jugador (registro de partidas y notificaciones).

- Se ha eliminado el icono de ajustes y se ha reemplazado por el del histórico. Los ajustes serán mostrados en la cuarta pestaña 'settings'.
- Se han puesto todos los botones de color turquesa (color secundario de la app) para no sobrecargarlo demasiado de morado.

Más información acerca de esto puede encontrarse en <https://github.com/davidcr01/WordlePlus/issues/23#issuecomment-1615853793>. Contiene un video ilustrativo al final.

III.3.4.2. Wordle avanzado

Una de las nuevas incorporaciones al juego original es la capacidad de elegir el tamaño de las palabras para personalizar la experiencia del jugador.

Al haber implementado el juego Wordle como un componente reutilizable y parametrizable, lo único que hay que añadir es un parámetro a la página *classic-wordle* que represente el número de letras. Posteriormente, este valor se le pasará al componente como se explicó en el Sprint 3 (Figura 54).

```
{
  path: 'classic-wordle/:length',
  loadChildren: () => import('./pages/classic-wordle/classic-wordle.module').then( m => m.ClassicWordlePageModule),
  canActivate: [AuthGuard]
},
```

Figura 54. Parámetro para seleccionar el tamaño de la palabra.

Para permitir la selección de la palabra al jugador, se ha utilizado un *popover*. Este componente es una caja de dialogo generada sobre la página actual, y evita tener que crear una nueva página.

En este caso, se han implementado:

- El componente *WordsPopoverComponent*: define la lista de tamaño de palabras permitidos y construye una lista de botones iterando sobre dicha lista.
- El controlador *handleSelectionPopover* del popover dentro de la página *tab1*, que crea el popover usando el componente cuando se hace click al botón de juego clásico.

El resultado ha sido el siguiente (Figura 55):

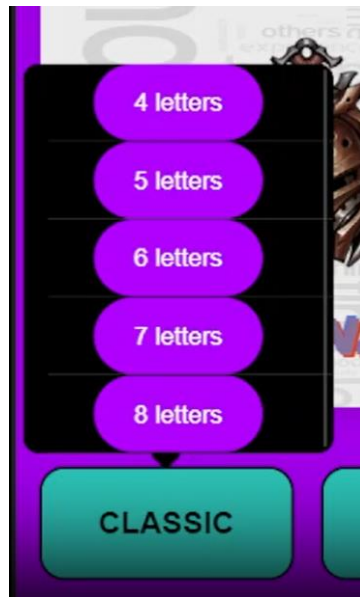


Figura 55. Popover de selección de número de letras

Más información detallada se encuentra en:
<https://github.com/davidcr01/WordlePlus/issues/25>

III.3.4.3. Notificaciones

Para las notificaciones, es necesario implementar el modelo, serializador, vista y ruta relacionados en la API, de forma similar a como se ha hecho en ocasiones anteriores. El modelo queda de la siguiente manera (Figura 56):

```
class Notifications(models.Model):
    player = models.ForeignKey(Player, on_delete=models.CASCADE,
related_name='notifications')
    text = models.CharField(max_length=200)
    link = models.URLField(blank=True)
    timestamp = models.DateTimeField(auto_now_add=True)
```

Figura 56. Modelo de las notificaciones.

Respecto a las vistas, se ha implementado de forma similar al de ClassicWordle, se obtienen las notificaciones asociadas al jugador identificado por el token, sin necesidad de pasar parámetros adicionales.

Más información puede consultarse en:
<https://github.com/davidcr01/WordlePlus/issues/27#issuecomment-1616018612>

En el *frontend*, se ha desarrollado:

- Componente *notification.popover*: de forma similar al *popover* de selección de tamaño, se ha definido otro *popover* para las notificaciones. Este componente tiene como HTML la construcción de una lista iterando

sobre las diferentes notificaciones, y la obtención de las notificaciones del jugador a través del servicio *NotificationService*.

- Servicio *NotificationService*: se encarga de obtener, almacena, crear y gestionar las notificaciones del jugador.

A continuación, se muestra el método *getNotifications* del servicio *NotificationService* (Figura 57). Obtiene las notificaciones de la variable del propio servicio, o del servicio de almacenamiento, o de la API en caso de que no estén almacenadas. Esto permite que las notificaciones se gestionen de forma eficiente y que no se obtengan de la API cada vez que el *popover* se despliegue. Cuando el usuario entra a la vista de *Main*, se refrescarán las notificaciones.

```
getNotifications(): Promise<any[]> {  
  return new Promise(async (resolve) => {  
    if (this.notifications.length > 0) {  
      resolve(this.notifications);  
    } else {  
      const storedNotifications = await this.storageService.getNotifications();  
      console.log(storedNotifications);  
      if (storedNotifications) {  
        this.notifications = storedNotifications;  
        resolve(this.notifications);  
      } else {  
        (await this.apiService.getNotifications()).subscribe((apiNotifications:  
any[]) => {  
          this.notifications = apiNotifications || [];  
          this.storageService.setNotifications(this.notifications);  
          resolve(this.notifications);  
        });  
      }  
    }  
  });  
}
```

Figura 57. Método *getNotifications* del servicio de notificaciones.

Más información puede consultarse en:
<https://github.com/davidcr01/WordlePlus/issues/27#issuecomment-1616714469>

Como resultado, puede consultarse el vídeo al final del siguiente enlace en donde se muestra cómo se actualiza la información del jugador además de sus notificaciones (Figura 58):

<https://github.com/davidcr01/WordlePlus/issues/27#issuecomment-1616718236>

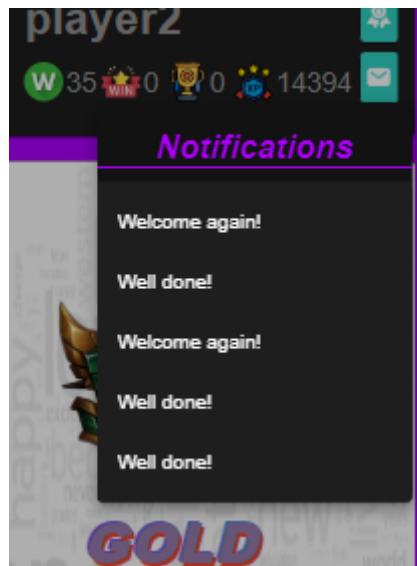


Figura 58. Popover de notificaciones.

III.3.4.4. Modificación de información personal

Esta funcionalidad se ha añadido dentro del apartado de Ajustes de la aplicación. Esta página se ha modelado como una lista de acciones que puede realizar el usuario. (Figura 59).

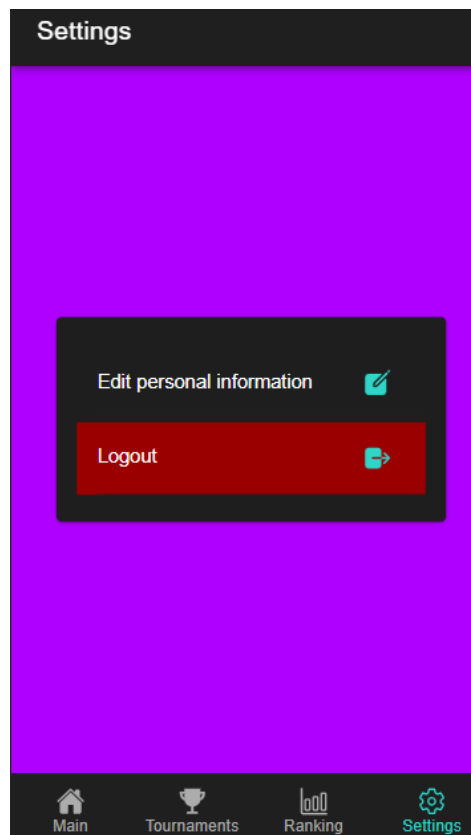


Figura 59. Página de ajustes.

Se ha creado la nueva página *edit-user* que se carga al pulsar en la opción de editar la información personal. Esta página contiene dos formularios autorellenados con la información del usuario y su avatar.

Para realizar esto, un nuevo serializador y vista han sido implementados en el *backend* para proporcionar solamente la información interesada. Más información se encuentra en: <https://github.com/davidcr01/WordlePlus/issues/26#issuecomment-1617130751>

Al cargar el formulario, se hace uso de la función *getUserInfo*, que realiza una petición POST a la API para obtener la información personal del jugador y obtiene su avatar del servicio de almacenamiento.

En caso de que el formulario de datos personales sea modificado por el usuario, se hace uso de la función *saveUserInfo*, que valida los campos y realiza una petición PATCH a la API para actualizar la información. Al modificar solamente parte del recurso y no su totalidad, se usa el método PATCH frente al método PUT.

En caso del avatar, se ofrece una vista previa del avatar actual. Cuando el usuario selecciona un nuevo avatar de su dispositivo, se actualiza esta vista previa con la nueva imagen usando la función *readAndPreviewAvatar*. Cuando el usuario confirma el formulario, la imagen es guardada en el almacenamiento y enviada a la API en base64 a través de la función *uploadAvatar* (Figura 60).

```
// En HTML
<input type="file" accept="image/*" #avatarInput
(change)="readAndPreviewAvatar(avatarInput.files[0])"/>
<ion-button expand="full" (click)="uploadAvatar()">Upload New Avatar</ion-
button>

// En TS
async uploadAvatar() {
  const file = this.avatarInput.nativeElement.files[0];
  if (file) {
    console.log(file);
    const reader = new FileReader();
    reader.onloadend = () => {
      const avatarData = reader.result as string;
      console.log(avatarData);
      this.saveAvatar(avatarData);
    };
    reader.readAsDataURL(file);
  }
}
```

Figura 60. Lógica de subida del avatar.

Por tanto, en la base de datos el campo *avatar* del usuario almacena la imagen en base64.

Más detalles acerca de esto pueden consultarse en. Al final, hay un video ilustrativo de la nueva funcionalidad: <https://github.com/davidcr01/WordlePlus/issues/26#issuecomment-1618488792>

Finalmente, la página de modificación de datos queda de la siguiente manera (Figura 61).

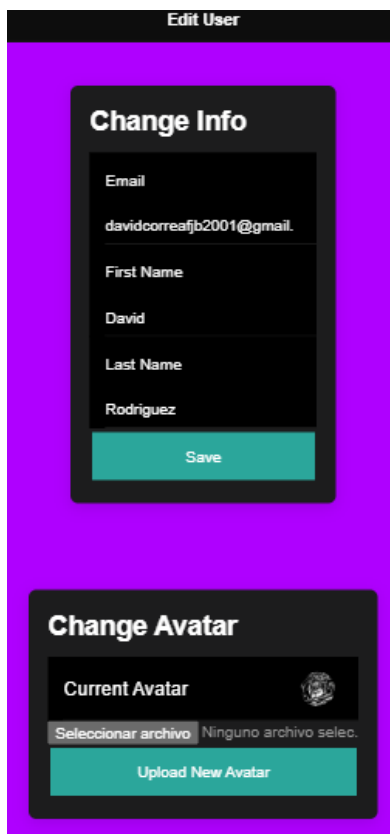
The image shows a mobile application interface for editing a user profile. At the top, there is a black header with the text 'Edit User' in white. Below this, the page has a light blue background. There are two main white cards with black borders. The first card is titled 'Change Info' in bold black text. It contains four input fields: 'Email' with the value 'davidcorrea@b2001@gmail.', 'First Name' with the value 'David', and 'Last Name' with the value 'Rodriguez'. Below these fields is a red 'Save' button. The second card is titled 'Change Avatar' in bold black text. It shows the 'Current Avatar' as a small circular profile picture. Below this is a button labeled 'Seleccionar archivo' (Select file) and the text 'Ninguno archivo selec.' (No file selected). At the bottom of this card is a red 'Upload New Avatar' button.

Figura 61. Página de modificación de datos.

III.3.4.5. Revisión y retrospectiva

Como **revisión** del Sprint, las historias de usuario que se plantearon se han completado satisfactoriamente, siguiendo los suficientes criterios de seguridad y calidad. Además, se ha implementado la vista *Main* que representa la página principal dentro de la plataforma. El resultado del incremento es satisfactorio, habiendo realizado un incremento atractivo y funcional. En este caso, el incremento se incorporará a la aplicación final (Release v2.0.0).

Como **retrospectiva** del Sprint, éste ha durado más tiempo de lo planificado, ya que las historias de usuario estaban relacionadas con nuevos desarrollos, y la implementación de la vista principal ha tomado más tiempo de lo esperado, ya que ha requerido muchas pruebas para comprobar que se adaptara correctamente a las pantallas. Sin embargo, la edición del avatar, que no estaba contemplada, ha sido completada satisfactoriamente.

III.3.5. Sprint 5

Nº	Objetivo	Fecha de entrega	
5	Creación torneo	10 de abril del 2023	
Ident.	Título	Estimación	Prioridad
HU.7	Como administrador quiero crear una sala abierta de un torneo	2	2
HU.26	Como jugador quiero poder ver los torneos disponibles	2	2
HU.17	Como jugador quiero poder ver mi perfil de jugador	2	3

El objetivo de este Sprint es comenzar a desarrollar la lógica de los torneos, inicialmente su creación por parte de los administradores, y su listado por parte de los jugadores.

Respecto a la HU.17, esta se considera implementada en el sprint 4, donde se implementó la página *Main*. En esta página, la cabecera de la página muestra toda la información del perfil del jugador.

El boceto del listado de los torneos, desarrollado con la herramienta *Figma*, es el siguiente (Figura 62). Se muestra un listado de la información de los torneos en cartas, con el botón *Join* del color secundario de la aplicación, obteniendo una vista atractiva y limpia:

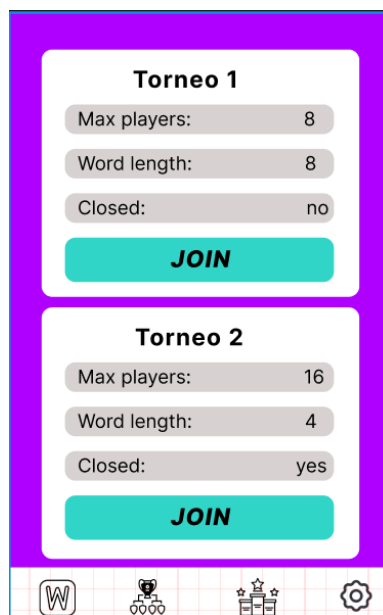


Figura 62. Listado de torneos

III.3.5.1. Creación del torneo

Al ser la creación del torneo una funcionalidad exclusiva de los administradores no es necesario implementar métodos de escritura en la API. Solamente se necesitará la petición GET para que los jugadores puedan recibir la información de los torneos.

