ADIVINA LA PALABRA

Icono

Descripción generada automáticamente

Este trabajo se encuentra bajo la licencia de “**Creative Commons**”

[Reconocimiento-NoComercial-CompartirIgual 4.0 Internacional](https://creativecommons.org/licenses/by-nc-sa/4.0/)

[Esta no es una licencia de Cultura Libre](https://creativecommons.org/share-your-work/public-domain/freeworks)

Dibujo con letras blancas

Descripción generada automáticamente con confianza media

**Eres libre de:**

**Compartir**: copiar y redistribuir el material en cualquier medio o formato

**Adaptar** : remezclar, transformar y construir sobre el material.

# AGRADECIMIENTOS

Me gustaría agradecer a todas las personas que han hecho que este proyecto siguiera adelante, ya que sin ellas no hubiera sido posible la realización de este.

En primer lugar, al responsable de formación Javier Marco, y a mi empresa Sopra Steria por cederme el derecho de la utilización de un proyecto realizado en la empresa para la entrega de este como trabajo final de ciclo.

Dar las gracias al equipo de Centro de Servicio Digitales de Sopra Steria por la formación recibida, incluyendo a todos mis compañeros de prácticas que me han ayudado y apoyado durante esta formación y la realización de este proyecto.

También estoy muy agradecido a mis profesores de 2º de DAW Salva, César y Lorenzo, por la fantástica enseñanza recibida de parte de ellos, y los buenos momentos pasados a su lado.

Por último, también quería dar las gracias a todos mis compañeros de ciclo por todo el apoyo recibido durante el curso y el buen ambiente que había en clase gracias a ellos.

ÍNDICE

[AGRADECIMIENTOS 3](#_Toc134895998)

[1. MARCO DE LA INVESTIGACIÓN 5](#_Toc134895999)

[1.1 TEMÁTICA ELEGIDA 6](#_Toc134896000)

[1.2 CONTEXTUALIZACIÓN 7](#_Toc134896001)

[2. ORGANIZACIÓN DEL PROYECTO 8](#_Toc134896002)

[2.1 TEMPORALIZACIÓN 8](#_Toc134896003)

[INVESTIGACIÓN 8](#_Toc134896004)

[PLANIFICACIÓN 10](#_Toc134896005)

[DESARROLLO 11](#_Toc134896006)

[2.2 RECURSOS 15](#_Toc134896007)

[FRONTEND 15](#_Toc134896008)

[BACKEND 18](#_Toc134896009)

[METODOLOGÍAS 21](#_Toc134896010)

[HERRAMIENTAS 22](#_Toc134896011)

[3. APLICACIÓN PRÁCTICA](#_Toc134896012)

[3.1 INTRODUCCIÓN](#_Toc134896013)

[3.2 CICLO DE VIDA](#_Toc134896014)

[3.2.1 ANÁLISIS DE REQUERIMIENTOS](#_Toc134896015)

[3.2.2 DISEÑO](#_Toc134896016)

[3.2.3 IMPLEMENTACIÓN](#_Toc134896017)

[3.2.4 PRUEBAS](#_Toc134896018)

[3.3 PROPUESTAS DE MEJORA](#_Toc134896019)

[4. MANUAL DE USO](#_Toc134896020)

[5. VALORACIÓN PERSONAL](#_Toc134896021)

[6. FUENTES BIBLIOGRÁFICAS](#_Toc134896022)

# 1. MARCO DE LA INVESTIGACIÓN

El **objetivo** de este proyecto es aprender cómo se trabaja en un **proyecto real** utilizando las metodologías ágiles Scrum y Kanban, además de aprender y utilizar la práctica de programación TDD.

Imagen que contiene Logotipo

Descripción generada automáticamenteA parte de aprender a trabajar con las **metodologías** mencionadas la idea de este proyecto es también poner en práctica los conocimientos recibidos durante el ciclo formativo, y las 6 semanas de formación recibidas en la empresa. Para esto se utilizarán principalmente los *frameworks* Spring y Angular.



## 1.1 TEMÁTICA ELEGIDA

Las **razones** por las cuales he elegido realizar el proyecto utilizando Java y Spring para la parte servidor, y Angular y TypeScript para la parte cliente, son porque en el ciclo formativo, me formé en la realización del *frontend* de aplicaciones web en Angular, y quería indagar más sobre esta tecnología, ya que me parece un *framework* de *frontend* de alta calidad, porque, para aplicaciones grandes, es muy robusto, y el *scaffolding* de la aplicación queda muy ordenado y estructurado.

Los **motivos** por los cuales he realizado la parte servidor con Java y Spring, son los siguientes: en la formación interna de la empresa donde he realizado las prácticas aprendí Java 8, microservicios, TDD, y Spring, y después de la formación quería afianzar los conceptos aprendidos e investigar un poco más acerca de Spring, para dominarlo, y aplicar todo lo aprendido en la empresa a un proyecto real.

