



TRABAJO FIN DE GRADO

INGENIERÍA INFORMÁTICA

Wordle+

Autor

David Correa Rodríguez

Director

Juan Manuel Fernández Luna



Escuela Técnica Superior de Ingenierías Informática y
de Telecomunicación

Granada, Julio de 2023

Índice

Wordle+	1
Capítulo I. Introducción	3
I.1. Motivación. (TODO)	3
I.2. Objetivos. (TODO)	3
I.3. Descripción de la aplicación.....	3
I.4. Estado del arte.	5
I.4.1. Aplicaciones web multiplataforma.....	5
I.4.2. Metodologías y enfoques	6
I.4.3 Proyectos similares	7
I.5. Descripción de la memoria (TODO).	8
Capítulo II. Historias de usuario. Planificación y presupuesto	9
II.1. Descripción de la metodología.	9
II.2. Historias de usuario.	11
II.3 Sprints.	24
II.3.1. Planificación de Sprints.....	24
II.3.2. División en tareas.	29
II.4. Presupuesto. (TODO)	41
Capítulo III. Desarrollo.....	41
III.1. Elección de las tecnologías.....	41
III.1.1. Frontend.....	41
III.1.2. Backend.	43
III.1.3. Despliegue.	45
III.1.4. Control de versiones y seguimiento del Sprint.	45
III.2. Diseño de la base de datos.....	46
III.2.1. Análisis del contexto.	46
III.2.2. Diagrama Entidad-Relación.	48
III.2.3. Paso a tablas.	51
III.2.4. Fusiones.	53
III.2.5. Normalización.	57
III.2.5.1. Primera Formal Normal (1FN).	57
III.2.5.2. Segunda Formal Normal (2FN).....	59
III.2.5.3. Tercera Formal Normal (3FN).....	61
III.2.5.4. Forma Normal de Boyce-Codd (FNBC).	63
III.3. Implementación de los Sprints.	65
III.3.1. Sprint 1.....	65
III.3.1.1. Infraestructura.....	65

III.3.1.2. Registro de usuarios	71
III.3.1.3. Pruebas y resultado. Revisión.	78
III.3.2. Sprint 2.....	80
III.3.2.1. Inicio y cierre de sesión	80
III.3.2.2. Pruebas y resultado. Revisión.	84
Capítulo IV. Conclusiones y trabajos futuros (todo).	85
Bibliografía y referencias.	86

CAPÍTULO I. INTRODUCCIÓN

I.1. Motivación. (TODO)

I.2. Objetivos. (TODO)

I.3. 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 gestores de eventos.
- 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: leyenda, maestro, veterano, principiante, iniciado), 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 gestores de eventos 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
- Implementación de un bot que juegue automáticamente a las partidas, con la posibilidad de añadirlo a los torneos y comprobar su comportamiento.
- Los torneos se dividirán según los rangos de los jugadores para que la experiencia de juego sea justa y los enfrentamientos sean equilibrados.
- Los torneos, además de dividirse por rango, también se dividirán por longitud de palabras (Torneo de maestros de 7 letras, torneo de iniciados de 4 letras, etc)

Nivel 3:

- Mejoras de rendimiento
- 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 única funcionalidad existente dicho es la segunda funcionalidad del primer nivel (Poder jugar la partida clásica de Wordle) y la penúltima del nivel 3 (Implementar un modo oscuro)

I.4. Estado del arte.

Esta sección del documento describe el estado actual de la tecnología, herramientas y metodologías relevantes para el desarrollo de aplicaciones web multiplataforma y los proyectos relacionados con la plataforma propuesta.

I.4.1. Aplicaciones web multiplataforma

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, tablets 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.

1.4.2. 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: este enfoque 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: en este enfoque, 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: en este enfoque, 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: este enfoque 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.

En el caso de este proyecto, se utilizarán todos los enfoques mencionados excepto el enfoque basado en servicios:

- Basado en componentes: todos los elementos de *frontend* definidos en *Ionic*²⁷ que puedan reutilizarse se definirán como componentes, evitando la duplicación de código.
- Metodología ágil⁵: el proyecto utiliza la metodología ágil de Scrum para realizar un seguimiento y valoración del trabajo conseguido dividido en diferentes *Sprints*, permitiendo una entrega continua del proyecto.
- Basado en contenedores: el proyecto en su totalidad se desplegará en contenedores para realizar una abstracción del sistema sobre el que se despliega, permitiendo la orquestación de los diferentes componentes.
- Basado en MVC: en este caso, el modelo es la base de datos *PostgreSQL*²⁹, la vista es el framework *Ionic*²⁷ y el controlador es el framework *Django REST API*³¹.

Respecto a las tecnologías relacionadas, véase el documento de [Tecnologías y herramientas](#).

En resumen, existen múltiples recursos disponibles para el desarrollo de aplicaciones web y móviles multiplataforma, desde libros y guías prácticas hasta frameworks de desarrollo y herramientas de programación específicas. Sin embargo, actualmente apenas existen proyectos que integren el framework *Ionic*²⁷ como *frontend* y *Django REST framework*³¹ como *backend* que estén en desarrollo o que sean relativamente recientes y lo suficientemente completos.

1.4.3 Proyectos similares

Concretando en la temática del proyecto, existen pocos juegos desarrollados utilizando las tecnologías mencionadas, además de que el juego base (Wordle) presenta una jugabilidad y personalización muy básica. Wordle fue desarrollado principalmente utilizando JavaScript¹⁶, HTML y CSS como juego de que se ejecuta en el navegador.

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, reimplementando 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.

I.5. Descripción de la memoria (TODO).

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 del trabajo que tiene que realizar el propietario del producto, en donde se incluyen requisitos, funcionalidades y mejoras del producto. Esta 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. También se le suele llamar Epic. 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 del 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 unidad de medida utilizada en proyectos ágiles para estimar la carga global de una historia de usuario. En este caso, se ha utilizado como referencia la escala de Fibonacci.