Se han desarrollado como es habitual el modelo, serializador y la vista de esta nueva tabla (Figura 63). El modelo se ha implementado exactamente como se definió en el diseño de la base de datos, y la vista permite un parámetro opcional *word_length* para filtrar los torneos por su longitud:

```
# Model
class Tournament(models.Model):
    name = models.CharField(max_length=20)
    description = models.TextField(blank=True)
    max_players = models.PositiveIntegerField()
    word_length = models.PositiveIntegerField()
    is_closed = models.BooleanField(default=False)

# View
class TournamentViewSet(viewsets.ReadOnlyModelViewSet):
    queryset = Tournament.objects.order_by('max_players')
    serializer_class = TournamentSerializer

    def get_queryset(self):
        queryset = super().get_queryset()
        # Obtain the word_length parameter
        word_length = self.request.query_params.get('word_length')

        if word_length:
            queryset = queryset.filter(word_length=word_length)
        return queryset
```

Figura 63. Modelo y vista de los torneos.

Nótese como la vista modifica la consulta dependiendo de la existencia del parámetro *word_length*.

Para su gestión para los administradores, el modelo ha sido añadido al *Admin site*. Un aspecto importante es que los torneos deben tener un tamaño potencia de 2, y la longitud de palabras tiene que coincidir con el ya definido (de 4 a 8 letras, ambos incluidos). Por tanto, se ha redefinido el formulario asociado a la creación de torneos en el *Admin site* (Figura 64):

```

class TournamentsForm(forms.ModelForm):
    word_length = forms.ChoiceField(choices=[
        (4, '4'), (5, '5'), (6, '6'), (7, '7'), (8, '8'),
    ])
    max_players = forms.ChoiceField(choices=[
        (2, '2'), (4, '4'), (8, '8'), (16, '16'),
    ])
    class Meta:
        model = Tournament
        fields = '__all__'

class TournamentAdmin(admin.ModelAdmin):
    list_display = ('name', 'description', 'max_players', 'word_length',
        'is_closed')
    form = TournamentsForm

```

Figura 64. Vista del administrador de los torneos.

De esta manera, al crear los torneos, estos valores serán seleccionados por desplegables, y no podrán tomar cualquier número.

Nótese que, gracias a esta implementación, las historias HU.8 y HU.9 (selección de tamaño de palabra y de número de jugadores) del sprint 6 han sido completadas en este desarrollo.

III.3.5.2. Listado de torneos

La vista de los torneos se encuentra en la segunda pestaña de la plataforma (*tab2*). El fichero HTML *tab2.page.html* se divide en dos partes:

- Barra de selección: utiliza un *ion-toolbar* y *ion-segment* para crear una barra de selección del tamaño de la palabra de torneo. Sirve como filtro para listar los torneos. Cada vez que cambia llama al método *loadTournaments()*;
- Listado de torneos: iterando sobre los torneos obtenidos, la información se muestra en cartas.

Un ejemplo de este código es el siguiente (Figura 65):


```

<ion-toolbar>
  <ion-segment [scrollable]="true" [(ngModel)]="wordLength"
  (ionChange)="loadTournaments()">
    <ion-segment-button value="">
      <ion-label>All</ion-label>
    </ion-segment-button>
    <ion-segment-button value="4">
      <ion-label>4 Lett.</ion-label>
    </ion-segment-button>
    ...
  </ion-segment>
</ion-toolbar>

...
<ion-card *ngFor="let tournament of tournaments">
  <ion-card-header class="tournament-header">
    <ion-card-title>{{ tournament.name }}</ion-card-title>
  </ion-card-header>
  ...

```

Figura 65. HTML del listado de torneos.

Respecto al controlador de la página, solicita a la API los torneos filtrados por la longitud de la palabra seleccionada en el *ion-segment*. Para ello, se ha definido un nuevo método *getTournaments()* en el *ApiService*.

La información devuelta se guarda en la variable *this.tournaments* para mostrar su contenido en el HTML. Más detalles puede encontrarse en el siguiente enlace: <https://github.com/davidcr01/WordlePlus/issues/37#issuecomment-1620306330>

El resultado es el siguiente (Figura 66):

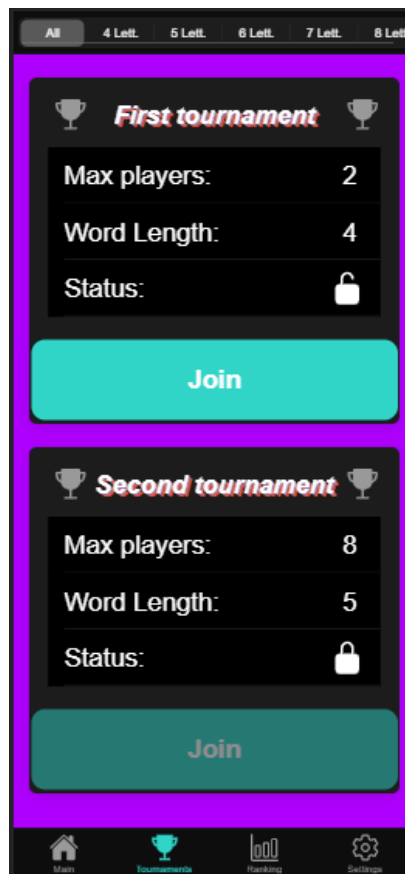


Figura 66. Listado de los torneos.

Con la barra de herramientas implementada, el filtrado de torneos (HU12 del sprint 8) queda completada.

Nótese que el botón *Join* del segundo torneo se encuentra deshabilitado, ya que el torneo está cerrado. En el siguiente enlace, se muestra un video ilustrativo de cómo cambia el listado de los torneos dependiendo del filtro: <https://github.com/davidcr01/WordlePlus/issues/37#issuecomment-1620315128>

III.3.5.3. Revisión y retrospectiva

Como **revisión** del Sprint, las historias de usuario que se plantearon se han completado satisfactoriamente, siguiendo los suficientes criterios de seguridad y calidad. Además, se ha implementado el filtro de los torneos que será de utilidad más adelante. El resultado del incremento es satisfactorio, habiendo realizado un incremento atractivo y funcional. Como todavía los jugadores no pueden jugar torneos, el incremento no se añadirá al producto final por ahora.

Como **retrospectiva** del Sprint, éste ha durado menos tiempo de lo planificado. Gracias al correcto desarrollo del *Admin site*, las historias de usuario planteadas han sido resueltas de forma rápida y sencilla. Es conveniente utilizar todas las utilidades que ofrezcan los *framework* que se adapten a los desarrollos para así maximizar y utilizar el tiempo de forma eficiente.

III.3.6. Sprint 6

Nº	Objetivo	Fecha de entrega	
6	Configuración torneo	24 de abril del 2023	
Ident.	Título	Estimación	Prioridad
HU.8	Como administrador quiero poder seleccionar el tamaño del torneo (número de participantes)	2	2
HU.9	Como administrador quiero poder seleccionar la longitud de la palabra del torneo	2	2
HU.10	Como administrador quiero poder crear un torneo y preseleccionar los participantes	3	2

El objetivo de este Sprint es poder configurar la creación del torneo, es decir, especificar los parámetros de tamaño del torneo y de la longitud de la palabra; además de poder crear participaciones y asignarlas a los torneos.

Como ya se mencionó en el sprint 5, las historias de usuario HU.8 y HU.9 se completaron en dicho Sprint.

III.3.6.1. Participaciones

Para ello, se ha añadido un nuevo modelo *Participations* siguiendo el modelado de la base de datos, además de su serializador y vista asociados.

Al estar el modelo *Participations* estrechamente relacionado con el de los torneos, la vista debe modificar la información del torneo y crear una nueva notificación al jugador de que se ha inscrito al torneo. Este es el caso del método *create* de la vista *ParticipationViewSet* (Figura 67):

```

def create(self, request, *args, **kwargs):
    tournament_id = request.data.get('tournament_id')
    player = request.user.player
    ...
    try:
        tournament = Tournament.objects.get(id=tournament_id)
    except Tournament.DoesNotExist:
        return Response({'error': 'Invalid tournament ID'}, status=400)
    ...

    participation = Participation.objects.create(tournament=tournament,
player=player)
    tournament.num_players += 1
    # Close the tournament if is full
    if tournament.num_players >= tournament.max_players:
        tournament.is_closed = True

    tournament.save()

    # Create the related notification to the player
    message = f"You were assigned in {tournament.name}. Good luck!"
    link = "http://localhost:8100/tabs/tournaments"
    notification = Notification.objects.create(player=player,
text=message, link=link)
    notification.save()

    serializer = self.get_serializer(participation)
    return Response(serializer.data, status=201)

```

Figura 67. Fragmento de la vista de las participaciones.

Además, este método debe realizar las siguientes comprobaciones:

- Que el usuario sea un jugador.
- Que el identificador del torneo se haya adjuntado en el cuerpo y que sea válido.
- Que el jugador no tenga una participación en dicho torneo.
- Que el torneo no esté cerrado.

De forma similar, en el método *list* para la petición GET, también se requiere el parámetro *tournament_id* para el listado de las participaciones.

III.3.6.2. Lógica del administrador

En el lado del administrador, la lógica a seguir es similar. Sin embargo, la vista del administrador tiene que actualizar los datos del torneo si se eliminan participaciones.

Para este caso, se han redefinido los siguientes métodos:

- *delete_model*: se ejecuta cuando se elimina un objeto en concreto, accediendo a su información. En este caso, el número de jugadores se

decrementa en 1 y se especifica que el torneo está abierto en caso de que no esté lleno (Figura 68).

- *delete_queryset*: se ejecuta cuando se hace una multiselección de objetos y se eliminan. En este caso, se eliminan las participaciones seleccionadas y se actualiza el número de jugadores al número de participaciones restantes, además de actualizar el campo *is_closed* si es necesario (Figura 69).

```
# Decreases the number of the players of the tournament
def delete_model(self, request, obj):
    tournament = obj.tournament
    tournament.num_players -= 1

    if tournament.num_players < tournament.max_players:
        tournament.is_closed = False

    tournament.save()
    super().delete_model(request, obj)
```

Figura 68. Método *delete_model* de la clase *ParticipationAdmin*

```
# Updates the number of players when using the multi-selection
def delete_queryset(self, request, queryset):
    tournaments = set()
    for participation in queryset:
        tournaments.add(participation.tournament)

    super().delete_queryset(request, queryset)

    for tournament in tournaments:
        num_participations =
Participation.objects.filter(tournament=tournament).count()
        tournament.num_players = num_participations

        if num_participations >= tournament.max_players:
            tournament.is_closed = True
        else:
            tournament.is_closed = False

    tournament.save()
```

Figura 69. Método *delete_queryset* de la clase *ParticipationAdmin*

En el siguiente enlace se encuentra más información acerca de este desarrollo. Además, al final del comentario, se han añadido dos videos ilustrativos para mostrar todos los casos de error y éxito de este desarrollo: <https://github.com/davidcr01/WordlePlus/issues/42#issuecomment-1620859422>

III.3.6.3. Revisión y retrospectiva

Como **revisión** del Sprint, las historias de usuario que se plantearon se han completado satisfactoriamente, siguiendo los suficientes criterios de seguridad y calidad. El resultado del incremento es satisfactorio, habiendo realizado un incremento funcional cubriendo todos los casos de error posibles. Como todavía los jugadores no pueden jugar torneos, pero sí inscribirse, el incremento no se añadirá al producto final por ahora.

Como **retrospectiva** del Sprint, éste ha durado menos tiempo de lo planificado, al igual que el anterior Sprint. Gracias al correcto desarrollo del *Admin site*, la última historia de usuario ha sido completada satisfactoriamente, sin necesidad de implementar una interfaz de usuario específica. Además, al haber realizado las dos primeras historias de usuario en el anterior sprint, nos hemos podido dedicar más a asegurar todos los casos de error de la gestión de participaciones, tanto en la API como en el *Admin site*.

III.3.7. Sprint 7

Nº	Objetivo	Fecha de entrega	
7	Perfil de jugador. Solicitudes de amistad. Lista de amigos.	19 de junio del 2023	
Ident.	Título	Estimación	Prioridad
HU.21	Como jugador quiero poder ver mi lista de amigos	3	2
HU.22	Como jugador quiero poder buscar a otro jugador por su nombre y mandarle una solicitud de amistad	3	2
HU.23	Como jugador quiero poder aceptar una solicitud de amistad	2	2

El objetivo de este Sprint es comenzar a desarrollar toda la lógica relacionada con los amigos y las solicitudes de amistad. El desarrollo consiste en crear peticiones de amistad que creen relaciones de amistad cuando sean aceptadas.

El boceto de la vista de los amigos (Figura 70) y de las peticiones de amistad (Figura 71) realizados en Figma son los siguientes:



Figuras 70 y 71. Vista de amigos y peticiones de amistad

Los bocetos muestran listados sencillos y limpios, con botones para acceder a las acciones asociadas. Para agrupar ambas funcionalidades, se ha utilizado una pestaña superior para seleccionar la vista. Los bocetos siguen la paleta de colores definida originalmente.

III.3.7.1. Modelado de amigos y peticiones

Para este desarrollo, es necesario crear los modelos, serializadores y vistas para gestionar los amigos y las peticiones de amistad. Los modelos *FriendList* y *FriendRequest* se han creado de acuerdo con el análisis de la base de datos.