Además de las razones dadas, al ser un proyecto realizado durante el periodo de prácticas en la empresa, se nos sugirió la utilización de las tecnologías utilizadas durante dicho periodo.

En relación con la **aplicación** que se va a desarrollar para poner en práctica la investigación de las tecnologías mencionadas, la empresa eligió un proyecto sencillo pero perfecto para practicar todo lo aprendido, **Proyecto Wordle: Adivina la Palabra.**

**Wordle** es una aplicación web donde el usuario puede jugar a adivinar una palabra de 5 letras. En cada partida, el usuario tendrá que intentar adivinar una palabra mediante unas pistas dadas en cada intento: las letras que estén en el lugar correcto, aparecerán de color verde, las letras que no estén incluidas en la palabra, de color gris, y, por último, las letras que estén en la palabra, pero no se encuentren en su ubicación correcta, aparecerán de color naranja. Para ganar la partida, el usuario deberá acertar la palabra en un máximo de 5 intentos. Además, Wordle incluye un sistema de *login* y un historial de partidas, donde el jugador podrá consultar sus partidas anteriores.

## 1.2 CONTEXTUALIZACIÓN

En cuanto a la idea de este proyecto, la **idea** **inicial** era realizar una aplicación web desde cero con Angular, Laravel y GraphSQL, pero después de las tutorías con mi tutor de proyecto, me aconsejó que eligiera una aplicación que tuviera ya hecha y la documentarla bien.

Después de esto, solicité a la empresa la posibilidad de utilizar el proyecto que estábamos desarrollando en ese momento, con la finalidad de realizar la documentación de este, y poder aprovechar el desarrollo de una **aplicación web real, en un entorno real.**

Volviendo a la idea de Wordle, conozco la utilización de las tecnologías por parte de la empresa, ya que, en la formación interna, el equipo de *frontend* llevaba 6 semanas investigando, aprendiendo y practicando el *framework* Angular, mientras que el equipo de *backend* estaba trabajando con Spring.

Además, la aplicación Wordle, se basa en una aplicación real que consiste en [adivinar la palabra del día](https://lapalabradeldia.com/), lo cual les pareció una aplicación interesante para practicar todo lo aprendido durante la formación ya que Wordle es una aplicación sencilla, pero tiene lo necesario para poder simular un proyecto real.

De cara a futuros proyectos, gracias a la investigación de Angular y Spring, se abre un mundo de posibilidades para la implementación de cualquier aplicación web ya que son dos tecnologías muy versátiles y se pueden utilizar para muchos proyectos.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamenteEn relación con el proyecto realizado, en un futuro se podría continuar implementando y añadiendo mejoras, ya que, al partir de una aplicación bastante sencilla, a la hora de mejorarla y añadir funcionalidades ofrece muchas posibilidades de mejora.

# 2. ORGANIZACIÓN DEL PROYECTO

## 2.1 TEMPORALIZACIÓN

Para la organización del proyecto he dividido el progreso en tres fases**, Investigación, Planificación y Desarrollo**:

### INVESTIGACIÓN

La fase de **investigación** del proyecto consiste en aprender todos los conceptos que serán necesarios para poder desarrollar correctamente la aplicación. Esta fase en principio consistiría en 2 partes; la investigación de Angular para el *frontend* y la investigación de Spring para el *backend*, pero finalmente se decidió solo investigar Spring, ya que todos los conocimientos de Angular necesarios se habían aprendido previamente en el curso del ciclo formativo.

|  |  |  |
| --- | --- | --- |
| Actividad | Observaciones | Horas |
| Lunes 20 de marzo | | |
| Repaso Java | Repaso general, funciones, tipos, POO. | 8 |
| Martes 21 de marzo | | |
| Repaso Java | Repaso general colecciones, fechas, enum. | 8 |
| Miércoles 22 de marzo | | |
| Repaso Java | Repaso general JDBC, acceso a BD | 8 |
| Jueves 23 de marzo | | |
| Java 8 | Introducción a Java 8, lambdas, interfaces funcionales y *streams*, video explicativo e investigación. | 8 |
| Viernes 24 de marzo | | |
| Java 8 | Introducción a Java 8, lambdas, interfaces funcionales y *streams*, ejercicios prácticos. | 7 |
| Lunes 3 de abril | | |
| Java 8 | Dates, Maven y microservicios, video explicativo e investigación. | 8 |
| Martes 4 de abril | | |
| Java 8 | Dates, Maven y microservicios, ejercicios prácticos. | 8 |
| Miércoles 5 de abril | | |
| Spring | Introducción a Spring, funcionamiento, estructura y decoradores, video explicativo e investigación. | 8 |
| Jueves 6 de abril | | |
| Spring | Introducción a Spring, funcionamiento, estructura y decoradores, ejercicios prácticos. | 8 |
| Viernes 7 de abril | | |
| Spring | API REST y JPA, video explicativo e investigación. | 7 |
| Lunes 10 de abril | | |
| Spring | API REST y JPA, ejercicios prácticos. | 8 |
| Martes 11 de abril | | |
| TDD | Introducción a TDD, video explicativo e investigación | 8 |
| Miércoles 12 de abril | | |
| TDD | TDD, ejercicios prácticos, Kata FizzBuzz | 8 |
| Jueves 13 de abril | | |
| TDD | TDD, ejercicios prácticos, Kata Potter | 8 |
| Viernes 14 de abril | | |
| GIT, Agile | Formación en GIT, comandos, GIT Flow y buenas prácticas.  Formación en metodologías ágiles, clase teórica y ejercicios prácticos, Scrum y Kanban | 8 |