Ident.	Título	Estimación	Prioridad
HU.1	Como jugador quiero poder registrarme	2	1
HU.2	Como gestor de eventos quiero poder registrarme	2	1
HU.3	Como usuario (jugador y gestor) quiero poder iniciar sesión	2	1
HU.4	Como usuario quiero poder cerrar sesión	1	1
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
HU.7	Como gestor de eventos quiero crear una sala abierta de un torneo	5	2
HU.8	Como GE quiero poder seleccionar el tamaño del torneo (número de participantes)	2	2
HU.9	Como GE quiero poder seleccionar la longitud de la palabra del torneo	2	2
HU.10	Como GE 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.12	Como jugador quiero poder seleccionar la longitud de la palabra del torneo (tipo de torneo)	3	2
HU.13	Como jugador quiero poder unirme a una sala abierta de un torneo	2	2

HU.1 4	Como usuario quiero poder modificar mis datos personales	2	3
HU.1 5	Como jugador quiero poder ver el ranking	5	3
HU.1 6	Como jugador quiero poder seleccionar el filtro del ranking	2	3
HU.1 7	Como jugador quiero poder ver mi perfil de jugador	2	3
HU.1 8	Como usuario quiero poder cambiar el idioma de la plataforma	2	4
HU.1 9	Como jugador quiero poder ver mi lista de amigos	3	2
HU.2 0	Como jugador quiero poder buscar a otro jugador por su nombre y mandarle una solicitud de amistad	3	2
HU.2 1	Como jugador quiero poder aceptar una solicitud de amistad	2	2
HU.2 2	Como jugador quiero poder eliminar un jugador de mi lista de amigos	1	4
HU.2 3	Como jugador quiero poder ver el perfil de jugador de un amigo	2	4
HU.2 4	Como usuario quiero poder cambiar el modo de visualización de la plataforma (modo predeterminado / modo oscuro)	3	5
HU.2 5	Como jugador quiero poder ver mis notificaciones a través del buzón	2	2

Identificador: HU1	Registro jugador
Descripción: Como jugador quiero poder registrarme	

Estimación 2	Prioridad 1	Entrega
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: HU2	Registro gestor de eventos	
Descripción: Como gestor de eventos quiero poder registrarme		
Estimación 2	Prioridad 1	Entrega
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: HU3	Iniciar sesión
Descripción: Como usuario (jugador y gestor) quiero poder iniciar sesión	

Estimación 2	Prioridad 1	Entrega
Pruebas de aceptación: <ul style="list-style-type: none"> • Introducir un nickname 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: HU4	Cerrar sesión	
Descripción: Como usuario quiero poder cerrar sesión		
Estimación 1	Prioridad 1	Entrega
<p>Pruebas de aceptación:</p> <ul style="list-style-type: none">Comprobar que el usuario no se encuentra en la lista de usuarios conectados en el sistema		
Observaciones:		

Identificador: HU5	Wordle clásico	
Descripción: Como jugador quiero poder jugar un Wordle clásico		
Estimación 8	Prioridad 1	Entrega
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: HU6	Longitud palabra Wordle clásico	
Descripción: Como jugador quiero poder seleccionar la longitud de la palabra de la partida clásica		
Estimación 2	Prioridad 2	Entrega
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: HU7	Crear sala torneo	
Descripción: Como gestor de eventos quiero crear una sala abierta de un torneo		
Estimación 5	Prioridad 2	Entrega
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• error• Introducir todos los campos correctamente y que se crea un nuevo torneo en la lista de torneos disponibles en el sistema		
Observaciones:		

Identificador: HU8		Tamaño torneo
Descripción: Como GE quiero poder seleccionar el tamaño del torneo (número de participantes)		
Estimación 2	Prioridad 2	Entrega
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: HU9	Tamaño palabra torneo	
Descripción: Como GE quiero poder seleccionar la longitud de la palabra del torneo		
Estimación 2	Prioridad 2	Entrega
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: HU10	Crear torneo fijo	
Descripción: Como GE quiero poder crear un torneo y preseleccionar los participantes		
Estimación 3	Prioridad 2	Entrega
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• 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: HU11		Crear partida 1vs1	
Descripción: Como jugador quiero poder jugar una partida 1vs1 con un amigo			
Estimación 5		Prioridad 2	Entrega
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: HU12	Seleccionar longitud palabra torneo jugador	
Descripción: Como jugador quiero poder seleccionar la longitud de la palabra del torneo (tipo de torneo)		
Estimación 3	Prioridad 2	Entrega
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: HU13	Unirse a sala	
Descripción: Como jugador quiero poder unirme a una sala abierta de un torneo		
Estimación 2	Prioridad 2	Entrega
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: HU14	Modificar datos personales	
Descripción: Como usuario quiero poder modificar mis datos personales		
Estimación 2	Prioridad 3	Entrega
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: HU15		Ver ranking
Descripción: Como jugador quiero poder ver el ranking		
Estimación 5	Prioridad 3	Entrega
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: HU16	Cambiar filtro ranking		
Descripción: Como jugador quiero poder seleccionar el filtro del ranking			
Estimación 2	Prioridad 3	Entrega	
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: HU17	Ver perfil jugador	
Descripción: Como jugador quiero poder ver mi perfil de jugador		
Estimación 2	Prioridad 3	Entrega
Pruebas de aceptación: <ul style="list-style-type: none">Comprobar que al usuario se le muestran los valores almacenados en el sistema		
Observaciones:		

Identificador: HU18	Seleccionar idioma	
Descripción: Como usuario quiero poder cambiar el idioma de la plataforma		
Estimación 2	Prioridad 4	Entrega
Pruebas de aceptación: <ul style="list-style-type: none">• Comprobar que al usuario se le muestran toda la plataforma en el idioma seleccionado• Comprobar que el selector de idioma ha cambiado y aparece el idioma seleccionado		
Observaciones:		

Identificador: HU19	Lista amigos	
Descripción: Como jugador quiero poder ver mi lista de amigos		
Estimación 3	Prioridad 2	Entrega
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: HU20	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

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: HU21		Aceptar solicitud amistad
Descripción: Como jugador quiero poder aceptar una solicitud de amistad		
Estimación 2	Prioridad 2	Entrega
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: HU22		Eliminar amigo
Descripción: Como jugador quiero poder eliminar un jugador de mi lista de amigos		
Estimación 1	Prioridad 4	Entrega
Pruebas de aceptación: <ul style="list-style-type: none"> • Comprobar que el usuario eliminado ya no forma parte de la lista de amigos • Comprobar que a ese usuario se le puede enviar una solicitud de amistad 		
Observaciones:		

Identificador: HU23		Ver perfil amigo
Descripción: Como jugador quiero poder ver el perfil de jugador de un amigo		
Estimación 2	Prioridad 4	Entrega
Pruebas de aceptación: <ul style="list-style-type: none"> Comprobar que al usuario se le muestran los atributos almacenados del jugador seleccionado 		
Observaciones:		

Identificador: HU24		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
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: HU25		Ver buzón
Descripción: Como jugador quiero poder ver mis notificaciones a través del buzón		
Estimación 2	Prioridad 2	Entrega
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.		