En este caso, ambos modelos son muy similares, aunque tengan objetivos distintos (Figura 72):

```

class FriendList(models.Model):
    sender = models.ForeignKey(Player, on_delete=models.CASCADE,
related_name='friend_requests_sent')
    receiver = models.ForeignKey(Player, on_delete=models.CASCADE,
related_name='friend_requests_received')
    created_at = models.DateTimeField(auto_now_add=True)

    class Meta:
        constraints = [
            models.UniqueConstraint(fields=['sender', 'receiver'],
name='unique_friendship')
        ]

    def clean(self):
        if self.sender == self.receiver:
            raise ValidationError('You can not be friend of yourself.')
        if FriendList.objects.filter(sender=self.receiver,
receiver=self.sender).exists():
            raise ValidationError('This friend relation already exists.')

```

Figura 72. Modelo de amigos.

Por motivos de rendimiento, se ha creado una restricción única entre *sender* y *receiver* en vez de convertirlos en clave primaria. Nótese que, a la hora de crear nueva información, es necesario comprobar que el jugador no sea amigo de si mismo, y que la relación de amistad no exista a la inversa. El modelo *FriendRequest* sigue la misma estructura y metodología.

En el caso de listar las relaciones de amistad, solo nos interesa devolver el otro jugador de la relación. De forma similar, para las solicitudes de amistad solamente es necesario obtener el solicitante de la petición.

En el caso de crear las peticiones de amistad, al gestionar además las relaciones de amistad y notificaciones, su vista queda un poco más extensa. Por ejemplo, el método para crear nuevas peticiones queda de la siguiente manera (Figura 73):


```

def create(self, request, *args, **kwargs):
    sender = getattr(request.user, 'player', None)
    # Additional checks...
    ...

    # Check if there is an existing request
    existing_request1= FriendRequest.objects.filter(sender=sender,
receiver=receiver)
    existing_request2 = FriendRequest.objects.filter(sender=receiver,
receiver=sender)
    if existing_request1.exists() or existing_request2.exists():
        return Response({'error': 'Friend request already sent'},
status=status.HTTP_400_BAD_REQUEST)

    # Check if there is an existing friendship
    existing_friendship1 = FriendList.objects.filter(sender=sender,
receiver=receiver)
    existing_friendship2 = FriendList.objects.filter(sender=receiver,
receiver=sender)
    if existing_friendship1.exists() or
existing_friendship2.exists():
        return Response({'error': 'Friendship already exists'},
status=status.HTTP_400_BAD_REQUEST)

    # Create the request and notify it
    friend_request = FriendRequest.objects.create(sender=sender,
receiver=receiver)
    Notification.objects.create(
        player=receiver,
        text='You have a new friend request!',
        link='http://localhost:8100/friendlist'
    )

    serializer = FriendRequestSerializer(friend_request)
    return Response(serializer.data, status=status.HTTP_201_CREATED)

```

Figura 73. Método para crear peticiones de amistad.

Además de otras comprobaciones, el método comprueba si la relación de amistad existe o si ya se ha creado la petición. En otro caso, se crea la petición de amistad y se notifica a ambos jugadores.

Cabe destacar que algunas condiciones de la vista se han definido en el modelo. Sin embargo, es preferible controlar los datos entrantes antes de intentar añadirlos para evitar errores de restricciones de la base de datos.

Más información de las relaciones de amistad se encuentra en: <https://github.com/davidcr01/WordlePlus/issues/44#issuecomment-1621589655>

Más información de las peticiones de amistad se encuentra en: <https://github.com/davidcr01/WordlePlus/issues/45#issuecomment-1622619734>

En las siguientes figuras (Figura 74 y 75) se muestra un ejemplo de creación de peticiones de amistad y obtención del listado de amigos. El jugador *player* es el emisor, y el jugador *player2* es el receptor.

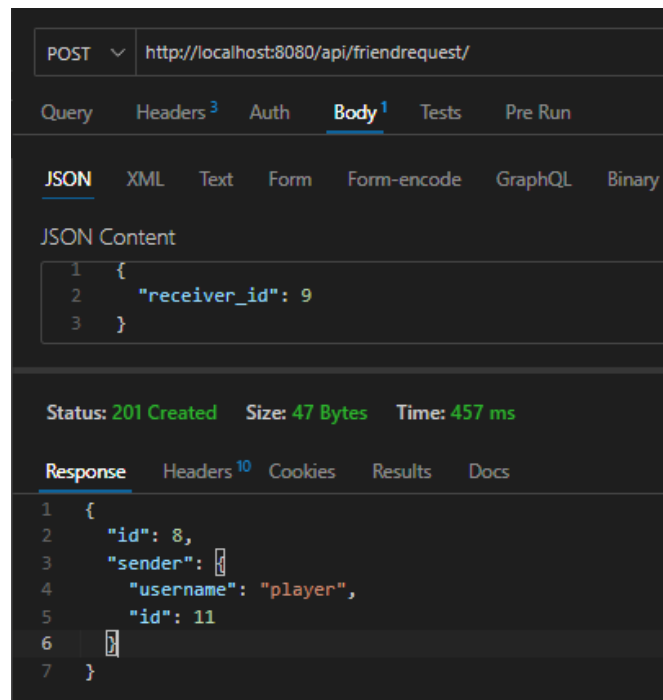


Figura 74. Creación de petición.

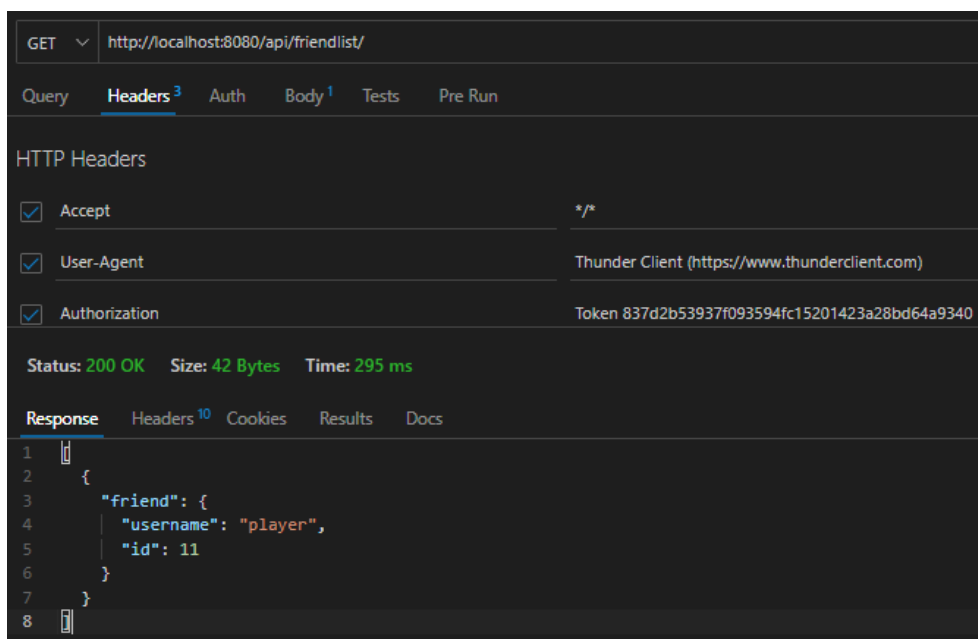


Figura 75. Listado de amigos tras aceptar la petición.

Para la aceptación o rechazo de peticiones, se han utilizado decoradores en la petición POST. Estos decoradores permiten añadir una extensión a la URL, */accept* y */reject*.

III.3.7.2. Vistas de amigos y peticiones

Para agrupar este desarrollo en el *frontend*, una nueva página *friendlist* ha sido generada. El código HTML de esta página se divide en 2 partes: la barra de

herramientas para seleccionar la pestaña y los resultados relacionados (amigos o solicitudes).

Si se pulsa en la pestaña de “Friends”, se mostrará la lista de amigos; si se pulsa en la pestaña “Request”, se mostrará la lista de solicitudes además de la barra de búsqueda de jugadores. El despliegue de información se realiza mediante condiciones y bucles. Por ejemplo, para mostrar el listado de solicitudes, se comprueba si la lista que las almacena no está vacía (Figura 76):

```
<!-- Show requests if there are requests -->
<div *ngIf="friendRequests && friendRequests.length > 0">
  <ion-card *ngFor="let request of friendRequests">
    <ion-card-header class="request-container">
      <ion-icon name="people" color="success"></ion-icon>
      <div class="username">
        {{ request.sender.username }}
      </div>
      <div class="buttons">
        <ion-button fill="clear"
(click)="acceptFriendRequest(request.id)">
          <ion-icon name="checkmark"></ion-icon>
        </ion-button>
        <ion-button fill="clear"
(click)="rejectFriendRequest(request.id)">
          <ion-icon name="close" color="danger"></ion-icon>
        </ion-button>
      </div>
    </ion-card-header>
  </ion-card>
</div>
```

Figura 76. Listado condicional de peticiones.

Respecto a la lógica de la página, diversos métodos han sido implementados para el desarrollo:

- *loadFriendList*: hace una llamada a *apiService* con el mismo nombre. Se encarga de obtener los amigos del jugador.
- *loadFriendRequest*: mismo método que el anterior pero para obtener las solicitudes de amistad.
- *segmentChanged*: controla cuando se pulsa una pestaña de la barra de herramientas. Cuando se pulsa en el segmento “Request”, se cargan las peticiones de amistad.
- *acceptFriendRequest*: hace una llamada a *apiService* con el mismo nombre. Se encarga de enviar la petición de amistad al servidor para que cree una nueva relación de amistad.

Un aspecto para destacar de este desarrollo es buscar jugadores para poder añadirlos como amigos. Para ello, se ha añadido al HTML el elemento *ion-searchbar* y se ha implementado el método *searchPlayers* (Figura 77). Este método obtiene el valor introducido en la barra de búsqueda y muestra aquellos nombres de usuario que contengan la cadena introducida.

```
// HTML
<ion-searchbar color="dark" placeholder="Search players"
(ionChange)="searchPlayers($event)"></ion-searchbar>
...
<ion-card *ngIf="filteredPlayers && filteredPlayers.length > 0">
  <ion-list>
    <ion-item *ngFor="let player of filteredPlayers">
      {{ player.username }}
      <ion-button fill="clear" slot="end" (click)="sendFriendRe-
quest(player.id)">
        <ion-icon name="add"></ion-icon>
      </ion-button>
    </ion-item>
  </ion-list>
</ion-card>

// TS
searchPlayers(event: any) {
  const username = event.target.value;
  this.showResults = true;
  if (!username) {
    this.filteredPlayers = [];
    return;
  }
  if (typeof username === 'string') {
    const filteredPlayers = this.playerUsernames.filter(player => {
      return
player['username'].toLowerCase().includes(username.toLowerCase());
    });
    this.filteredPlayers = filteredPlayers;
  }
}
```

Figura 77. Búsqueda de jugadores

Finalmente, los resultados de las vistas son los siguientes (Figura 78 y 79)

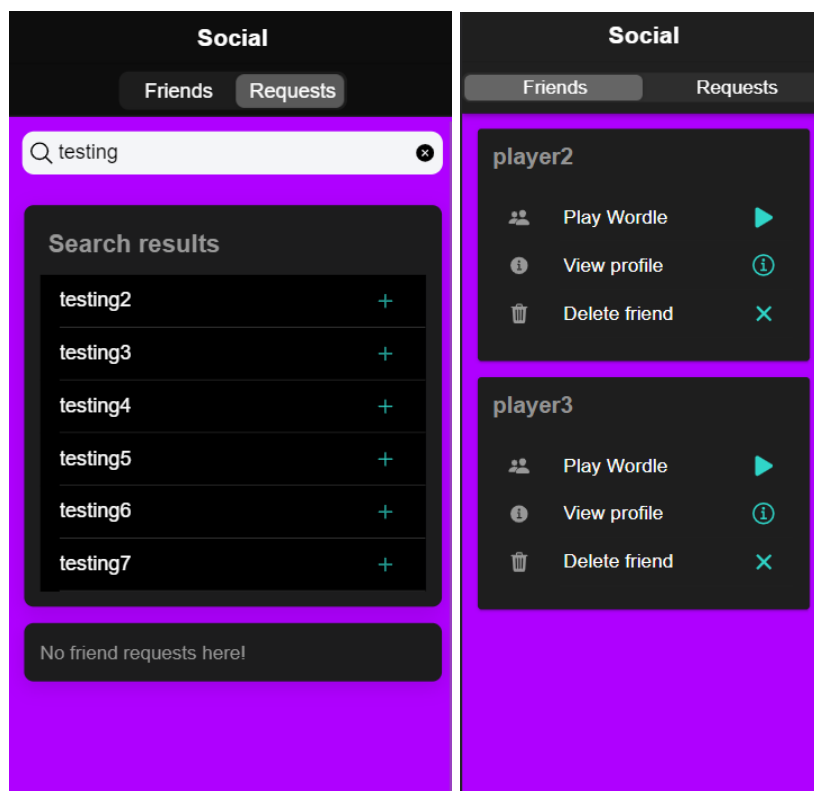


Figura 78 y 79. Vistas de peticiones de amistad y amigos.

Más información acerca de la implementación de las vistas se encuentran en los siguientes enlaces:

- <https://github.com/davidcr01/WordlePlus/issues/45#issuecomment1623890703>
- <https://github.com/davidcr01/WordlePlus/issues/44#issuecomment1622079155>

III.3.7.3. Revisión y retrospectiva

Como **revisión** del Sprint, las historias de usuario que se plantearon se han completado satisfactoriamente, siguiendo los suficientes criterios de seguridad y calidad. El resultado del incremento es satisfactorio, habiendo realizado un incremento funcional cubriendo todos los casos de error posibles. Al haber desarrollado gran parte de la lógica de las relaciones y peticiones de amistad, el incremento se añadirá al producto final (Release v2.1.0)

Como **retrospectiva** del Sprint, éste ha durado aproximadamente el tiempo planificado. Hemos invertido bastante tiempo en comprobar casos de error y posibles heurísticas respecto a las peticiones y relaciones de amistad. Estas prácticas de exhaustivo testeo son necesario para crear un producto robusto y de calidad. Por otro lado, también se han implementado aspectos relacionados con la denegación de peticiones de amistad, que será útil en futuros Sprints.