### PLANIFICACIÓN

La fase de **planificación** del proyecto consiste en planificar el desarrollo del proyecto, captura de requisitos, análisis de estos, creación de las historias de usuario, y la planificación de cada Sprint Backlog.

|  |  |  |
| --- | --- | --- |
| Actividad | Observaciones | Horas |
| Lunes 17 de abril | | |
| Requisitos | El Product Owner se reúne con el cliente para hacer la captura de requisitos | - |
| Martes 18 de abril | | |
| Historias de usuario | El Product Owner presenta los requisitos y se deciden las historias de usuario que tendrá el proyecto | - |
| Miércoles 19 de abril | | |
| Sprint planning 1 | Ceremonia del primer sprint donde nos reunimos todo el equipo de desarrollo con el Scrum Master y estimamos cada HU en puntos de esfuerzo, redactamos las tareas de cada HU y acordamos cuantos puntos de esfuerzo entrarán en el primer sprint | 2 |
| Martes 2 de mayo | | |
| Sprint planning 2 | Ceremonia del segundo sprint donde nos reunimos todo el equipo de desarrollo con el Scrum Master y estimamos cada HU en puntos de esfuerzo, redactamos las tareas de cada HU y acordamos cuantos puntos de esfuerzo entrarán en el segundo sprint | 3 |

### DESARROLLO

La fase de **desarrollo** del proyecto es donde se realiza la implementación del proyecto, se divide en sprints de 2 semanas aproximadamente, cada día todo el equipo de desarrollo se reúne en una ceremonia de scrum que se llama daily donde todos ponen en común que han estado haciendo durante la jornada anterior, que van a hacer durante esta jornada y si tienen algún bloqueo, al final de cada sprint se celebra la ceremonia sprint review donde se presenta el trabajo de todo el sprint a los steak holders y la ceremonia de sprint retrospective donde se reúne todo el equipo de desarrollo para poner en común que ha ido bien y que se puede mejorar de cara al siguiente sprint.