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	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

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

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

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

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

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

Nº	Objetivo	Fecha de entrega	
-----------	-----------------	-------------------------	--

7	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

Nº	Objetivo	Fecha de entrega	
8	Inscripción al torneo	22 de mayo del 2023	
Ident.	Título	Estimación	Prioridad
HU.12	Como jugador quiero poder seleccionar la longitud de la palabra del torneo (tipo de torneo)	3	2
HU.13	Como jugador quiero poder unirme a una sala abierta de un torneo	2	2

Nº	Objetivo	Fecha de entrega	
9	Visualización del ranking	05 de junio del 2023	
Ident.	Título	Estimación	Prioridad
HU.15	Como jugador quiero poder ver el ranking	5	3
HU.16	Como jugador quiero poder seleccionar el filtro del ranking	2	3

Nº	Objetivo	Fecha de entrega	
10	Perfil de jugador. Solicitudes de amistad. Lista de amigos.	19 de junio del 2023	
Ident.	Título	Estimación	Prioridad

HU.19	Como jugador quiero poder ver mi lista de amigos	3	2
HU.20	Como jugador quiero poder buscar a otro jugador por su nombre y mandarle una solicitud de amistad	3	2
HU.21	Como jugador quiero poder aceptar una solicitud de amistad	2	2

Nº	Objetivo	Fecha de entrega	
11	Eliminación de amigos. Perfiles de amigos. Cambio de idioma	03 de julio del 2023	
Ident.	Título	Estimación	Prioridad
HU.22	Como jugador quiero poder eliminar un jugador de mi lista de amigos	1	4
HU.23	Como jugador quiero poder ver el perfil de jugador de un amigo	2	4
HU.18	Como usuario quiero poder cambiar el idioma de la plataforma	2	4

Nº	Objetivo	Fecha de entrega	
12	Modo oscuro	17 de julio del 2023	
Ident.	Título	Estimación	Prioridad
HU.24	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.

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: HU1		Registro jugador
Identificador	Título de la tarea	
Tarea 1-1 X	Definir el modelo de jugador en la BBDD	
Tarea 1-2 X	Implementación la inserción automática del jugador en la BBDD	
Tarea 1-3 X	Definir el modelo de usuario en la BBDD	
Tarea 1-4 X	Implementación la inserción automática del usuario en la BBDD	
Tarea 1-5 X	Definir el controlador asociado al registro del jugador	
Tarea 1-6 x	Diseño e implementación de la IU asociada a la creación del jugador	

Observaciones:**Identificador:** HU2

Registro del GE

Identificador	Título de la tarea
Tarea 2-1 x	Definir el modelo del GE en la BBDD
Tarea 2-2 x	Implementación la inserción automática del GE en la BBDD
Tarea 2-3 x	Definir el controlador asociado al registro del GE
Tarea 2-4	Diseño e implementación de la IU asociada a la creación del GE

Observaciones:

- El modelo de usuarios ya está creado.

Identificador: HU3

Iniciar sesión

Identificador	Título de la tarea
Tarea 3-1	Definir el controlador asociado al inicio de sesión
Tarea 3-2 x	Diseño e implementación de la IU asociada al inicio de sesión

Observaciones:

- El modelo de usuario está creado

Identificador: HU4	Cerrar sesión				
<table> <tr> <th>Identificador</th><th>Título de la tarea</th></tr> <tr> <td>Tarea 4-1</td><td>Definir el controlador asociado al cierre de sesión</td></tr> </table>		Identificador	Título de la tarea	Tarea 4-1	Definir el controlador asociado al cierre de sesión
Identificador	Título de la tarea				
Tarea 4-1	Definir el controlador asociado al cierre de sesión				
Observaciones: <ul style="list-style-type: none"> - El modelo de usuario está creado 					

Identificador: HU5	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> </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
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						
Observaciones:							

Identificador: HU6	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> <tr> <td>Tarea 6-2</td><td>Diseño e implementación de la IU asociada a la selección de la 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	Tarea 6-2	Diseño e implementación de la IU asociada a la selección de la 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: HU7

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 GEs.

Identificador: HU8

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 GEs.

Identificador: HU9	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: <ul style="list-style-type: none"> - Vista apta solo para GEs. 	

Identificador: HU10	Crear torneo fijo
Identificador	Título de la tarea
Tarea 7-1	Definir el controlador asociado a la creación del torneo
Tarea 7-2	Enviar notificación correspondiente a los jugadores seleccionados.
Tarea 7-3	Diseño e implementación de la IU asociada a la creación del torneo fijo
Observaciones: <p>El modelo del torneo ya está creado. Vista apta solo para GEs.</p>	

Identificador: HU11	Crear partida 1vs1
Identificador	Título de la tarea
Tarea 11-1	Definir el controlador asociado a la creación de la partida 1vs1
Tarea 11-2	Diseño e implementación de la IU asociada a la partida 1vs1
Tarea 11-3	Enviar notificación correspondiente al jugador.
Observaciones: Se completará cuando se implemente la lista de amigos. El modelo del jugador ya está creado.	

Identificador: HU12	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.
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: HU13	Unirse a sala
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.

Observaciones:

El modelo de las salas ya está creado.

El modelo de los jugadores ya está creado.

Identificador: HU14

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: HU15

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: HU16

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.
Observaciones:	

Identificador: HU17	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: HU18	Seleccionar idioma
Identificador	Título de la tarea
Tarea 18-1	Definir el controlador asociado a cambiar el idioma de la aplicación
Observaciones:	

Identificador: HU19	Lista amigos
Identificador	Título de la tarea
Tarea 19-1	Definir el controlador asociado a obtener la lista de amigos.
Tarea 19-2	Diseño e implementación de la IU asociada a la lista de amigos.
Observaciones:	
- El modelo de los jugadores ya está creado.	

Identificador: HU20	Añadir amigo
Identificador	Título de la tarea
Tarea 20-1	Definir el controlador asociado a añadir un amigo.
Tarea 20-2	Diseño e implementación de la IU asociada a añadir a un amigo (botón en lista de amigos)
Tarea 20-3	Enviar notificación correspondiente al jugador.
Observaciones:	
- El modelo de los jugadores ya está creado.	

Identificador: HU21	Aceptar solitud amistad
Identificador	Título de la tarea
Tarea 21-1	Definir el controlador asociado a aceptar a un amigo.
Tarea 21-2	Diseño e implementación de la IU asociada a aceptar a un amigo (botón)
Tarea 21-3	Enviar notificación correspondiente al jugador.
Observaciones: <ul style="list-style-type: none"> - El modelo de los jugadores ya está creado. 	

Identificador: HU22	Eliminar amigo
Identificador	Título de la tarea
Tarea 22-1	Definir el controlador asociado a eliminar a un amigo
Tarea 22-2	Diseño e implementación de la IU asociada a eliminar a un amigo (botón)
Observaciones: <ul style="list-style-type: none"> - El modelo de los jugadores ya está creado. 	