III.3.8. Sprint 8

Nº	Objetivo	Fecha de entrega	
8	Multijugador 1vs1	8 de mayo del 2023	
Ident.	Título	Estimación	Prioridad
HU.11	Como jugador quiero poder jugar una partida 1vs1 con un amigo	5	2
HU.27	Como jugador quiero poder ver mi lista de partidas multijugador pendientes	2	2

El objetivo de este Sprint es añadir a la plataforma una de las funcionalidades más innovadoras e importantes: la posibilidad de jugar un Wordle con otro jugador. Para ello, además de implementar la lógica del multijugador, hay que implementar un listado de partidas pendientes.

III.3.8.1. Modelado de partidas multijugador

Para almacenar y gestionar las partidas multijugador, es necesario crear un nuevo modelo, serializador y vista en el *backend*. En el caso del modelo, se ha implementado al igual que se definió en el análisis de la base de datos.

En cuanto a los serializadores, se han añadido dos nuevos serializadores:

- *GameDetailSerializer*: maneja toda la información definida en el modelo. Es útil para obtener el listado de partidas o la información de una partida en concreto.
- *GameCreateSerializer*: maneja toda la información excepto a *player1*, debido a que es el jugador identificado que creará la partida.

La manera de seleccionar el serializador correcto en la vista es sobrescribiendo el método *get_serializer_class* (Figura 80):

```
def get_serializer_class(self):  
    if self.action in ['list', 'retrieve', 'completed_games', 'pending_games']:  
        return GameDetailSerializer  
    elif self.action in ['create', 'partial_update']:  
        return GameCreateSerializer  
    return super().get_serializer_class()
```

Figura 80. Serializador condicional.

En cuanto a la vista, se ha definido un nuevo conjunto de vistas *GameViewSet* con los métodos GET, POST y PUT. La estrategia es que el jugador que crea la partida use el método POST publicando sus resultados, y el segundo jugador actualice dicho objeto con sus resultados, resolviendo así la partida (Figura 81).

```

def partial_update(self, request, *args, **kwargs):
    instance = self.get_object()
    player = request.user.player
    ...
    allowed_fields = ['player2_xp', 'player2_time',
'player2_attempts']
    data = {key: request.data.get(key) for key in allowed_fields}
    ...
    if player2_xp is not None:
        player1_xp = instance.player1_xp
        if player2_xp > player1_xp:
            instance.winner = instance.player2
            instance.winner.wins_pvp += 1
            instance.winner.save()
        elif player2_xp < player1_xp:
            instance.winner = instance.player1
            instance.winner.wins_pvp += 1
            instance.winner.save()
        else:
            ...

    serializer = self.get_serializer(instance, data=data,
partial=True)
    serializer.is_valid(raise_exception=True)
    serializer.save()

    return Response({'winner': instance.winner.user.username})

```

Figura 81. Método *partial_update* de la vista de las partidas multijugador.

Nótese cómo solamente se permiten los campos relacionados con los resultados del segundo jugador, y se devuelve el nombre de usuario del ganador dependiendo de las estadísticas de ambos jugadores.

Además, para los métodos GET se han utilizado dos decoradores *pending_games* y *completed_games* para obtener las partidas completadas y pendientes. La sección de partidas pendientes será necesaria para que el segundo jugador pueda responder a la partida. El boceto sería el mismo que la lista de amigos, pero mostrando el tiempo que ha tardado el otro jugador en completar la partida y la longitud de la palabra a adivinar.

El método *pending_games*, necesario para ver el listado de las partidas pendientes queda de la siguiente manera (Figura 82). Nótese que las partidas pendientes son aquellas que aún no tiene ganador y en las que el segundo jugador es el jugador identificado:

```

@action(detail=False, methods=['get'])
def pending_games(self, request):
    player = getattr(request.user, 'player', None)
    if not player:
        return Response({'error': 'Player not found'}, status=404)
    queryset = Game.objects.filter(player2=player,
winner=None).order_by('timestamp')
    serializer = self.get_serializer(queryset, many=True)
    return Response(serializer.data)

```

Figura 82. Método para listar las partidas pendientes.

Más información acerca de esta parte del desarrollo se encuentra en: <https://github.com/davidcr01/WordlePlus/issues/50#issuecomment-1625914789>

III.3.8.2. Vistas del multijugador

En el caso de la vista del multijugador, esta es idéntica a la vista del juego clásico original, excepto que en la parte superior de la interfaz aparecerá los nombres de usuario de la partida. Por ello mismo, no se ha diseñado un boceto asociado.

Para gestionar la creación y respuesta de las partidas, se han diseñado dos nuevas páginas *CreateGame* y *RespondGame*. Estas páginas tienen código HTML muy similar, ambos usan el componente *WordleDashboard* pero su lógica es diferente.

- *CreateGame* recibe por parámetros el identificador y nombre de usuario del jugador con quien se quiere jugar la partida. Cuando el jugador que inicia la partida la completa, se crea la partida multijugador con sus resultados.
- *RespondGame* recibe por parámetro el identificador de la partida que se creó en *CreateGame*. De forma análoga, el segundo jugador completa la partida, publica sus resultados y se resuelve la partida.

Al tener lógicas distintas, y para no desarrollar código demasiado enrevesado, se ha optado por la división en estas páginas. Para la comunicación entre *WordleDashboard* y las anteriores páginas, se ha usado el evento *gameFinished* para ejecutar los métodos *finishGame* (Figura 83):


```

// En wordle-dashboard.component.ts
@Output() gameFinished: EventEmitter<any> = new EventEmitter();
...
if (this.isMultiplayer) {
    const gameFinishedEvent = {
        time: timeConsumed,
        xp: xP,
        attempts: attemptsConsumed,
        selectedWord: this.rightGuessString,
    };
    this.gameFinished.emit(gameFinishedEvent);
}

// En create-game.page.ts
async finishGame(time: number, xp: number, attempts: number,
selectedWord: string) {
    const gameData = {
        player2: this.opponentId,
        player1_time: time,
        player1_xp: xp,
        player1_attempts: attempts,
        word: selectedWord,
    };

    (await this.apiService.createGame(gameData)).subscribe(

// En respond-game.page.ts
async finishGame(time: number, xp: number, attempts: number,
selectedWord: string) {
    const gameData = {
        player2_time: time,
        player2_xp: xp,
        player2_attempts: attempts,
    };

    (await this.apiService.resolveGame(this.idGame, gameData)).subscribe(

```

Figura 83. Comunicación entre componente y páginas

De esta manera, el componente del panel envía los resultados del jugador en concreto, y cada página asigna los resultados al primer jugador (en caso de *create-game*) o al segundo jugador (en caso de *respond-game*).

Las siguientes figuras 84 y 85 muestran un ejemplo de partida multijugador entre *player* (el que crea la partida) y *testing* (el que acepta la partida). Nótese como la palabra a adivinar es la misma, y cuando el segundo jugador completa la partida, el resultado es mostrado:

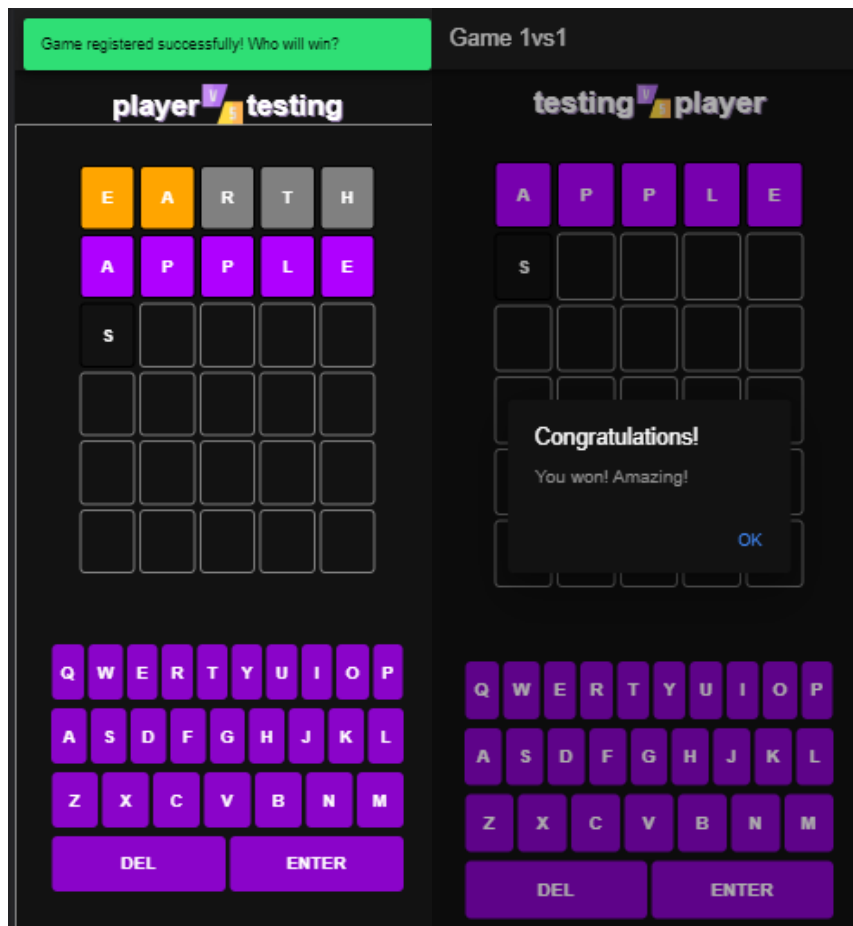


Figura 84 y 85. Partida multijugador.

Para evitar casos de error, la vista en el *backend* de las partidas multijugador comprueba que el segundo jugador esté involucrado en la partida, y que dicha partida no haya sido ya resuelta.

Más información acerca del desarrollo de las páginas se encuentra en: <https://github.com/davidcr01/WordlePlus/issues/50#issuecomment-1627885551>

III.3.8.3. Vistas de las partidas pendientes

Para mostrarle al jugador las partidas que tiene pendientes, es decir, aquellas a las que ha sido retado pero que aún no ha jugado, se ha creado una nueva página *History*. El objetivo de esta página es albergar el histórico de partidas en solitario y multijugador, además de las partidas multijugador pendientes. Esta página se ha implementado de forma similar a la lista de amigos, con una barra de herramientas para seleccionar qué información desplegar.

En el caso de las partidas pendientes, se rellena una lista usando una nueva llamada a la API *getPendingPVPGames* que usa el método GET con el decorador *pending_games* mencionado anteriormente (Figura 86 y 87).

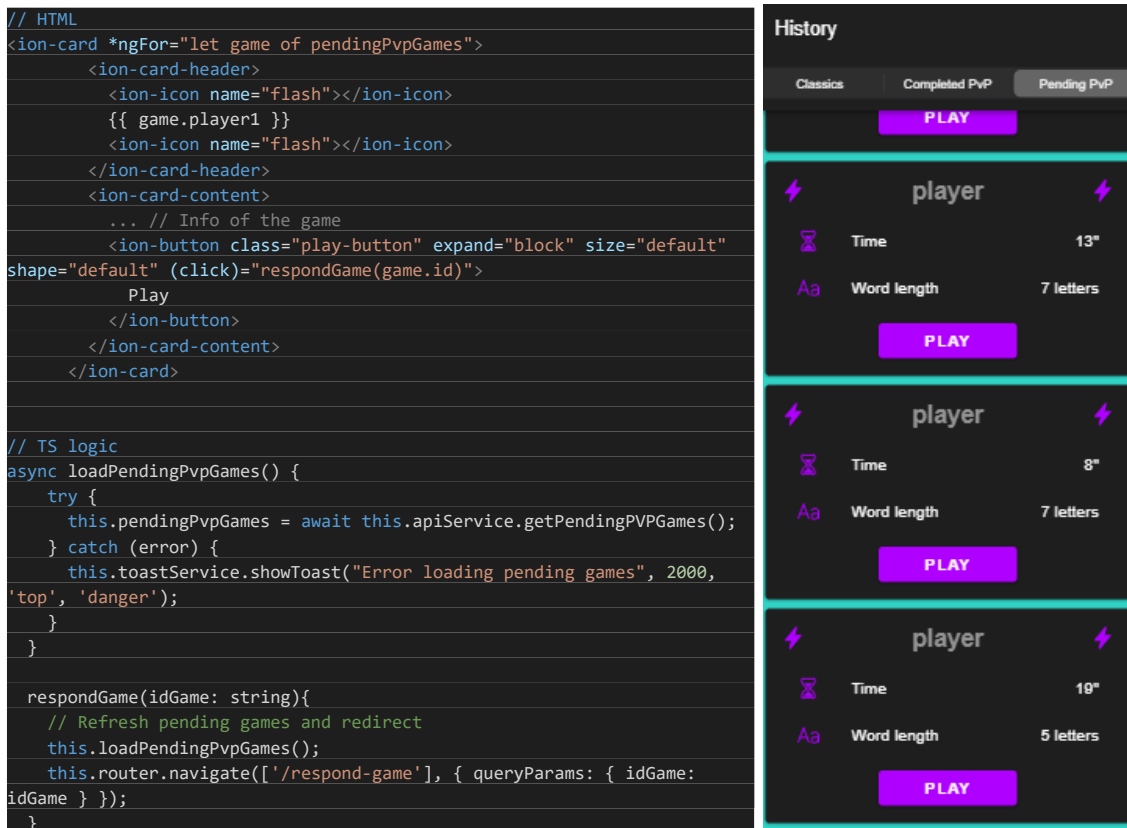


Figura 86 y 87. Histórico de partidas pendientes.

III.3.8.4. Revisión y retrospectiva

Como **revisión** del Sprint, las historias de usuario que se plantearon se han completado satisfactoriamente, siguiendo los suficientes criterios de seguridad y calidad. El resultado del incremento es satisfactorio, habiendo realizado un incremento funcional cubriendo todos los casos de error posibles. Al haber realizado el multijugador en su completitud, el incremento se añadirá al producto final (Release v3.0.0).

Como **retrospectiva** del Sprint, éste ha durado aproximadamente el tiempo planificado. Hemos invertido bastante tiempo en comprobar casos de error y en implementar una única lógica para el multijugador. Sin embargo, a causa de su complejidad e incoherencia se optó por separarlo en páginas diferentes. Es necesario evaluar todas las posibles soluciones y no siempre implementar la más eficiente, ya que también puede ser la más compleja.