|  |  |  |
| --- | --- | --- |
| Actividad | Observaciones | Horas |
| Miércoles 19 de abril | | |
| HU1 – Crear seed | Crear la base del proyecto, repositorio de git y configurar proyecto Java. | 8 |
| Jueves 20 de abril | | |
| HU2 – Verificar existencia palabra | Crear BD para almacenar el diccionario de palabras y crear endpoint (/checkIfWordExists) que verifique si una palabra existe en el diccionario, controlar posibles errores, y realización de los tests. | 8 |
| Viernes 21 de abril | | |
| HU3 – Nueva partida | Crear la entidad “Game”, y crear un endpoint (/newGame) para almacenar una nueva partida en la BD y que devuelva el id de la partida creada, gestión de errores y realización de los tests | 7 |
| Lunes 24 de abril | | |
| HU4 – Validación de la partida | Crear endpoint (/validatePositions) que reciba un id de partida y una palabra, y que recupere la palabra correcta de esa partida, gestión de errores en la petición y realización de los tests del controller | 8 |
| Martes 25 de abril | | |
| HU4 – Validación de la partida | Validar la palabra pasada con la palabra correcta de la partida, gestión de errores si la palabra pasada no existe y realización de tests del servicio | 8 |
| Miércoles 26 de abril | | |
| HU4 – Validación de la partida | Validar cada letra de cada posición de la palabra pasada con la palabra correcta y devolver un JSON con cada letra validada, implementación de los tests de los nuevos métodos del servicio | 8 |
| Jueves 27 de abril | | |
| HU5 – Fin de la partida | Creación del endpoint getCorrectWord/{game\_id}, e implementación del servicio que compruebe si la palabra esta acertada y mensaje de partida ganada, realización de los tests del servicio implementado | 8 |
| Viernes 28 de abril | | |
| Sprint 1 – Review y Retrospective | Sprint review del primer sprint, reunión con los steak holders y revisión de issues para el próximo sprint, se indica que se especifique mejor el mensaje de error en cada casuística y cambio de color principal de la aplicación.  Sprint retrospective del primer sprint, reunión de todo el equipo de desarrollo donde se pone en común que hemos trabajado muy bien con mucha comunicación, y los aspectos a mejorar son la organización del Taiga y del GIT, y el uso de TDD en todo momento. | 7 |
| Martes 2 de mayo | | |
| HU6 – Limitar la partida a 5 intentos | Mejora del endpoint /validatePositions para limitar la partida a solo 5 intentos | 8 |
| Miércoles 3 de mayo | | |
| HU6 – Limitar la partida a 5 intentos | Test unitarios de la nueva mejora del servicio para limitar la partida a 5 intentos | 8 |
| Jueves 4 de mayo | | |
| HU7 - Login | Configuración de spring security para que la autenticación sea necesaria en todos los endpoints de la API exceptuando los endpoints de login y register | 8 |
| Viernes 5 de mayo | | |
| HU7 - Login | Creación de las clases necesarias para gestionar los tokens con JWT, JWTEntryPoint, para controlar el acceso de la API y enviar un 401 si no está autorizado, JWTUtils para generar el token, validarlo y extraer información de este, y JWTFilter para realizar los filtros de autenticación necesarios | 8 |
| Lunes 8 de mayo | | |
| HU7 - Login | Creación de la entidad User, su servicio, su controlador y su repositorio para autenticar los datos de usuario en las peticiones a /login y enviar el token si son correctos | 8 |
| Martes 9 de mayo | | |
| HU7 - Login | Realización de test unitarios de todo el proceso de autenticación implementado | 8 |
| Miércoles 10 de mayo | | |
| HU9 – Histórico 10 últimas partidas | Creación del endpoint /getLastTenGames, e implementación del servicio para devolver las ultimas 10 partidas del usuario que las solicita | 8 |
| Jueves 11 de mayo | | |
| HU9 – Histórico 10 últimas partidas | Editar la entidad Game para marcarla como ganada o perdida después de los 5 intentos, y editar todos los servicios que la utilizaban | 8 |
| Viernes 12 de mayo | | |
| HU9 – Histórico 10 últimas partidas | Testeo de todo el proceso del nuevo endpoint creado /getLastTenGames | 7 |
| Lunes 15 de mayo | | |
| HU10 – Ver todas las partidas | Creación del endpoint /getAllGames que devuelve todas las partidas de un usuario, gestión de errores, si el usuario no tiene más de 10 partidas, mensaje de error. | 8 |
| Martes 16 de mayo | | |
| TEST | Testeo de toda la aplicación antes de la entrega y mejora y refactorización del código. | 8 |
| Miércoles 17 de mayo | | |
| Sprint 2 – Review y Retrospective | Sprint review del segundo sprint, reunión con los steak holders y revisión de issues para el próximo sprint, se indican las mejoras que debemos implementar de cara al siguiente sprint.  Sprint retrospective del segundo sprint, reunión de todo el equipo de desarrollo donde se pone en común que hemos trabajado muy bien con mucha comunicación, y tenemos muy buen rollo en el equipo, hemos mejorado en la organización del Taiga y del GIT, pero aún nos queda mejorar un poco en el uso del TDD en todo el proceso de programación, no debemos hacer código sin hacer el test antes. | 8 |

## 2.2 RECURSOS

En este apartado de la memoria indicaré todos los recursos, tecnologías y metodologías empleadas para el desarrollo del proyecto. Dividiré este apartado en cuatro secciones: Frontend, Backend, Metodologías y Herramientas.

### FRONTEND

HTML

HTML (HyperText Markup Language) es un lenguaje de marcas utilizado para definir la estructura básica de la Web, mediante etiquetas que definen texto, imágenes y otro contenido para mostrarlo en un navegador Web.

\* Versión utilizada 🡪 HTML5

CSS

Icono

Descripción generada automáticamente

CSS (Cascading Style Sheet) es un lenguaje de estilos que sirve para describir como deben ser renderizadas las etiquetas HTML de una página web. Da estilo y mejora la apariencia de un sitio web.

\* Versión utilizada 🡪 CSS3

Logotipo

Descripción generada automáticamenteSASS

SASS es un procesador CSS, SASS nos permite dotar de funcionalidad al lenguaje de estilos CSS, permitiéndonos así utilizar funciones, bucles, condicionales, variables… Haciendo así que el CSS sea más mantenible y reutilizable.

\* Versión utilizada 🡪1.62.1

BOOTSTRAP

Logotipo

Descripción generada automáticamenteBootstrap es un framework de CSS que tiene plantillas de diseño, y clases ya hechas que nos permite dar estilo a una página web de una forma sencilla y rápida. Utiliza SASS para desarrollar sus propias clases y permite ampliar y personalizar los estilos.

\* Versión utilizada 🡪 5.3

Imagen que contiene dibujo

Descripción generada automáticamenteTYPESCRIPT

TypeScript es un lenguaje de programación fuertemente tipado basado en JavaScript. TypeScript nos permite dar dinamismo a la página web, pudiendo así dotar de lógica y funcionalidad a la parte del navegador del sitio web.