Identificador: HU23		Ver perfil amigo
Identificador	Título de la tarea	
Tarea 22-1	Definir el controlador asociado a ver el perfil de un amigo	
Tarea 22-2	Diseño e implementación de la IU asociada a ver el perfil de un amigo (botón)	
Observaciones: El modelo de los jugadores ya está creado. La IU asociada a ver el perfil de un jugador ya está creada		

Identificador: HU24		Cambiar modo visualización
Identificador	Título de la tarea	
Tarea 24-1	Definir el controlador asociado al cambio de visualización	
Tarea 24-2	Diseño e implementación de la IU asociada al cambio de visualización (botón)	
Observaciones:		

Identificador: HU25		Ver buzón
Identificador	Título de la tarea	
Tarea 25-1	Definir el modelo de las Notificaciones en la BBDD	
Tarea 25-2	Implementación la inserción automática de las notificaciones en la BBDD	
Tarea 25-3	Definir el controlador asociado al buzón	
Tarea 25-4	Diseño e implementación de la IU asociada al buzón.	
Observaciones:		

II.4. Presupuesto. (TODO)

CAPÍTULO III. DESARROLLO

III.1. Elección de las tecnologías.

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; Express como gestor de peticiones HTTP; React¹⁹ o Angular²⁰ como herramienta principal para el *frontend*; y Node.js²¹ 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 (MERN¹³) 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, Flutter⁶ 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 Dart²², por lo que se aleja del uso de tecnologías web, que es uno de los objetivos de este proyecto.

- **React Native**²³: se presenta como otra alternativa interesante, en donde el lenguaje principal es JavaScript, con posibilidad de escribir módulos en Objective-C²⁴, Swift²⁵ o Java²⁶. Sin embargo, como ya he comentado anteriormente, he realizado un proyecto utilizando React¹⁵, por lo que el aprendizaje que me aportaría esta alternativa se vería reducido.
- **Ionic Framework**²⁷: 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 contruidos. 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**²³, 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: React¹⁵, Angular¹⁶ y Vue²⁸. 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 Ionic²³ utilizando el lenguaje de programación Angular¹⁶.

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**²⁹, 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 Docker³², 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 REST, 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 REST 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 Express, es uno de los métodos más utilizados para crear APIs. Sin embargo, esta alternativa ya ha sido utilizada en el proyecto anteriormente mencionado.
- **Python**³⁰: uno de los lenguajes multiparadigma más utilizados en el panorama actual, permite construir APIs de forma sencilla mediante el framework **Django REST framework**³¹. 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 (CICD⁴) 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**³² 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.

III.1.4. Control de versiones y seguimiento del Sprint.

GitHub es sin duda alguna el sitio 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 GitHub, 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 GitHub de formas distintas.

El sistema propuesto es crear varios hitos o *milestones* en GitHub, que representen las funcionalidades del proyecto, como por ejemplo “registro de usuarios”, “login de usuarios”, “Wordle clásico”, y a dichos hitos asociarles las respectivas Issues y Pull Request. De esta manera, se permite un seguimiento mucho más óptimo y una documentación más extensa del progreso del proyecto, teniendo en un mismo lugar los cambios realizados en el proyecto y su seguimiento asociado. Esta información ha sido extraída de:

<https://docs.github.com/es/issues/using-labels-and-milestones-to-track-work/about-milestones>

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 gestores de eventos, 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 **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 gestores de eventos.
 - 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:

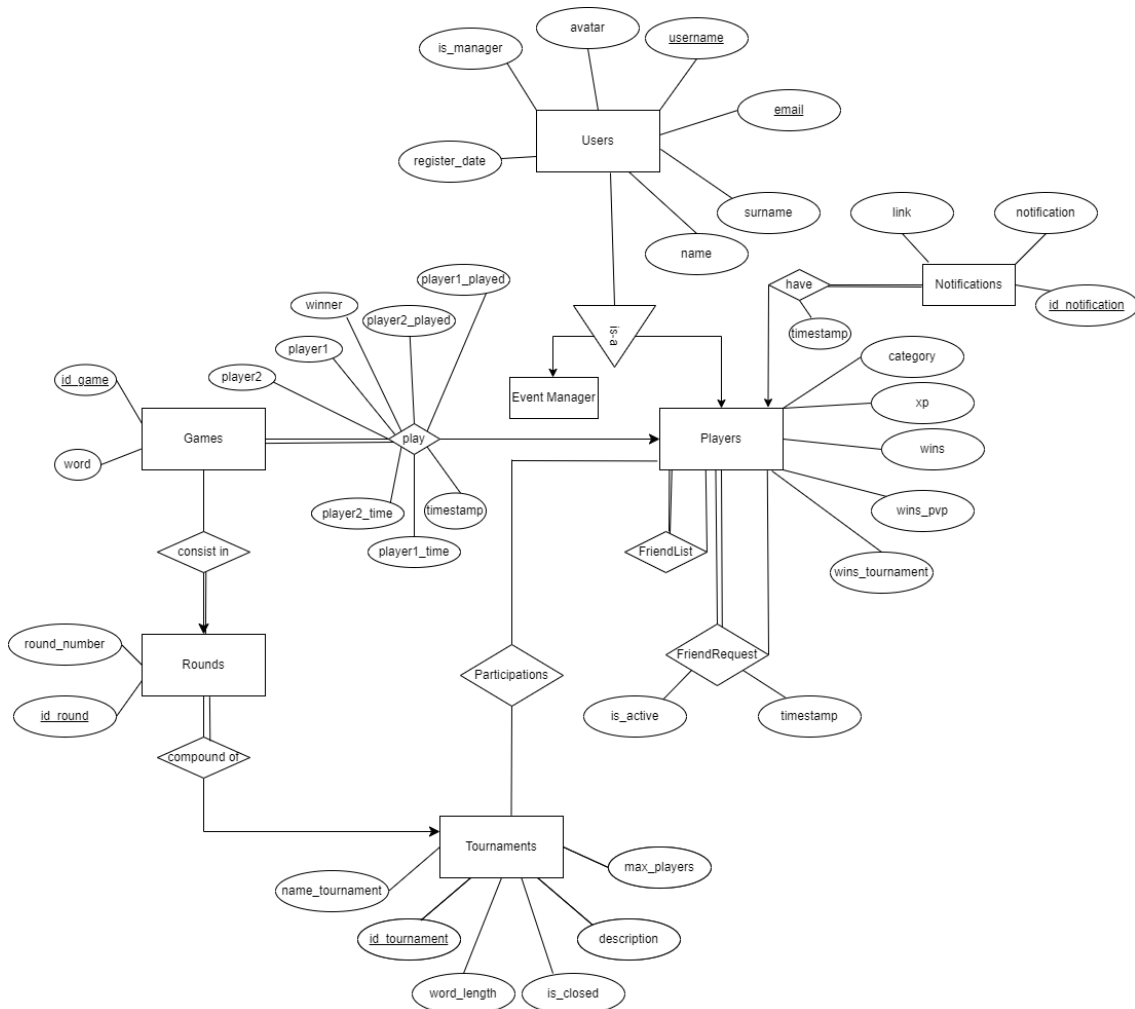
- Las entidades **Jugadores** y **Gestores de eventos** son entidades heredadas de **Usuarios**, por lo que tendrán una relación de herencia *es-un*.
- La entidad **Gestores de eventos** resulta innecesaria al tener un subconjunto (y solamente dicho subconjunto) de los campos de la entidad **Usuarios**.
- 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.³³