III.3.9. Sprint 9

Nº	Objetivo	Fecha de entrega	
9	Histórico de partidas completadas. Inscripción al torneo	22 de mayo del 2023	
Ident.	Título	Estimación	Prioridad
HU.28	Como jugador quiero poder ver mis partidas multijugador completadas	2	2
HU.12	Como jugador quiero poder seleccionar la longitud de la palabra del torneo (tipo de torneo)	2	2
HU.13	Como jugador quiero poder unirme a una sala abierta de un torneo y jugar al torneo	4	2

El objetivo de este Sprint es mostrar las partidas multijugador completadas, además de implementar toda la lógica relacionada con el sistema de torneos, tanto la selección del tipo de torneo, como la inscripción y participación en él.

Nótese como la HU.12 fue implementada en el Sprint 5, en el listado de torneos disponibles. Más detalles pueden encontrarse en: <https://github.com/davidcr01/WordlePlus/issues/37#issuecomment-1620306330>

III.3.9.1. Lista de partidas completadas

De forma similar al listado de partidas pendientes, se ha desarrollado una lista de partidas completadas del jugador, en donde se debe mostrar cierta información de la partida, además de si el jugador fue es el ganador o no.

El boceto de esta vista, implementado en Figma, queda de la siguiente manera (Figura 88). Se establece como cabecera de la carta el enfrentamiento entre los jugadores, y como contenido el resultado del enfrentamiento y los tiempos y experiencias de la partida. El objetivo de esta vista es mostrar de forma limpia la información de las partidas, siguiendo la línea de otros diseños anteriores.

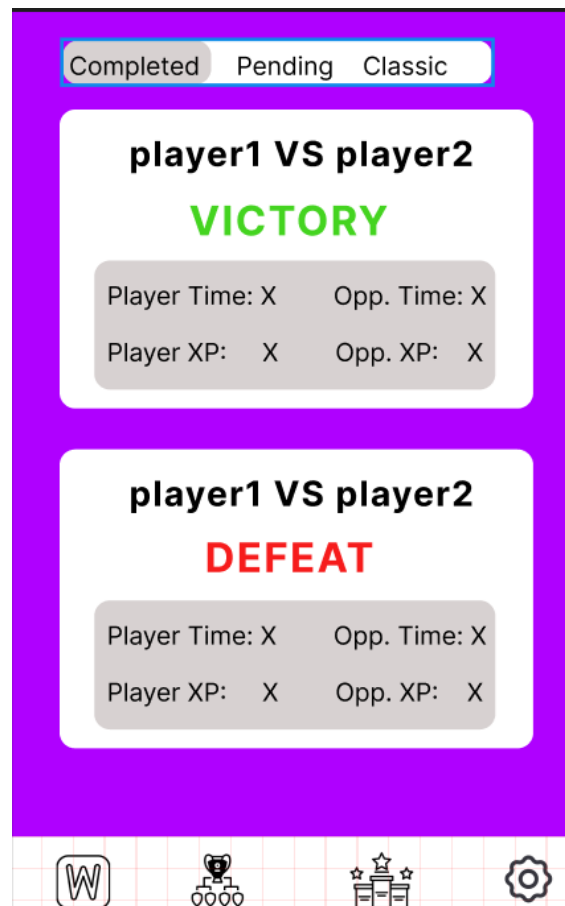


Figura 88. Boceto de partidas completadas.

Para el listado, es necesario añadir un método en el *apiService* que haga uso del método *completed_games* definido en el sprint 8. Este método devuelve todas las partidas que tengan ganador, y en donde el jugador identificado es participante de dicha partida (Figura 89). Nótese como se establece un límite de 15 objetos.

```
# Completed games are those which the winner is not null
@action(detail=False, methods=['get'])
def completed_games(self, request):
    limit = 15
    player = getattr(request.user, 'player', None)
    if not player:
        return Response({'error': 'Player not found'}, status=404)

    queryset = Game.objects.filter(Q(player1=player) |
Q(player2=player), ~Q(winner=None)).order_by('-timestamp')[:limit]
    serializer = self.get_serializer(queryset, many=True)
    return Response(serializer.data)
```

Figura 89. Método para obtener las partidas completadas.

Tras obtener esta lista, determinada información se muestra en el código HTML de la página *history*, siempre que el segmento seleccionado de la barra de herramientas sea el de *Completed PVP*.

Un aspecto importante de este listado es que el jugador a la izquierda del enfrentamiento siempre es el jugador identificado. Por tanto, toda la información en la parte derecha será la del oponente, y el mensaje de “Victoria” o “Derrota” también debe ser condicional. El siguiente código (Figura 90) ejemplifica el listado de nombres de usuario y el resultado de la partida. Nótese cómo también las cadenas *Victory* y *Defeat* se les aplica una clase distinta dependiendo del resultado:

```
<ion-card *ngFor="let game of completedPvpGames">
  <ion-card-header>
    {{ username }}
    
    {{ game.player1 === username ? game.player2 : game.player1 }}
  </ion-card-header>
  <ion-card-content>
    <ion-list class="game-info-list">
      <ion-item>
        <ion-label class="result" [class.victory-result]="game.winner ===
playerId" [class.defeat-result]="game.winner !== playerId">
          {{ game.winner === playerId ? 'Victory' : 'Defeat' }}
        </ion-label>
      </ion-item>
    </ion-list>
  </ion-card-content>
</ion-card>
```

Figura 90. Renderizado condicional de la partida completada.

De forma análoga, el tiempo y experiencia del jugador identificado se obtiene de forma condicional, dependiendo de si es el jugador 1 o 2. De esta manera, la información sale invertida para ambos jugadores. El resultado de la vista es el siguiente (Figura 91).

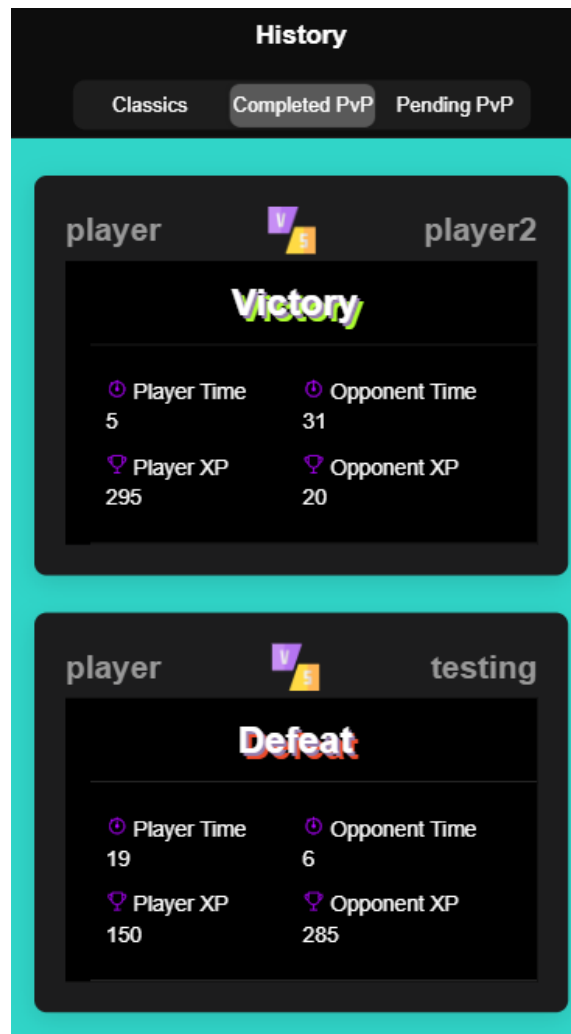


Figura 91. Listado de partidas completadas.

Más información acerca de este desarrollo puede encontrarse en: <https://github.com/davidcr01/WordlePlus/issues/60#issuecomment-1629240439>

Además, se ha implementado un nuevo componente extra *Loading* que consiste en mostrar un icono de carga cada vez que se estén recibiendo datos del *backend* antes de listarlos. Más información acerca de este componente: <https://github.com/davidcr01/WordlePlus/issues/60#issuecomment-1629241071>

III.3.9.2. Lista de torneos inscritos

Para que un participante pueda acceder a un torneo, es necesario listar los torneos a los que está inscrito. Para ello, se ha desarrollado un nuevo apartado *Joined* en la vista de listado de torneos. Se ha añadido un nuevo método *player_tournament* en el *TournamentViewSet* para obtener los torneos en los que el jugador identificado participa, y un nuevo método en *apiService* que hace uso del método anterior.

La información desplegada de estos torneos es igual que en el listado de torneos disponibles. Sin embargo, se ha cambiado el botón *Join* por *Enter*, y este botón estará disponible siempre que el torneo esté cerrado (Figura 92).

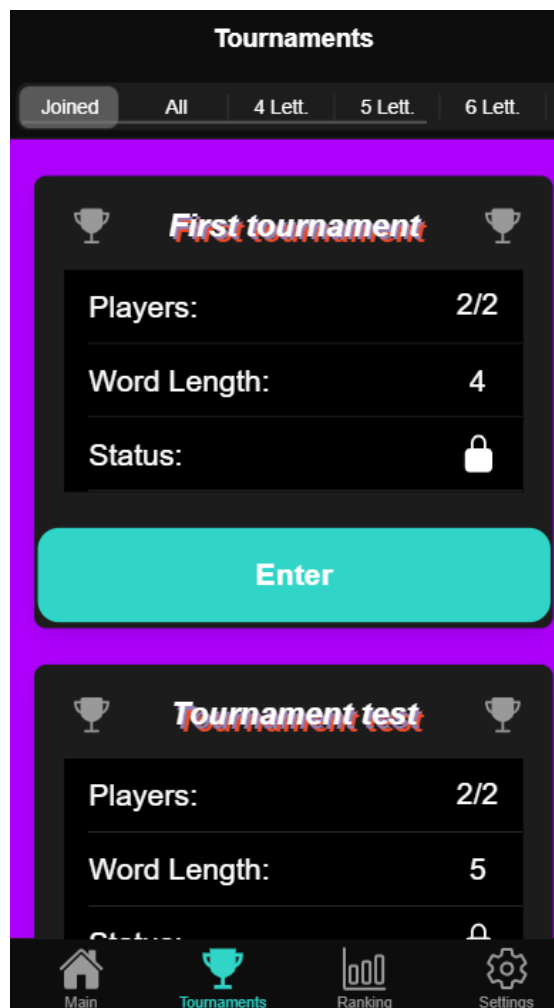


Figura 92. Listado de inscripciones a torneos.

Más información acerca de este desarrollo se encuentra en el apartado *Frontend* del siguiente enlace:

<https://github.com/davidcr01/WordlePlus/issues/49#issuecomment-1630910472>

III.3.9.3. Rondas y partidas

Un aspecto crucial en el sistema de torneos es que:

- Las rondas y las partidas de la primera ronda deben crearse automáticamente al cerrar el torneo.
- Las partidas de las siguientes rondas deben crearse automáticamente al cerrar la ronda anterior.

Se han añadido al fichero de tablas los modelos *Round* y *RoundGame* (Consist in en el diseño de la base de datos) para este desarrollo.

Para el primer punto, se ha modificado el método *create* de *ParticipationViewSet*. La misma lógica se ha implementado en el *Admin site* para los torneos con participantes preseleccionados (Figura 93):

```
def create(self, request, *args, **kwargs):
    ...
    # Close the tournament if is full
    if tournament.num_players >= tournament.max_players:
        tournament.is_closed = True

    rounds = int(math.log2(tournament.max_players))
    for round_number in range(1, rounds+1):
        round = Round.objects.create(tournament=tournament,
number=round_number)
        if round_number == 1:
            # Assign games to the first round
            participants =
Participation.objects.filter(tournament=tournament)
            participants_count = participants.count()
            for i in range(0, participants_count, 2):
                player1 = participants[i].player
                player2 = participants[i + 1].player
                new_game = Game.objects.create(player1=player1,
player2=player2, is_tournament_game=True)
                RoundGame.objects.create(round=round, game=new_game)
```

Figura 93. Gestión automática de rondas y partidas de la primera ronda.

Nótese como se crean todas las rondas además de la primera. Estas rondas quedarán vacías, pero servirá de primera mano en el *frontend* para saber las rondas del torneo.

Para el segundo punto, al ser un evento automático (que no es provocado por un jugador o un administrador), se ha implementado una nueva **señal**. Las señales en *DRF*³³ *más atrás* son funciones que se ejecutan cuando ocurre un evento en el servidor. En este caso, la siguiente señal (Figura 94) añadida en un nuevo fichero *signals.py* es ejecutada cada vez que se guarda una partida de un torneo (*Game*). La señal:

- Comprueba que todas las partidas de la ronda en cuestión tengan el campo del ganador rellenado, es decir, que las partidas estén completadas.
- En caso afirmativo, se obtienen los ganadores y,
 - o Si es la última ronda, se establece el ganador del torneo y se actualiza su información.
 - o Si no es la última ronda, se crean las partidas entre los ganadores y se asocian a la siguiente ronda.

```

@receiver(post_save, sender=Game)
def game_completed(sender, instance, created, **kwargs):
    if instance.is_tournament_game and instance.winner:
        round = Round.objects.filter(roundgame__game=instance).last()
        if round:
            round_games = RoundGame.objects.filter(round=round)
            if all(game.game.winner for game in round_games):
                tournament = round.tournament
                # Get current round
                current_round_number = tournament.current_round
                num_rounds = int(math.log2(tournament.max_players))

                # Get winners of current round
                winners = [game.game.winner for game in round_games]
                if current_round_number == num_rounds:
                    winner = instance.winner
                    winner.wins_tournament += 1
                    winner.xp += 1000
                    winner.save()
                else:
                    # Get next round
                    next_round_number = current_round_number + 1
                    next_round = Round.objects.get(tournament=tournament,
number=next_round_number)
                    tournament.current_round = next_round_number
                    tournament.save()

                    # Create games and assign them to the round
                    for i in range(0, len(winners), 2):
                        player1 = winners[i]
                        player2 = winners[i + 1]
                        new_game = Game.objects.create(player1=player1,
player2=player2, is_tournament_game=True)
                        RoundGame.objects.create(round=next_round,
game=new_game)

```

Figura 94. Señal para gestionar las siguientes rondas y partidas.