\* Versión utilizada 🡪5.0.4

Logotipo, nombre de la empresa

Descripción generada automáticamenteANGULAR

Angular es un framework de frontend desarrollado en TypeScript y de código abierto. Angular como todos los frameworks, nos facilita el trabajo, en este caso para desarrollar la parte del cliente de una manera más rápida, estructurada. Angular sirve para desarrollar SPA (Single Page Application) y nos permite desarrollar aplicaciones web robustas, escalables y optimizables.

\* Versión utilizada 🡪 16.0.0

Un conjunto de letras blancas en un fondo blanco

Descripción generada automáticamente con confianza bajaANGULAR MATERIAL

Angular material es una librería de estilos, basada en la guía de estilos Material Design. Esta desarrollada por el equipo de Angular y preparada para integrarse perfectamente con Angular. Esta librería ofrece plantillas de estilo ya hechas para facilitar el diseño de una página web.

\* Versión utilizada 🡪 16.0.0

JASMINE

Icono

Descripción generada automáticamente

Jasmine es una suite de testing que sigue la metodología Behavior Driven Development. Tiene cosas muy buenas como que no requiere un DOM para hacer los tests. Además, Jasmine, tiene una sintaxis muy clara, ofrece funciones que permiten una escritura más fácil, y cada test creado por el usuario es una función.

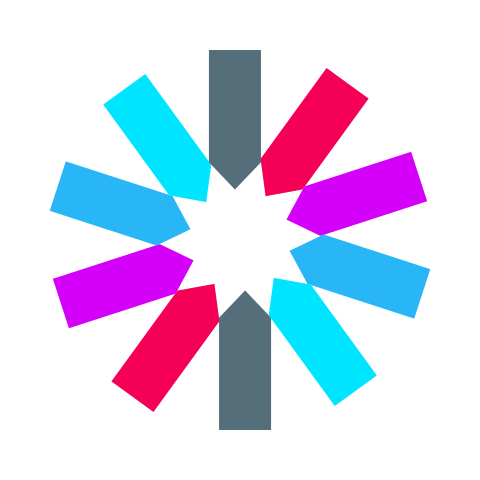
\* Versión utilizada 🡪 4.6.0

Icono

Descripción generada automáticamenteKARMA

Karma es el test-runner, es decir, el módulo que permite automatizar algunas de las tareas de las suites de testing, como Jasmine. Karma, además, ha sido desarrollado directamente por el equipo de Angular. Karma se encarga de ejecutar los test de Javascript según se vayan construyendo, de tal forma que ante cualquier fallo el desarrollador se dará cuenta de inmediato.

\* Versión utilizada 🡪 6.4.2

 JWT

JSON Web Token (JWT) es un token que sirve para trasmitir información de forma segura entre cliente y servidor de forma compacta y autónoma a través de un objeto JSON. Esta información se puede verificar y es segura porque está firmada digitalmente. Los JWT se pueden firmar usando una clave secreta (con el algoritmo HMAC) o un par de claves pública/privada usando RSA o ECDSA. JWT se usa tanto en cliente como en servidor ya que es la forma de transmisión que usamos para la autorización y autenticación de usuarios.

### BACKEND

JAVA

Logotipo

Descripción generada automáticamente con confianza mediaJava es un lenguaje de programación y una plataforma informática lanzada en 1995 por Sun Microsystems. Desde entonces ha evolucionado mucho y cuenta con muchas versiones, disponibles. Java es un lenguaje de programación orientada a objetos, fuertemente tipado y compilado e interpretado.

Lo que diferencia a Java de otros lenguajes de programación es que se puede ejecutar en diferentes sistemas operativos ya que para ejecutarse se compila y se genera un código bytecode el cual son instrucciones en código máquina que son interpretados por la máquina virtual de Java JVM.

\* Versión utilizada 🡪 SE 17

Texto, Logotipo, nombre de la empresa

Descripción generada automáticamenteMAVEN

Maven es una herramienta open-source que facilita el proceso de compilar y generar ejecutables a partir del código fuente, sin importar las librerías o dependencias del código, trabajo que antes era mucho más pesado ya que había que analizar que dependencias y librerías se utilizaban y donde incluirlas para poder compilar una aplicación. Aparte de facilitar el proceso de compilación Maven también sirve para la gestión de dependencias e incluso como herramienta de documentación.

\* Versión utilizada 🡪 3.9.1

SPRING

Logotipo

Descripción generada automáticamenteSpring es un framework para el desarrollo de aplicaciones web en Java. La finalidad de Spring es s estandarizar, agilizar, manejar y resolver los problemas que puedan ir surgiendo en el proceso de programación. Spring ayuda en el desarrollo ahorrando la escritura de código, como el enrutamiento, las consultas SQL… Permitiéndonos así centrarnos solo en la lógica de negocio de la aplicación y no “perder” tanto tiempo escribiendo código que se escribe siempre en todas las aplicaciones que se desarrollan. Aparte de estas facilidades Spring también ofrece la posibilidad de configurar la aplicación en muchos aspectos, como por ejemplo la seguridad de la aplicación a través de Spring Security.