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



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 gestores de eventos en la base de datos puede hacerse con el campo *is_manager* 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:

Entidades:

<i>Users</i>
username - PK
email - UNIQUE
name
surname
register_date
is_manager
avatar

<i>Players</i>
username - PK, FK (Users)
email - UNIQUE, FK (Users)
wins
wins_pvp
wins_tournament
xp
category

<i>Games</i>
id_game - PK
word

<i>Notifications</i>
id_notification - PK
username - FK (Users)
text
link

<i>Tournaments</i>
id_tournament - PK
name_tournament
description
max_players
word_length
is_closed

<i>Rounds</i>
id_round - PK
round_number

Relaciones:

<i>FriendRequests</i>
sender - FK (Players)
receiver - FK (Players)
<i>PK (sender, receiver)</i>
timestamp
is_active

<i>FriendList</i>
sender - FK (Players)
receiver - FK (Players)
<i>PK (sender, receiver)</i>

<i>Participations</i>
username - FK (Players)
id_tournament - FK (Tournaments)
<i>PK (player, id_tournament)</i>

<i>Play</i>
id_game - FK (Games), PK
player1 - FK (Players)
player2 - FK (Players)
timestamp
player1_played
winner
player2_time
player1_time
player2_played

<i>Compound of</i>
id_round - PK, FK (Rounds)
id_tournament - FK (Tournaments)

<i>Consist in</i>
id_game - FK, PK (Games)
id_round - FK (Rounds)

<i>Have</i>
id_notification - FK, PK (Notifications)
timestamp

III.2.4. Fusiones.

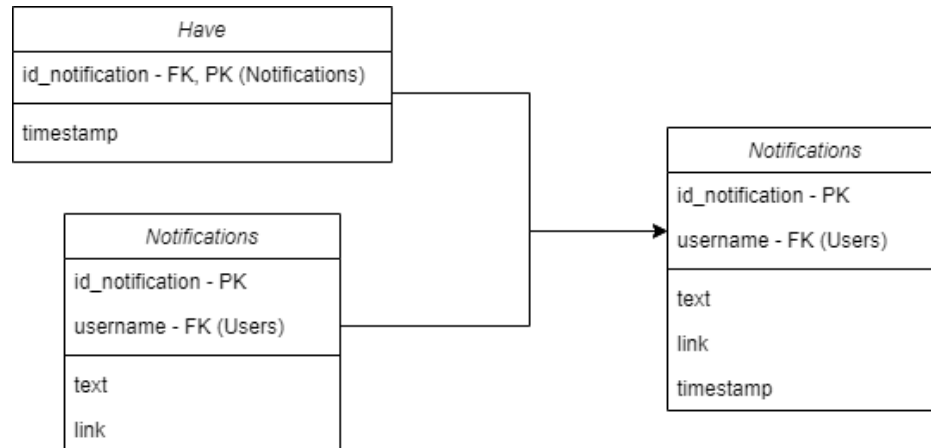
Las fusiones consisten en combinar varias tablas, permitiendo una reducción del número de estas, siempre y cuando no se pierda información (tanto de datos como de restricciones). Las fusiones también permiten mejoras de eficiencia a nivel de almacenamiento, es decir, se ahorra espacio de almacenamiento; 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:

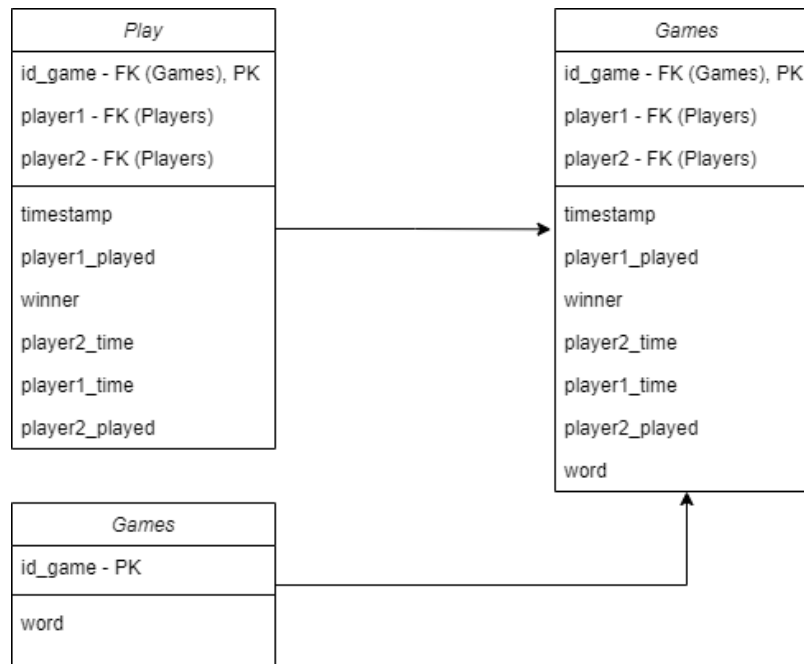
- Que las tablas tengan la misma clave primaria o candidata.
- Que ninguna de las tablas proceda de herencia.
- Que semánticamente la fusión tenga sentido.

1. La fusión entre las tablas *Users* y *Players* no puede darse, al pertenecer la tabla *Players* de una herencia de la Tabla *Users*.