En el caso del *frontend*, se ha creado una nueva página *tournamentrounds* que accede el jugador inscrito, y en donde se muestran las rondas y las partidas. El boceto de esta vista realizado en Figma queda de la siguiente manera (Figura 95). Se muestra una barra de tareas con tantos segmentos como rondas haya en el torneo. El segmento seleccionado por defecto será la ronda actual del torneo.

Por cada segmento, se mostrará una carta con un listado de las partidas de dicha ronda, y se mostrará un botón *Play* en caso de que sea necesario que el jugador identificado juegue a dicha partida. Se ha seguido la misma filosofía de diseños anteriores.

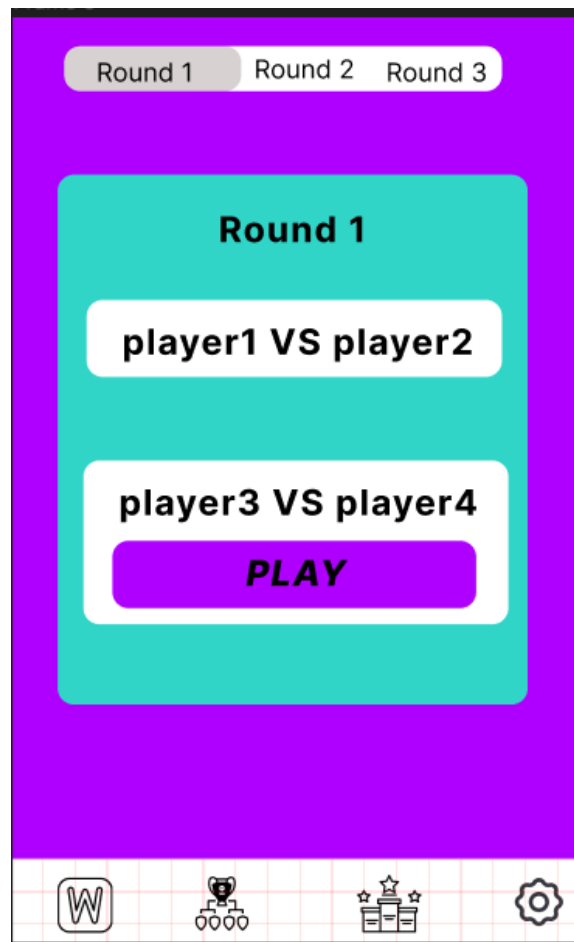
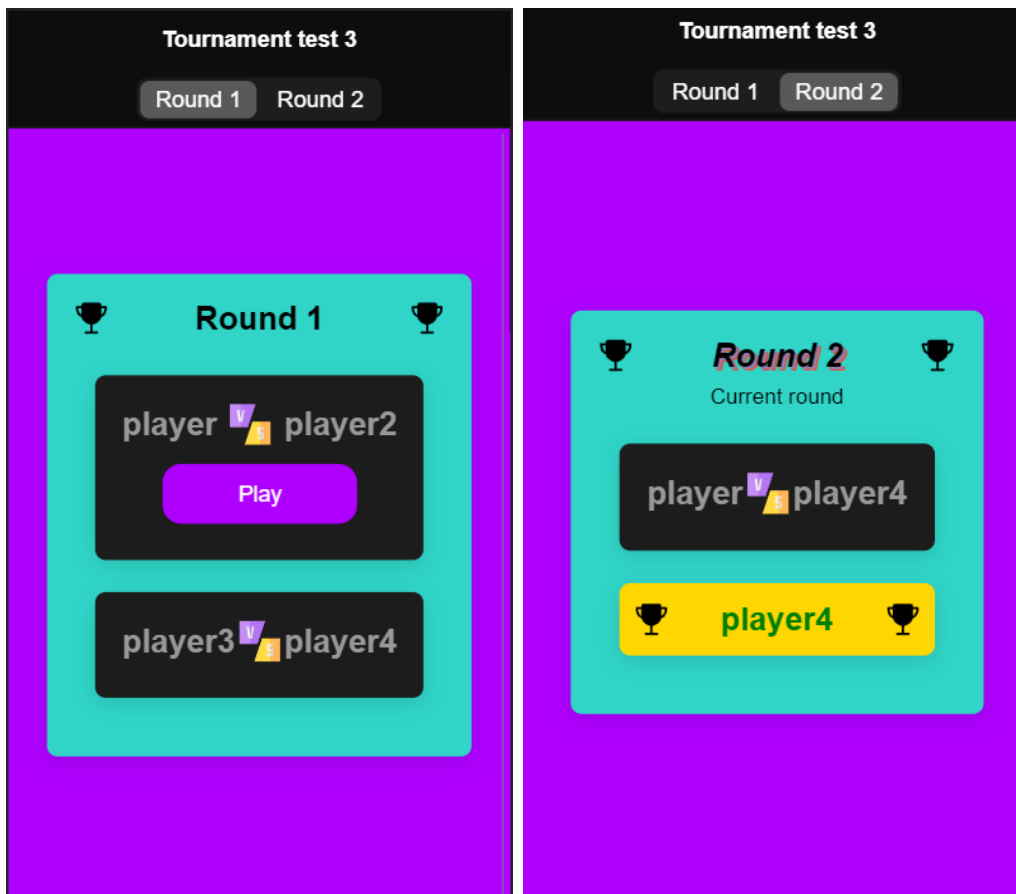


Figura 95. Boceto de las partidas de las rondas.

En el caso del HTML, se ha hecho un listado de elementos como el que se ha realizado en la página *history* o *friendlist*, cargando las rondas y partidas del torneo en concreto. Para controlar la ronda actual, se ha añadido el campo *current_round* al modelo Tournament. Con estos cambios, un torneo de dos rondas finalizado queda de la siguiente manera (Figura 96 y 97). En este caso, se muestra el ganador del torneo en la última ronda:



Figuras 96 y 97. Página *tournamentrounds*.

Más detalles de este desarrollo se puede encontrar en el siguiente enlace: <https://github.com/davidcr01/WordlePlus/issues/49#issuecomment-1630910472>

III.3.9.4. Jugando al torneo

Un aspecto importante a tener en cuenta es que las partidas de los torneos no son iguales que las partidas multijugador. Éstas últimas son creadas por un jugador, y modificadas por el otro jugador. En el caso de las partidas de los torneos, éstas son creadas automáticamente, por lo que ambos jugadores deberán de modificar la correspondiente partida con un método PATCH.

Para ello, se han creado:

- Un nuevo método PATCH *partial_update_tournament*, posteriormente renombrado *tournament* en la vista de las partidas para que cada jugador publique sus resultados. Este método solamente permite publicar la información del jugador identificado, y no la del otro jugador de la partida. Más información se encuentra en: <https://github.com/davidcr01/WordlePlus/issues/49#issuecomment-1630096360>
- Una nueva página *game-tournament* muy similar a *respond-game*, que hace uso del método mencionado anteriormente. Como la palabra de la partida es generada por el primer jugador que juega la partida, y el

segundo jugador debe adivinar la misma palabra, se ha hecho un renderizado condicional del componente *WordleDashboard*, pero no ha sido necesario modificarlo de nuevo. Más información se encuentra en: <https://github.com/davidcr01/WordlePlus/issues/49#issuecomment-1631803380>

De esta manera, se ha conseguido que el componente *WordleDashboard* sea utilizado en las partidas clásicas, en las partidas multijugador (para ambos jugadores) y en las partidas de los torneos (para ambos jugadores).

III.3.9.5. Revisión y retrospectiva

Como **revisión** del Sprint, las historias de usuario que se plantearon se han completado satisfactoriamente, siguiendo los suficientes criterios de seguridad y calidad. El resultado del incremento es satisfactorio, habiendo realizado un incremento funcional cubriendo todos los casos de error posibles. Al haber completado todo el sistema de torneos, el incremento se añadirá al producto final (Release v3.1.0).

Como **retrospectiva** del Sprint, éste ha durado más tiempo del tiempo planificado. Hemos invertido bastante tiempo en la gestión de rondas y partidas automáticamente, comprobando los casos de error y que éstas se crearan correctamente. Sin embargo, la correcta implementación del componente *WordleDashboard* en los anteriores sprints ha permitido un desarrollo más rápido de lo esperado, quedando un resultado coherente con el código implementado anteriormente.

III.3.10. Sprint 10

Nº	Objetivo	Fecha de entrega	
10	Visualización del ranking. Wordles completados	05 de junio del 2023	
Ident.	Título	Estimación	Prioridad
HU.29	Como jugador quiero poder ver la lista de los Wordles completados	2	2
HU.15	Como jugador quiero poder ver el ranking	4	3
HU.16	Como jugador quiero poder seleccionar el filtro del ranking	2	3

El objetivo de este Sprint es terminar de completar el histórico de partidas añadiendo la lista de Wordles completados, además de implementar la vista y lógica del *ranking*.

III.3.10.1. Lista de Wordles completados

Para obtener la lista de Wordles completados, es necesario añadir un método *list* al *viewset* de *ClassicWordle*. Este método obtiene un total de 15 las partidas clásicas del jugador identificado. A continuación se muestra un ejemplo de lo que retorna esta función (Figura 98):

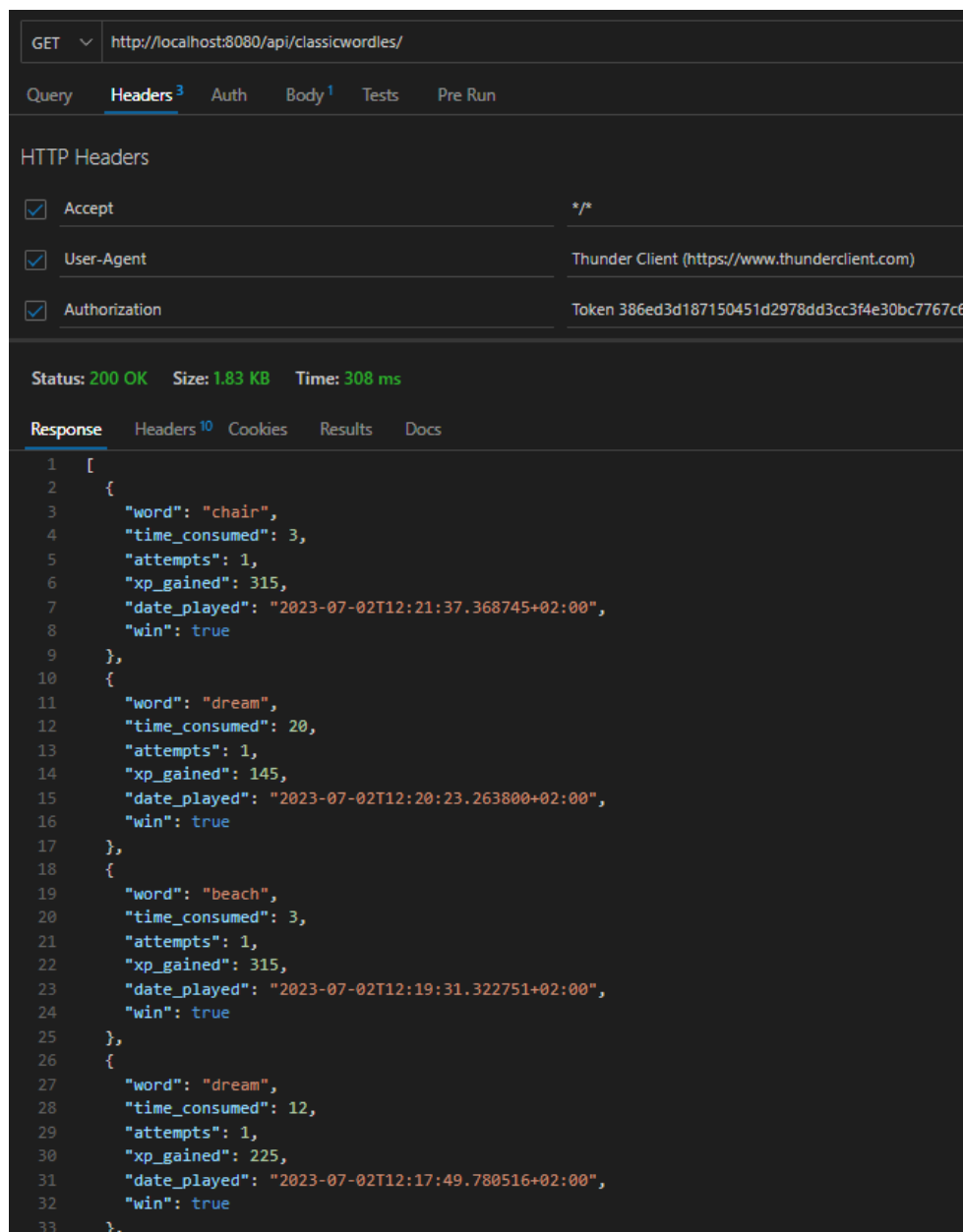


Figura 98. Respuesta de listado de Wordles completados

En cuanto a la vista en el *frontend*, ésta será muy similar a la vista de partidas multijugador pendientes y completadas. En este caso, se ha desarrollado el

siguiente boceto (Figura 99) en Figma. El boceto muestra la información de los Wordles completados en cartas, y dichas cartas contienen la palabra adivinar, si se ha adivinado o no (cambiando el color del resultado) y otros parámetros adicionales como el tiempo consumido, los intentos realizados y la experiencia obtenida. Los estilos han seguido la misma filosofía que los diseños anteriores.