\* Versión utilizada 🡪 3.0.5

LOMBOK

Un dibujo de un personaje animado

Descripción generada automáticamente con confianza bajaLombok es una librería de Java que nos permite eliminar código que se repite en todas las clases, y que Java nos obliga a incorporar si queremos utilizar estas funcionalidades, como, por ejemplo, los getters y setters de atributos, el toString. Todo esto Lombok lo soluciona con anotaciones como @Getter, @Setters…

\* Versión utilizada 🡪 1.18.26

\* En el proceso de desarrollo descubrimos una solución alternativa a Lombok que incorpora Java propiamente y hemos utilizado en algunas clases. La solución alternativa son los records, disponibles oficialmente a partir de la versión JDK 16. (adjuntamos enlace para más información)

JUNIT

Icono

Descripción generada automáticamenteJunit es un framework de Java que sirve para realizar tests automatizados en Java (unitarios y de integración), y controlar y evaluar si el funcionamiento de cada clase, y sus métodos funciona de forma correcta. Junit nos ofrece clases y métodos para facilitarnos la tarea de testo.

\* Versión utilizada 🡪 5.7. 1

MOCKITO

Imagen que contiene dibujo, señal

Descripción generada automáticamenteMockito es un framework que sirve para hacer tests. Mockito facilita la realización de pruebas ya que permite la creación de objetos simulados, esto sirve para simular la instancia de un objeto cuan aún no se ha creado, como es el caso de los métodos de testeos los cuales necesitan de la utilización de objetos que no han sido instanciados. Además, Mockito nos proporciona anotaciones para ahorrarnos la escritura de código y eliminar el código repetitivo en nuestra aplicación.

\* Versión utilizada 🡪 5.3.1

MYSQL

Logotipo

Descripción generada automáticamenteMySQL es un sistema de gestión de base de datos relacional. Es un software de código abierto desarrollado por Oracle. Se considera como la base de datos de código abierto más utilizada en el mundo. MySQL sirve para almacenar y administrar datos con las acciones CRUD. Otra característica importante de MySQL es que utiliza el modelo cliente-servidor.

\* Versión utilizada 🡪 8.0

### METODOLOGÍAS

SCRUM

Imagen que contiene alimentos, vidrio

Descripción generada automáticamenteScrum es una metodología de desarrollo ágil. La filosofía de scrum es la colaboración, de ahí la traducción de su nombre “melé” que viene del rugby donde los jugadores se apoyan unos a otros. El objetivo de scrum es desarrollar proyectos de manera más rápida y eficiente. Como en scrum la colaboración es muy importante, al desarrollar un proyecto siguiendo esta metodología la comunicación y el apoyo del equipo es fundamental.

Scrum consiste en abordar cualquier proyecto dividiéndolo en sprints o partes más pequeñas. Dentro de este entorno de trabajo hay que seguir una serie de fases para abordar cada tarea, y participan unos roles específicos que garantizan el cumplimiento de esta filosofía de trabajo.

Ejemplo de cómo hemos trabajado en Scrum en este proyecto. (enlace)

KANBAN

Logotipo, Icono

Descripción generada automáticamenteKanban (letrero o tarjeta en japonés) es un sistema de gestión de flujo de trabajo, sirve para ver de forma visual el proceso de desarrollo, normalmente se divide en un tablero con 3 tarjetas To do (tareas para realizare) Doing (tareas en progreso) y Done (tareas realizadas)

Ejemplo de cómo hemos trabajado en Kanban en este proyecto. (enlace)

TDD

Diagrama

Descripción generada automáticamenteTDD (Test Driven Development) es una práctica de programación en la que se empieza a programar por el test. Se basa en la filosofía de baby-steps lo que significa “ir pasito a pasito”, primero se hace el test, después se implementa el método lo más sencillo posible para que pase el test, y después se va refactorizando, probando el test después de cada step.

### HERRAMIENTAS

VISUAL STUDIO CODE

Icono

Descripción generada automáticamenteVisual Studio Code es un editor de código, desarrollado por Microsoft. Incluye soporte para la depuración, control integrado de Git, resaltado de sintaxis, finalización inteligente de código, fragmentos y refactorización de código. Además, adicionalmente puedes añadir extensiones para facilitar el proceso de desarrollo.

HEIDISQL

Icono

Descripción generada automáticamenteHeidiSQL es una herramienta open-source que sirve para conectarse a MySQL. HeidiSQL permite crear base de datos, tablas, campos, insertar datos, hacer consultas, y todas las acciones posibles para gestionar una base de datos SQL.