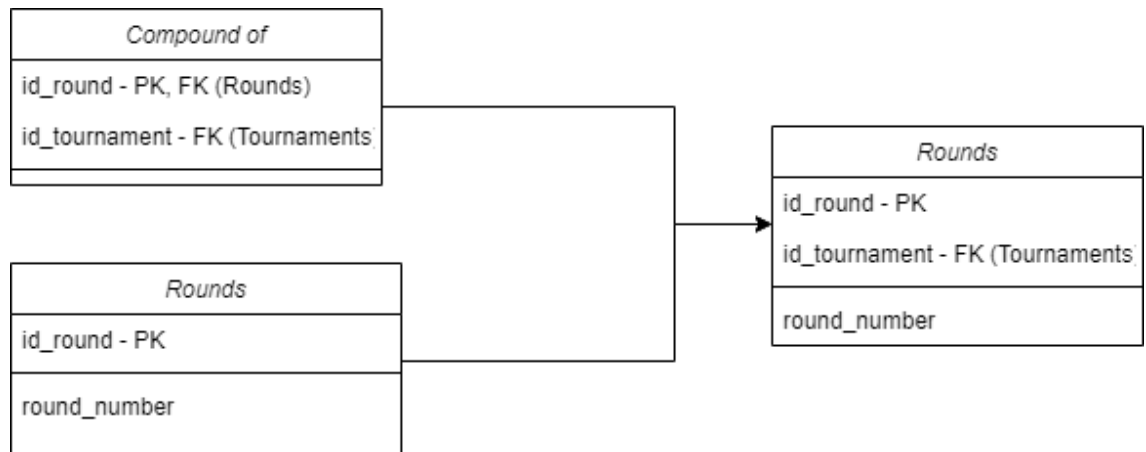
- La fusión entre las tablas *Have* y *Notifications* puede darse, al tener la misma clave primaria (*id_notification* de *Notifications*) y ser semánticamente compatibles, ya que una notificación siempre va a estar asociada a un jugador. Cabe mencionar que no se puede dar la fusión entre *Have* y *Players* ya que un jugador puede que no tenga notificaciones en ningún momento.



- 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*.
- 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.
- 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.



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.



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

<i>Users</i>
username - PK
email - UNIQUE
name
surname
register_date
is_manager

<i>Notifications</i>
id_notification - PK
username - FK (Users)
text
link
timestamp

<i>FriendRequests</i>
sender - FK (Players)
receiver - FK (Players)
<i>PK (sender, receiver)</i>
timestamp
is_active

<i>Players</i>
username - PK, FK (Users)
email - UNIQUE, FK (Users)
wins
wins_pvp
wins_tournament
xp
category
avatar

<i>Tournaments</i>
id_tournament - PK
name_tournament
description
max_players
word_length
is_closed

<i>FriendList</i>
sender - FK (Players)
receiver - FK (Players)
<i>PK (sender, receiver)</i>

<i>Participations</i>
username - FK (Players)
id_tournament - FK (Tournaments)
<i>PK (player, id_tournament)</i>

<i>Rounds</i>
id_round - PK
id_tournament - FK (Tournaments)
round_number

<i>Games</i>
id_game - PK
player1 - FK (Players)
player2 - FK (Players)
timestamp
player1_played
player2_played
winner
player1_time
player2_time
word

<i>Consists in</i>
id_game - FK, PK (Games)
id_round - FK (Rounds)

III.2.5. Normalización.

El proceso de normalización 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.³⁴

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.³⁴

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).

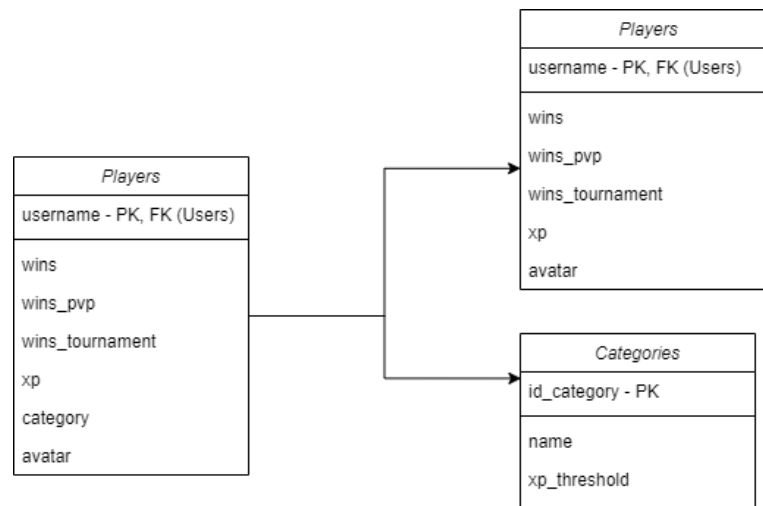
La condición necesaria para que una base de datos se encuentre en 3FN es que se encuentre en 2FN y, además, no contenga ninguna dependencia transitiva entre los atributos que no son clave.

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. ³⁵

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. 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, ya no existe la dependencia transitiva.

- **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:

<i>Users</i>
username - PK email - UNIQUE
name surname register_date is_manager

<i>Players</i>
username - PK, FK (Users) email - UNIQUE, FK (Users)
wins wins_pvp wins_tournament xp category avatar

<i>Rounds</i>
id_round - PK id_tournament - FK (Tournaments)
round_number

<i>Tournaments</i>
id_tournament - PK
name_tournament description max_players word_length is_closed

<i>Notifications</i>
id_notification - PK username - FK (Users)
text link timestamp

<i>Games</i>
id_game - PK player1 - FK (Players) player2 - FK (Players)
timestamp player1_played player2_played winner player1_time player2_time word

<i>FriendRequests</i>
sender - FK (Players) receiver - FK (Players) PK (sender, receiver)
timestamp is_active

<i>FriendList</i>
sender - FK (Players) receiver - FK (Players) PK (sender, receiver)

<i>Consists in</i>
id_game - FK, PK (Games) id_round - FK (Rounds)

<i>Participations</i>
username - FK (Players) id_tournament - FK (Tournaments) PK (player, id_tournament)

<i>Categories</i>
id_category - PK
name xp_threshold

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 GitHub, 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*³² 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*²⁹

- **Django:** este directorio alberga el proyecto y la aplicación de *Django Rest Framework*³¹ 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*³¹, de ahí que sean directorios al mismo nivel.
- **Ionic:** este directorio alberga la aplicación de Ionic Framework²⁷, 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.yml*.

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
```

Este *Dockerfile*:

- Define la imagen base utilizada para construir el contenedor. En este caso, se utiliza la imagen oficial de *Python*³⁰ 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*³⁰ 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 *Python*³⁰, se instalan las dependencias necesarias y se copia el código al contenedor.

La issue relacionada en *GitHub* 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
```

Este *Dockerfile*:

- 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*²⁷.
- 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*²⁷ utilizando *Node.js* en la etapa de construcción, y luego utiliza una imagen de *NGINX*³⁶ en la etapa de producción para servir los archivos generados por la compilación de la aplicación de *Ionic*²⁷. 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* relacionada es [Dockerize the frontend \(S1\)](#).