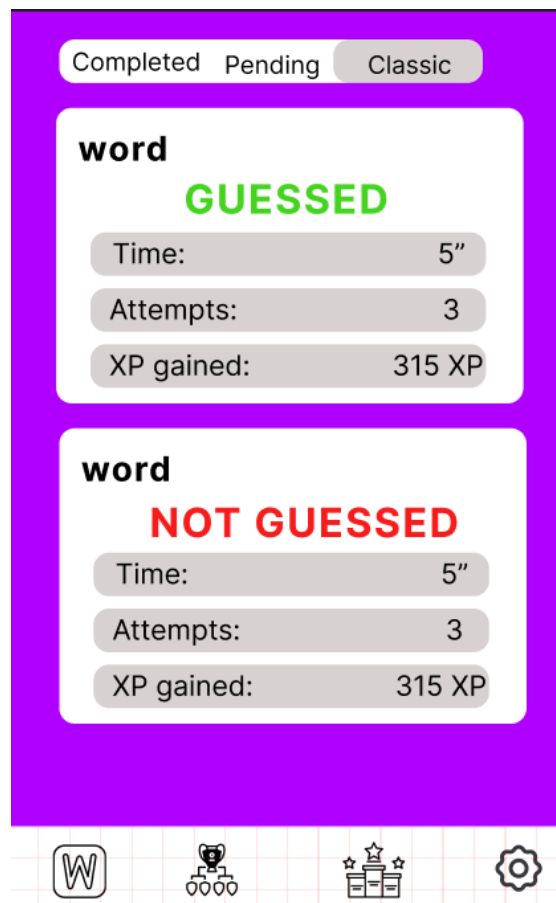


Figura 99. Boceto de Wordles completados.

En cuanto a la maquetación HTML, se sigue la misma filosofía que en el resto de los listados realizados: se obtienen las partidas del jugador, se almacenan en una lista y en HTML itera sobre esta lista, mostrando la información usando diferentes etiquetas (Figura 100).

```
<ion-card *ngFor="let game of completedWordles">
  <ion-card-header>
    {{ game.word }}
  </ion-card-header>
  <ion-card-content>
    <ion-list class="game-info-list">
      <ion-item>
        <ion-label class="result" [class.victory-result]="game.win"
[class.defeat-result]="!game.win">
          {{ game.win ? 'Guessed' : 'Not guessed' }}
        </ion-label>
      </ion-item>
    </ion-list>
  </ion-card-content>
</ion-card>
```

Figura 100. HTML del listado de Wordles completados.

Finalmente, la vista ha sido completada con el siguiente resultado (Figura 101):

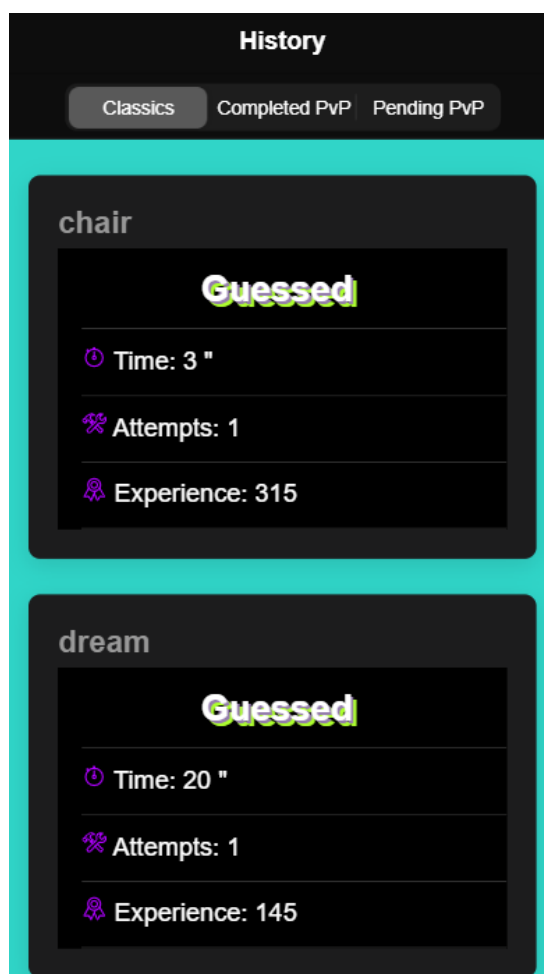


Figura 101. Listado de Wordles completados.

Más información acerca de este desarrollo puede consultarse en: <https://github.com/davidcr01/WordlePlus/issues/61>

III.3.10.2. Ranking

Para el *ranking*, se ha diseñado el siguiente boceto en Figma (Figura 102), en donde se muestra el filtro del ranking en una barra de herramientas, una primera fila con los valores a mostrar (nombre y parámetros del jugador), y el listado de estos jugadores.





	Wins	PVP	Tourn.	XP
Name				
player1	7	3	1	2500
player2	6	2	3	2500
player3	4	3	1	500
player4	4	3	0	0
player5	4	1	0	1500
player6	3	4	0	400
player7	3	1	0	1500
player8	2	0	0	2500
player9	1	3	0	2500
player10	0	0	1	2570
player11	0	2	1	2500

Figura 102. Boceto del *ranking*.

Para obtener este listado, se ha implementado un nuevo método GET en el *viewset* de los jugadores denominado *ranking* que obtiene 15 jugadores ordenados en orden descendente y filtrados por un parámetro opcional en la ruta. Un ejemplo de esta ruta podría ser <http://<host>:<port>/api/players/ranking/?filter=<parameter>>. Esta vista hace uso del serializador de Jugadores implementado en el primer Sprint.

En cuanto al frontend (página *tab3*), la filosofía es idéntica al resto de listados: barra de herramientas con el filtro, y en este caso se itera sobre una lista en donde cada fila representa la información del jugador. Cada vez que se pulsa en el filtro, se hace uso de la función *loadPlayers*, (Figura 103) que utiliza la URL mencionada anteriormente. Nótese como el parámetro del filtro se obtiene de la variable *selectedFilter*, que corresponde el filtro seleccionado de la barra de herramientas. Por defecto, el *ranking* se ordenará por el número de victorias.

```

async loadPlayers() {
  try {
    this.players = await
this.apiService.getPlayersRanking(this.selectedFilter) as any;
    console.log(this.players);
  } catch (error) {
    this.toastService.showToast("Error loading ranking", 2000, 'top',
'danger');
  }
}

```

Figura 103. Método para obtener los jugadores.

Finalmente, el *ranking* ha quedado con el siguiente resultado (Figura 104 y 105):

Wins W.PvP Tourmmts. xP					Wins W.PvP Tourmmts. xP				
Name	W	WIN	Trophy	xP	Name	W	WIN	Trophy	xP
player2	35	9	1	15414	player3	0	4	1	1000
player	19	4	0	6045	player2	35	9	1	15414
testing	2	4	0	630	player4	0	2	1	1000
testing5	0	0	0	0	testing5	0	0	0	0
testing6	0	0	0	0	testing6	0	0	0	0
testing7	0	0	0	0	testing7	0	0	0	0
player5	0	0	0	0	testing2	0	0	0	0
player3	0	4	1	1000	testing	2	4	0	630

Figuras 104 y 105. Resultado del *ranking*.

Más información acerca del desarrollo del ranking se encuentra en el siguiente enlace: <https://github.com/davidcr01/WordlePlus/issues/52#issuecomment-1633274498>

III.3.10.3. Revisión y retrospectiva

Como **revisión** del Sprint, las historias de usuario que se plantearon se han completado satisfactoriamente, siguiendo los suficientes criterios de seguridad y calidad. El resultado del incremento es satisfactorio, habiendo realizado un incremento funcional cubriendo todos los casos de error posibles. Al haber realizado pocos cambios, aún no se incorporará el código al producto final. La siguiente *release* se publicará cuando se implementen nuevos desarrollos.

Como **retrospectiva** del Sprint, éste ha durado menos tiempo del tiempo planificado. Al haber realizado varios listados similares, el desarrollo de este Sprint ha sido sencillo. Además, se ha utilizado el serializador de los jugadores que se definió en el primer Sprint, por lo que se ha acelerado el proceso de

desarrollo. Es fundamental intentar utilizar código desarrollado anteriormente en las nuevas funcionalidades para optimizar el tiempo invertido y obtener un producto con la máxima calidad posible.

III.3.11. Sprint 11

Nº	Objetivo	Fecha de entrega	
11	Eliminación de amigos. Perfiles de amigos.	03 de julio del 2023	
Ident.	Título	Estimación	Prioridad
HU.24	Como jugador quiero poder eliminar un jugador de mi lista de amigos	1	4
HU.25	Como jugador quiero poder ver el perfil de jugador de un amigo	2	4
HU.30	Como jugador quiero poder aceptar una solicitud de amistad	2	4

III.3.11.1. Eliminación de amigo y de solicitud de amistad

Para la eliminación de un amigo, se ha añadido un nuevo método *destroy* en el *viewset* de la lista de amigos. El siguiente método (Figura 106) comprueba la existencia del jugador a eliminar y del jugador que solicita la eliminación, además de que exista la relación de amistad.

```
def destroy(self, request, *args, **kwargs):
    player = getattr(request.user, 'player', None)
    friend_id = kwargs.get('pk')

    if not player:
        return Response({'error': 'Player not found'}, status=404)
    try:
        friend = Player.objects.get(id=friend_id)
    except Player.DoesNotExist:
        return Response({'error': 'Friend not found'}, status=404)

    if not FriendList.objects.filter(Q(sender=player, receiver=friend) |
    Q(sender=friend, receiver=player)).exists():
        return Response({'error': 'Player is not friends with this user'},
        status=403)
```

Figura 106. Eliminación de amigos.

En el *frontend*, en el listado de amigos se añadió un botón de eliminado de amigo. Para lograr la eliminación, se ha enlazado a dicho botón la función *confirmDeleteFriend* en la lógica de la página de la lista de amigos. Esta función despliega una alerta de confirmación, y si se acepta se elimina el amigo con la función *deleteFriend*, haciendo uso del método mencionado anteriormente (Figuras 107 y 108).

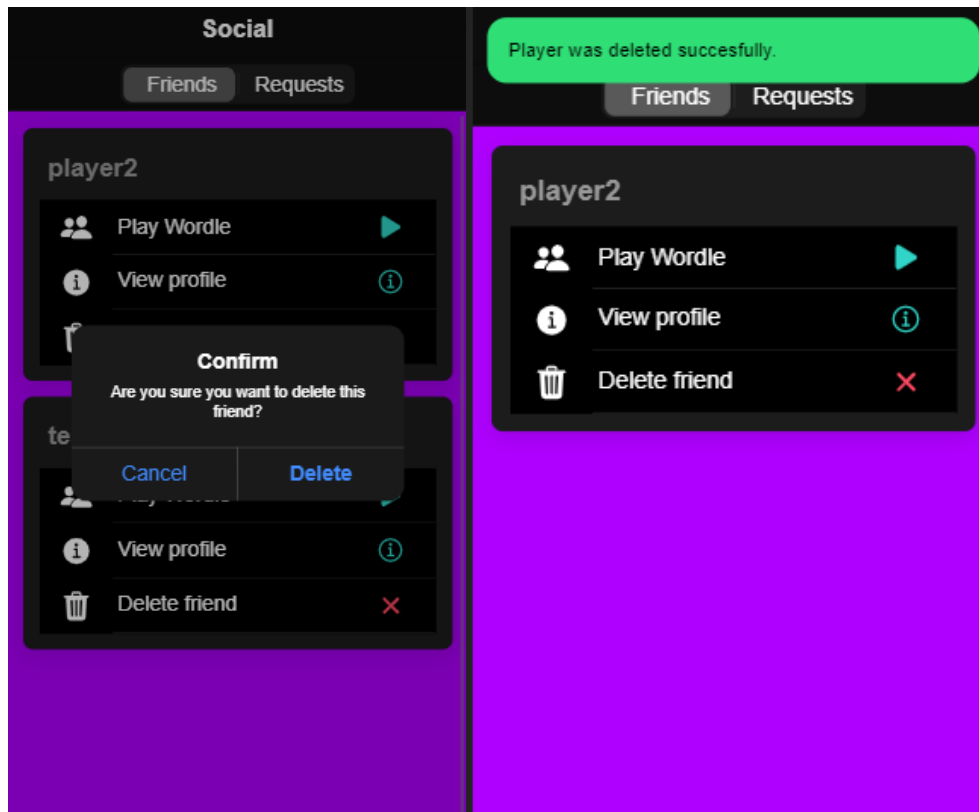


Figura 107 y 108. Alerta de confirmación y resultado de eliminación.

También se ha añadido la función *rejectFriendRequest*, que hace uso del método *reject* de las solicitudes de amistad que se implementó en el sprint 7. Una vez que el usuario pulsa sobre el botón de rechazar la petición de amistad, éstas se recargan, haciendo que la petición rechazada desaparezca de la lista (Figura 109).

```

async rejectFriendRequest(requestId: number) {
  (await this.apiService.rejectFriendRequest(requestId)).subscribe(
    async (response) => {
      console.log('Friend request rejected successfully', response);
      this.loadFriendRequests();
    },
    async (error) => {
      console.error('Error rejecting friend request:', error);
    }
  );
}

```

Figura 109. Método para rechazar peticiones de amistad.

Más información acerca de este desarrollo se encuentra en el siguiente enlace:

<https://github.com/davidcr01/WordlePlus/issues/55#issuecomment-1633292169>

III.3.11.2. Perfil de amigo

Para visualizar el perfil de jugador de un amigo se ha creado un nuevo componente *PlayerInfoPopover* que muestra la información asociada. En vez de realizar una nueva página, se ha desarrollado un *popover* al igual que se hizo con las notificaciones y con la selección de la longitud de palabra del Wordle (Figura 110).

El componente recibe como parámetro el identificador del jugador, obtiene su información asociada con el método de la API *getPlayerData* y la muestra en la página de la lista de amigos.

```
export class PlayerInfoPopoverComponent implements OnInit {
  @Input() playerId: string;
  playerInfo: any;
  constructor(private apiService: ApiService) {}

  ngOnInit() {
    this.loadPlayerData();
  }

  async loadPlayerData() {
    if (this.playerId) {
      (await this.apiService.getPlayerData(this.playerId)).subscribe(
        (response: any) => {
          this.playerInfo = response;
        },
        (error) => {
          console.error('Error retrieving player data:', error);
        })
    }
  }
}
```

Figura 110. Componente *PlayerInfoPopover*.