Icono

Descripción generada automáticamenteGIT

GIT es un sistema de control de versiones distribuido, lo que significa que un clon local del proyecto es un repositorio de control de versiones completo. Git está pensando en la eficiencia, la confiabilidad y compatibilidad del mantenimiento de versiones de aplicaciones cuando estas tienen un gran número de archivos de código fuente.

Icono

Descripción generada automáticamenteGITHUB

GitHub es un servicio en la nube que aloja GIT. GitHub permite colaborar en un proyecto GIT y realizar cambios en un mismo proyecto, guardando un histórico de todos los cambios realizados. GitHub también ofrece la posibilidad de tener varios repositorios con distintos proyectos alojados en Internet.

GOOGLE CHROME

Logotipo

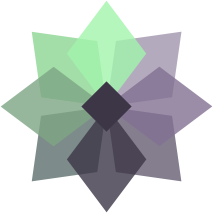
Descripción generada automáticamenteGoogle Chrome es un navegador web de código cerrado​​ desarrollado por Google, aunque derivado de proyectos de código abierto. En este proyecto Chrome ha sido el navegador utilizado para hacer las pruebas de caja blanca o de funcionalidad. También gracias a las herramientas de desarrollo de Chrome, se ha utilizado para depurar código en la parte cliente a través del inspector, y la consola que incorpora el mismo.

TEAMS

Icono

Descripción generada automáticamenteTeams es una aplicación desarrollada por Microsoft. Teams sirve para colaborar dentro de un equipo, estar informados y conectados, permite compartir archivos, hablar por chat y realizar reuniones. En este proyecto teams se ha usado para hacer reuniones con el equipo, realizar ceremonias de Scrum como las dailys o las retrospective.

TAIGA

Taiga es una herramienta desarrollada por Sopra Steria que sirve para gestionar el proceso de desarrollo, mediante un tablero Kanban. Las tarjetas de los tableros de cada sprint son, User Story, New, In progress, Merge Request, Ready For Test, Closed y Needs info. Además, taiga incorpora una sección de backlog donde se pueden ver todas las historias de usuario que hay y estimar cada una de estas. También tiene un apartado de issues donde incorpora todas las incidencias.

KLAXOON

Icono

Descripción generada automáticamenteKlaxoon es una herramienta la cual te permite celebrar las ceremonias de Scrum de una forma más visual. En este proyecto solo hemos utilizado Klaxoon para las Sprint Retrospective, en las que mediante postits apuntábamos los aspectos que han ido bien durante el sprint y los que se pueden mejorar, pero Kalxoon incorpora muchas más funcionalidades para trabajar en equipo de una forma más visual.

# 3. APLICACIÓN PRÁCTICA

## 3.1 INTRODUCCIÓN

Se pide realizar una aplicación para poder jugar al juego de adivina la palabra. Este juego consistirá en que habrá una palabra correcta elegida al azar entre 10.000 palabras de un diccionario de palabras de 5 letras y en español.

El usuario tendrá 5 intentos para acertar la palabra, si la acierta con 5 intentos o menos ganará la partida y quedará registrada, si después de los 5 intentos no ha conseguido acertar la palabra correcta, se desvelará esta y la partida quedará guardada como perdida en su histórico de partidas.

Durante el juego se le irán dando ciertas pistas al jugador para que pueda acertar la palabra, las letras de las palabras introducidas irán cambiando de color según si la letra está en la posición correcta (verde), si la letra está en la palabra, pero no en la posición correcta (amarillo) y si la letra no está en la palabra (gris).

Todo lo explicado anteriormente no será posible si el usuario no ha iniciado sesión ya que si no se identifica no podrá acceder a la aplicación.

\*\*\*Captura de una partida a medias\*\*\*

En este apartado, aplicación práctica, explicaré todo el proceso de desarrollo desde el análisis de requisitos hasta las pruebas de la aplicación ya terminada.

\* *El Proyecto Wordle – Adivina la palabra, al ser un proyecto desarrollado en la empresa de prácticas, ha sido desarrollado utilizando las metodologías utilizadas en la empresa que iré explicando a lo largo de la descripción de esta aplicación.*

## 3.2 CICLO DE VIDA

### 3.2.1 ANÁLISIS DE REQUERIMIENTOS

El apartado de análisis de requerimientos consiste en recoger todos los requisitos del cliente y analizar que deberemos hacer para cumplir todos estos.

REQUISITOS **INICIALES**

Primero se reúne el Product Owner con el cliente y hace la captura de requisitos iniciales:

* Se pide una aplicación para que un usuario pueda jugar una partida al juego de adivina la palabra. Para esto se concreta que deberá ser una palabra de 5 letras, que este incluida en el diccionario de palabras proporcionado. El usuario tendrá 5 intentos para acertar la palabra, una vez ganada o perdida la partida se guardará en el historial de partidas del usuario, donde podrá consultar todas sus partidas jugadas.