Ambos *Dockerfiles* serán utilizados por el fichero *docker-compose.yaml*

```

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"

```

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.
- **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* 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	▶ ⋮

Respecto a GitHub, 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*³¹:

- **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 DRF³¹ 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*.
- **Serializadores**: son clases que se utilizan para convertir los objetos de Django (como los modelos) 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, 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

```

Se define el serializador para el modelo *Player*. 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

```

Posteriormente, se define la vista 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*³¹ 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)
```

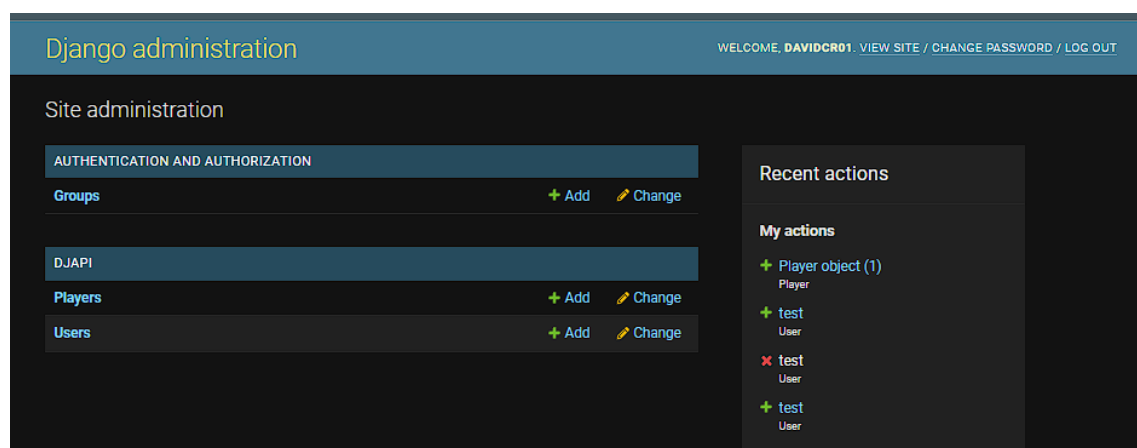
Por último, las vistas son habilitadas mediante rutas, definidas en el archivo *urls.py*. Por ejemplo, en el caso de los jugadores, se añade al router de la aplicación la ruta para los jugadores:

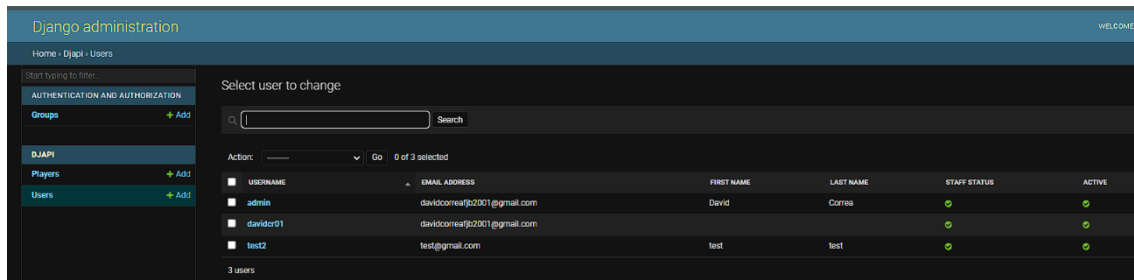
```
router = routers.DefaultRouter()
router.register(r'api/players', views.PlayerViewSet)
```

La issue relacionada con GitHub 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* 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. Esta interesante opción de Django ahorra la necesidad de hacer otra aplicación exclusiva para los administradores.





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* [Customize the Users authentication \(S1\)](#).

Frontend

En cuanto al *frontend*, *Ionic Framework*²⁷ 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 TypeScript³⁷. El entorno proporciona componentes predefinidos y permite crear componentes personalizados.

En este Sprint, la única vista a implementar era el formulario de registro de usuarios. El correspondiente boceto, realizado en la herramienta *Figma*, queda de la siguiente manera:

*Ionic Framework*²⁷ 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* 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>
```

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* 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);
}
}
```

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 GitHub [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 Django³¹. 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. Pruebas y resultado. Revisión.

En cuanto a las pruebas, se comprueba que el formulario contempla los posibles casos de error:

The image displays three sequential screenshots of the Wordle+ registration form, illustrating various error handling scenarios. Each screenshot features the Wordle+ logo at the top and a 'REGISTER' button at the bottom.

- Left Screenshot:** Shows the form with the 'Email' field highlighted in red, indicating it is required. The 'Username' field contains 'david2'. The 'First name' and 'Last name' fields are empty. The 'Password' field is masked with dots. The 'Staff code (optional)' field is empty.
- Middle Screenshot:** Shows the form with the 'Username' field containing 'staffmember'. The 'Email' field contains 'staff@gmail.com'. The 'First name' field contains 'David' and the 'Last name' field contains 'Correa'. The 'Password' field is masked with dots. The 'Staff code (optional)' field contains '819493'. A red error message at the bottom states: 'A user with that username already exists. ×'.
- Right Screenshot:** Shows the form with the 'Username' field containing 'staffmember2'. The 'Email' field contains 'staff@gmail.com'. The 'First name' field contains 'David' and the 'Last name' field contains 'Correa'. The 'Password' field is masked with dots. The 'Staff code (optional)' field contains '819493'. A red error message at the bottom states: 'Invalid staff code. ×'.

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

WORDLE+

Username

Email

First name

Last name

Password

Staff code (optional)

REGISTER

User created successfully ✓

Change user

staffmember

Username: staffmember
Required. 150 characters or fewer. Letters, digits and @/

Password: algorithm: pbkdf2_sha256 iterations: 390000 s
Raw passwords are not stored, so there is no way to see

Personal info

First name: David

Last name: Correa

Email address: staff@gmail.com

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 GE. Inicio y cierre de sesión	27 de febrero del 2023	
Ident.	Título	Estimación	Prioridad
HU.2	Como GE quiero poder registrarme	2	1
HU.3	Como usuario (jugador y gestor) 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 gestores de eventos (administradores) y desarrollar el acceso a la plataforma mediante el inicio de sesión, además del cierre de sesión.

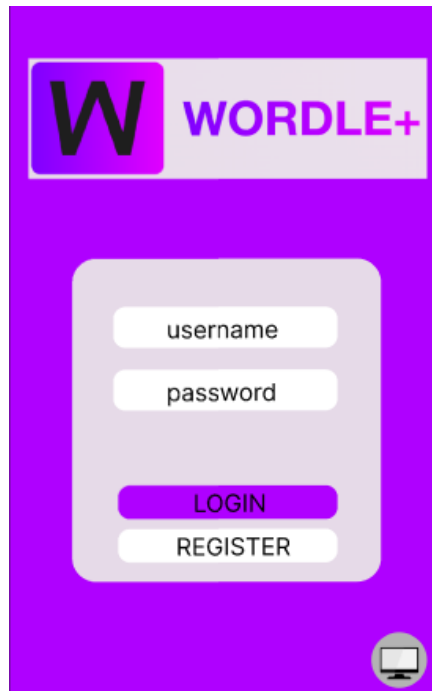
En sintonía con el proyecto de GitHub, este Sprint está estrechamente relacionado con los siguientes *milestones*. Se pueden consultar las *issues* relacionadas de cada *milestone*:

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

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 gestores de eventos 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. El correspondiente boceto, realizado en la herramienta Figma, queda de la siguiente manera:



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*³¹ 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()* 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'
  }
);
}
```