El resultado ha quedado de la siguiente manera (Figura 111):

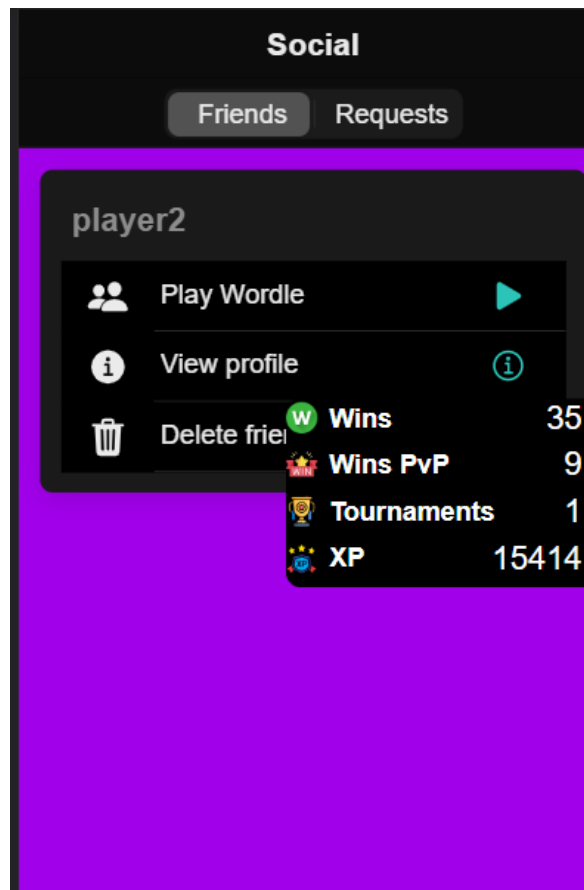


Figura 111. Resultado del perfil del amigo.

Más información acerca de este desarrollo se encuentra en el siguiente enlace: <https://github.com/davidcr01/WordlePlus/issues/54#issuecomment-1633347317>

III.3.11.3. Revisión y retrospectiva

Como **revisión** del Sprint, las historias de usuario que se plantearon se han completado satisfactoriamente, siguiendo los suficientes criterios de seguridad y calidad. El resultado del incremento es satisfactorio, habiendo realizado un incremento funcional cubriendo todos los casos de error posibles. Al haber acumulado suficientes cambios, este Sprint junto con el anterior se añadirán al producto final (Release v3.2.0).

Como **retrospectiva** del Sprint, éste ha durado menos tiempo del tiempo planificado. Al haber realizado gran parte de los controladores en Sprints anteriores, el desarrollo de este Sprint se ha visto reducido. El tiempo sobrante se ha dedicado a mejorar el diseño del *popover* para una mejor adaptabilidad a la resolución de pantalla, y a resolver posibles casos de error.

III.3.12. Sprint 12

Nº	Objetivo	Fecha de entrega	
12	Modo oscuro. Cambio de idioma	17 de julio del 2023	
Ident.	Título	Estimación	Prioridad
HU.19	Como usuario quiero poder cambiar el modo de visualización de la plataforma (modo predeterminado / modo oscuro)	3	5

El objetivo de este Sprint es permitir al usuario cambiar el modo de visualización de la plataforma, de modo claro a modo oscuro y viceversa. Al haber utilizado la tecnología *Ionic Framework*²⁸ en el *frontend*, este desarrollo se ha implementado automáticamente. Cuando el usuario cambia el modo de visualización del dispositivo, cambia el modo de visualización de la plataforma.

Las vistas en modo claro y oscuro se encuentran en el siguiente enlace:

<https://github.com/davidcr01/WordlePlus/issues/56>

III.3.12.1. Revisión y retrospectiva

Como **revisión** del Sprint, la historia de usuario que se planteó se ha completado satisfactoriamente, logrando unas vistas adaptables al modo de visualización del dispositivo. Al no haber nuevo código incorporado, el producto final permanecerá sin cambios (Release v3.2.0).

Como **retrospectiva** del Sprint, éste ha durado menos tiempo del tiempo planificado. Al haber escogido una tecnología que desarrollara automáticamente esta tarea, hemos tenido mayor tiempo para mejorar, refinar y pulir aspectos de diseño y rendimiento de la plataforma. El análisis exhaustivo que se realizó de las tecnologías disponibles ha permitido poder sacar el máximo provecho a las ventajas que éstas ofrecen.

CAPÍTULO IV. CONCLUSIONES Y

TRABAJOS FUTUROS

Tras completar todas las historias de usuario planteadas inicialmente, se ha logrado desarrollar un producto final de calidad y coherencia para el juego Wordle mejorado. La aplicación ha sido implementada en un entorno adaptable y flexible utilizando *Docker*³⁴, lo que ha permitido encapsular toda la aplicación en contenedores, facilitando su despliegue y gestión. Aunque la plataforma se haya desarrollado en local, el entorno *Docker*³⁴ brinda la posibilidad de subir y alojar la aplicación en la nube, como en AWS⁴¹. Esto ofrece beneficios como escalabilidad, alta disponibilidad y facilidad de despliegue en entornos distribuidos.

En cuanto a la interfaz de usuario, se ha logrado implementar una aplicación híbrida adaptable a dispositivos pequeños, como smartphones (tanto Android como iOS); dispositivos medianos como tablets; y dispositivos con resolución más grandes para ordenadores y portátiles.

En cuanto al *backend*, se ha logrado desarrollar una API independiente del frontend, lo que brinda independencia entre ambas partes y permite futuras mejoras y expansiones sin afectar la funcionalidad principal.

En cuanto a los trabajos futuros, se plantean varias áreas de mejora y expansión para el proyecto:

1. Mejoras de diseño en las vistas: Se pueden realizar mejoras en el diseño de las vistas para proporcionar una experiencia visual más atractiva y agradable para los usuarios. Esto incluye la selección de colores, tipografías, disposición de elementos y animaciones que mejoren la interfaz de usuario.
2. Mejoras de rendimiento en la API: Es posible optimizar el rendimiento de la API para garantizar una respuesta rápida y eficiente ante las solicitudes de los usuarios. Esto puede incluir técnicas de caching, optimización de consultas a la base de datos y escalado horizontal para manejar un mayor volumen de tráfico.
3. Implementación del multilinguaje: Se puede agregar soporte para varios idiomas en la aplicación, lo que permitiría a los usuarios disfrutar del juego en su idioma preferido. Esto requerirá la internacionalización de las cadenas de texto, la configuración de traducciones y la adaptación de las interfaces de usuario para admitir diferentes idiomas.
4. Sistema alternativo para obtener palabras aleatorias: Se puede explorar la implementación de un sistema diferente para obtener palabras aleatorias en el juego. Esto podría incluir la integración de un diccionario

más amplio, la aplicación de algoritmos de generación de palabras aleatorias o la incorporación de palabras específicas relacionadas con temas o categorías elegidas por los usuarios.

5. Desarrollo de un bot independiente: Se podría desarrollar un bot autónomo que pueda participar en partidas y torneos junto con otros jugadores. Este bot jugaría de forma independiente utilizando algoritmos y estrategias para competir con los jugadores humanos, brindando una experiencia desafiante y enriquecedora.
6. Habilitar la navegación cifrada SSL: Se recomienda habilitar la comunicación segura entre los componentes de la aplicación mediante HTTPS. Esto garantizará la confidencialidad y la integridad de los datos transmitidos, proporcionando una capa adicional de seguridad para los usuarios.

BIBLIOGRAFÍA

-
- ¹ GeeksForGeeks. *Component Based Software Engineering*. s.f. <https://www.geeksforgeeks.org/component-based-software-engineering/> (último acceso: 27 de marzo de 2023).
- ² IEEEExplore. «Microservices-based software architecture and approaches.» *ieeexplore.ieee.org*. 8 de junio de 2017. <https://ieeexplore.ieee.org/abstract/document/7943959> (último acceso: 27 de marzo de 2023).
- ³ IEEEExplore. «DockerSim: Full-stack simulation of container-based Software-as-a-Service (SaaS) cloud deployments and environments.» *ieeexplore.ieee.org*. 30 de noviembre de 2017. <https://ieeexplore.ieee.org/abstract/document/8121898> (último acceso: 23 de marzo de 2023).
- ⁴ GeeksForGeeks. *MVC Framework Introduction*. 6 de marzo de 2023. <https://www.geeksforgeeks.org/mvc-framework-introduction/> (último acceso: 27 de marzo de 2023).
- ⁵ RedHat. *¿Qué es la metodología ágil?* s.f. <https://www.redhat.com/es/devops/what-is-agile-methodology> (último acceso: 22 de febrero de 2023).
- ⁶ SA, ERIKA. *atlassian.com*. s.f. <https://www.atlassian.com/es/agile/scrum/scrum-metrics> (último acceso: 2 de febrero de 2023).
- ⁷ Kabanize. *¿Qué es Kanban?* s.f. <https://kanbanize.com/es/recursos-de-kanban/primeros-pasos/que-es-kanban#:~:text=Kanban%20es%20un%20m%C3%A9todo%20Lean,la%20eficiencia%20y%20mejorar%20continuamente>.
- ⁸ Unity. *¿Qué es CI/CD?* s.f. <https://unity.com/es/solutions/what-is-ci-cd#:~:text=La%20CI%2FCD%2C%20o%20integraci%C3%B3n,la%20entrega%20continua%20de%20c%C3%B3digo>.
- ⁹ Google. *developer.android.com*. s.f. <https://developer.android.com/studio> (último acceso: 25 de febrero de 2023).
- ¹⁰ Flutter. <https://flutter.dev/>. s.f. <https://flutter.dev/> (último acceso: 25 de febrero de 2023).
- ¹¹ IBM. *www.ibm.com*. 9 de mayo de 2019. <https://www.ibm.com/es-es/cloud/learn/lamp-stack-explained>.
- ¹² XenForo. *linux.org*. s.f. <https://www.linux.org/>.
- ¹³ Apache. *httpd.apache.org/*. s.f. <https://httpd.apache.org/> (último acceso: 25 de febrero de 2023).
- ¹⁴ Oracle. *mysql.com*. s.f. <https://www.mysql.com/> (último acceso: 25 de febrero de 2023).
- ¹⁵ PHP. *php.net*. s.f. <https://www.php.net/> (último acceso: 25 de febrero de 2023).
- ¹⁶ JavaScript. *javascript.com*. s.f. <https://www.javascript.com/> (último acceso: 25 de febrero de 2023).

-
- ¹⁷ MongoDB. *What is the MERN stack?* s.f. <https://www.mongodb.com/mern-stack#:~:text=MERN%20stands%20for%20MongoDB%2C%20Express,MongoDB%20%E2%80%94%20document%20database.>
- ¹⁸ Inc., MongoDB. *mongodb.com*. s.f. <https://www.mongodb.com/> (último acceso: 25 de febrero de 2023).
- ¹⁹ Foundation, OpenJS. *expressjs.com*. 2017. <https://expressjs.com/>. TypeScript. *typescriptlang.org*. s.f. <https://www.typescriptlang.org/> (último acceso: 14 de mayo de 2023).
- ²⁰ Platforms, Meta. *reactjs.org*. s.f. <https://es.reactjs.org/>.
- ²¹ Angular. *angular.io*. s.f. <https://angular.io/> (último acceso: 25 de febrero de 2023).
- ²² Foundation, OpenJS. *nodejs.org*. s.f. <https://nodejs.org/en/> (último acceso: 25 de febrero de 2023).
- ²³ Dart. *dart.dev*. s.f. <https://dart.dev/> (último acceso: 25 de febrero de 2023).
- ²⁴ Native, React. <https://reactnative.dev/>. s.f.
- ²⁵ Apple. *Objective-C*. s.f. <https://developer.apple.com/library/archive/documentation/Cocoa/Conceptual/ProgrammingWithObjectiveC/Introduction/Introduction.html>.
- ²⁶ Inc, Apple. *swift*. s.f. <https://www.apple.com/es/swift/>.
- ²⁷ Oracle. *java.com*. s.f. <https://www.java.com/es/> (último acceso: 25 de febrero de 2023).
- ²⁸ Ionic. <https://ionicframework.com/>. s.f.
- ²⁹ You, Evan. *vuejs.org*. s.f. <https://vuejs.org/> (último acceso: 25 de febrero de 2023).
- ³⁰ PostgreSQL. *postgresql.org*. s.f. <https://www.postgresql.org/> (último acceso: 25 de febrero de 2023).
- ³¹ IBM. *ibm.com*. s.f. <https://www.ibm.com/es-es/topics/rest-apis> (último acceso: 26 de junio de 2023).
- ³² Python. *python.org*. s.f. <https://www.python.org/> (último acceso: 25 de febrero de 2023).
- ³³ Framework, Django Rest. *django-rest-framework.org*. s.f. <https://www.django-rest-framework.org/> (último acceso: 25 de febrero de 2023).
- ³⁴ Docker. *docker.com*. s.f. <https://www.docker.com/> (último acceso: 25 de febrero de 2023).
- ³⁵ GitHub. *github.com*. s.f. github.com (último acceso: 26 de junio de 2023).
- ³⁶ LucidChart. *Qué es un diagrama entidad-relacion*. s.f. <https://www.lucidchart.com/pages/es/que-es-un-diagrama-entidad-relacion> (último acceso: 13 de febrero de 2023).
- ³⁷ Microsoft. *Fundamentos de la normalización de bases de datos*. s.f. <https://learn.microsoft.com/es-es/office/troubleshoot/access/database-normalization-description> (último acceso: 16 de marzo de 2023).
- ³⁸ IBM. *Tercera forma normal*. s.f. <https://www.ibm.com/docs/es/db2-for-zos/11?topic=ssepek-11-0-0-intro-src-tpc-db2z-thirdnormalform-html> (último acceso: 21 de marzo de 2023).
- ³⁹ NGINX. *nginx.com*. s.f. <https://www.nginx.com/> (último acceso: 13 de mayo de 2023).

⁴⁰ TypeScript. *typescriptlang.org*. s.f. <https://www.typescriptlang.org/> (último acceso: 14 de mayo de 2023).

⁴¹ Amazon. *aws.amazon.com*. s.f. <https://aws.amazon.com/es/>.