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 DRF³¹, 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.

DRF³¹ 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 DRF³¹ se ejecutan cuando se realiza una petición HTTP, antes de ejecutar la correspondiente vista.

Sin embargo, quizá es 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

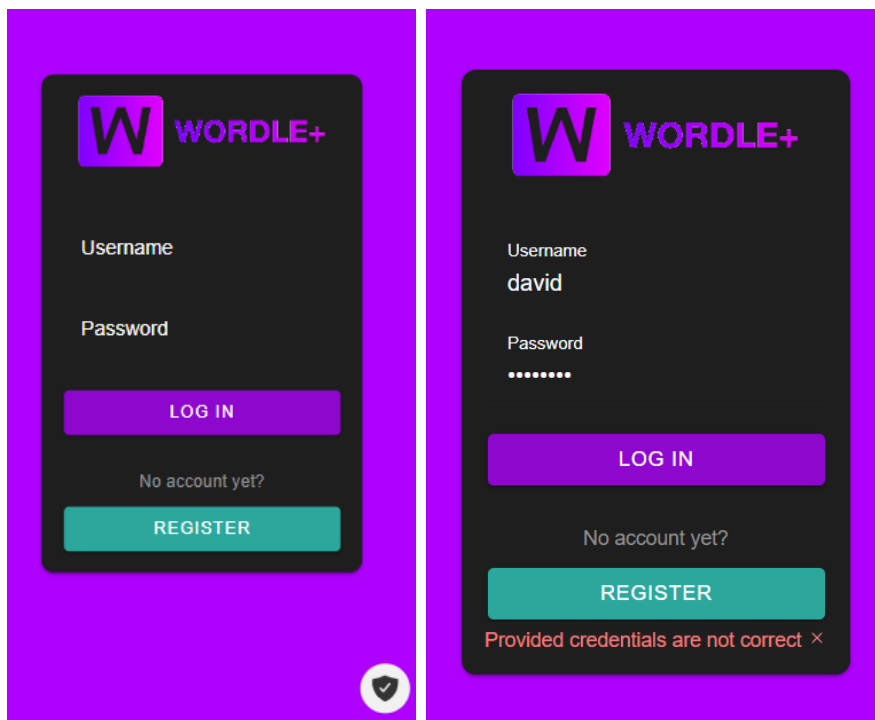
almacenado en *Ionic Storage* y redirige al usuario a la página de inicio de sesión. Más información puede ser consultada en la *issue* [Logout \(S2\)](#)

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. Pruebas y resultado. Revisión.

El resultado de las vistas han sido las siguientes:



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:

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

Cuando se pulsa el botón de *logout* se comprueba que el token es eliminado del almacenamiento correctamente:

Entradas totales: 0		
#	Clave	Valor

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.

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 GitHub, pero sin dejar de profundizar en los detalles e implementación desarrollados. Estas buenas prácticas deben de seguir realizándose en los siguientes Sprints.

CAPÍTULO IV. CONCLUSIONES Y TRABAJOS FUTUROS (TODO).

BIBLIOGRAFÍA Y REFERENCIAS.

-
- 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/>.
- ⁹ 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%94%20document%20database.>
- ¹⁸ Inc., MongoDB. *mongodb.com*. s.f. <https://www.mongodb.com/> (último acceso: 25 de febrero de 2023).
- ¹⁹ Platforms, Meta. *reactjs.org*. s.f. <https://es.reactjs.org/>.
- ²⁰ Angular. *angular.io*. s.f. <https://angular.io/> (último acceso: 25 de febrero de 2023).
- ²¹ Foundation, OpenJS. *nodejs.org*. s.f. <https://nodejs.org/en/> (último acceso: 25 de febrero de 2023).
- ²² Dart. *dart.dev*. s.f. <https://dart.dev/> (último acceso: 25 de febrero de 2023).
- ²³ Native, React. *https://reactnative.dev/*. s.f.
- ²⁴ Apple. *Objective-C*. s.f. <https://developer.apple.com/library/archive/documentation/Cocoa/Conceptual/ProgrammingWithObjectiveC/Introduction/Introduction.html>.
- ²⁵ Inc, Apple. *swift*. s.f. <https://www.apple.com/es/swift/>.
- ²⁶ Oracle. *java.com*. s.f. <https://www.java.com/es/> (último acceso: 25 de febrero de 2023).
- ²⁷ Ionic. *https://ionicframework.com/*. s.f.
- ²⁸ You, Evan. *vuejs.org*. s.f. <https://vuejs.org/> (último acceso: 25 de febrero de 2023).
- ²⁹ PostgreSQL. *postgresql.org*. s.f. <https://www.postgresql.org/> (último acceso: 25 de febrero de 2023).
- ³⁰ Python. *python.org*. s.f. <https://www.python.org/> (último acceso: 25 de febrero de 2023).
- ³¹ Framework, Django Rest. *django-rest-framework.org*. s.f. <https://www.django-rest-framework.org/> (último acceso: 25 de febrero de 2023).
- ³² Docker. *docker.com*. s.f. <https://www.docker.com/> (último acceso: 25 de febrero de 2023).
- ³³ 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).
- ³⁴ 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).
- ³⁵ 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).
- ³⁶ NGINX. *nginx.com*. s.f. <https://www.nginx.com/> (último acceso: 13 de mayo de 2023).
- ³⁷ TypeScript. *typescriptlang.org*. s.f. <https://www.typescriptlang.org/> (último acceso: 14 de mayo de 2023).