HISTORIAS DE USUARIO

Después de hacer la captura de requisitos iniciales **se describen y se detallan las historias de usuario** que se deberán desarrollar.

Se concreta que estas historias de usuario tendrán el siguiente formato:

**#ID Titulo – Prioridad** (*importancia de la HU según el cliente*)

**Detalle** 🡪 Resumen rápido de la HU con el formato:

* **Como** 🡪 **Actor** del sistema
* **Quiero** 🡪 Que **acción** se va a realizar
* **Para** 🡪 **Para qué** quiero realizar la acción

**Descripción** 🡪 Se **describen los requisitos** de la HU que han sido consultados con el cliente anteriormente.

**Criterio de aceptación** 🡪 Casos de uso de la HU, **como deben suceder** las acciones que se van a implementar. Se describirán con el formato:

* **Dado** 🡪 Actor
* **Cuando** 🡪 Acción que realiza el actor
* **Entonces** 🡪 Consecuencia de la acción

#1 EN\*-crear seed - MUST

**Must**

Inicializar proyecto.

* Base disponible para que el equipo pueda empezar a trabajar en paralelo en él.

#2 Verificar existencia palabra - MUST

**Como** Usuario

**Quiero** saber si estoy escribiendo una palabra que exista en un teclado virtual

**Para** jugar a "Adivina la palabra"

**Descripción:**

En esta historia de Usuario se necesita desarrollar los elementos mínimos que permitan al usuario validar si está introduciendo una palabra de 5 letras que exista. Todo debajo del título "Adivina la palabra"

Para ello se necesitará:

* Un teclado virtual donde el usuario pueda introducir las letras, este estará en formato español.
* Un espacio con borde donde el usuario pueda ver la palabra escrita antes de enviarla.

El botón “enter“ del teclado no permitirá enviar una palabra si esta no tiene una longitud exacta de 5 caracteres.

Una vez enviada la palabra, se comprobará si es una palabra existente y ocurrirá lo siguiente:

* Si la palabra existe se mostrará el borde del espacio en verde.
* Si la palabra no existe se mostrará el borde del espacio en rojo y un mensaje debajo con el siguiente texto: "La palabra enviada no existe, inténtelo de nuevo"
* Si hay un error en el proceso, se mostrara el siguiente mensaje en pantalla: "UPS, algo ha ido mal, no podremos saber lo looser que eres..."

**Criterios aceptación**

1. **Dado** que soy usuario, **cuando** accedo al juego, **entonces** veo el titulo Bienvenido a Adivina la palabra, un espacio vacío y el teclado virtual.
2. **Dado** que soy usuario, **cuando** pulso letras en el teclado virtual, **entonces** se muestran en el espacio en el mismo orden en que las pulse.
3. **Dado** que soy usuario, **cuando** he introducido menos de 5 letras, **entonces** el botón enviar no se habilita.
4. **Dado** que soy usuario, **cuando** he introducido 5 letras, **entonces** el botón enviar se habilita.
5. **Dado** que soy usuario, **cuando** he enviado una palabra que existe, **entonces** el juego me muestra el espacio con en verde.
6. **Dado** que soy usuario, **cuando** he enviado una palabra que NO existe, **entonces** el juego me muestra el espacio con en rojo y el mensaje que aparece en la descripción.
7. **Dado** que soy usuario, **cuando** hay un error en el proceso, **entonces** la herramienta me avisa con el mensaje de la descripción.

#3 Nueva partida - MUST

**Como** usuario

**Quiero** jugar una nueva partida

**Para** para divertirme un rato

**Descripción:**

* En esta HU se necesita que al acceder un usuario se genere una nueva partida.
* Para generar una nueva partida se debe crear un ID de partida al acceder el usuario y vincularse con una palabra que exista en la base de datos como palabra de la partida.
* Si falla la creación de la partida, deberá mostrar al usuario un modal con el mensaje "Ha habido un fallo al generar la partida, ya se ve lo looser que eres, recarga anda" y bloquearle la pantalla.

**Criterios aceptación**

1. **Dado** que soy usuario, **cuando** accedo al juego, **entonces** se genera una nueva partida como aparece en la descripción y puedo jugar.
2. **Dado** que he accedido al juego, **cuando** falla la generación de nueva partida, **entonces** veo un modal con el mensaje que aparece en la descripción y se bloquea la pantalla.

#4 Validación palabra - MUST

**Como** usuario

**Quiero** jugar a Adivina la palabra

**Para** para ver mi habilidad

**NOTA**

Esta HU es una evolución de las HUs #2 y #3, por lo tanto, deben estar previamente desarrolladas.

**Descripción**:

En esta HU se pretende evolucionar el juego separando las letras de la palabra en diferentes casillas para intentar acertarla en diferentes intentos.

En primero lugar debe haber 5 casillas donde se colocará cada letra. Si pulso una casilla se convierte en la casilla seleccionada.

Si tengo una casilla seleccionada, al pulsar una letra en el teclado se escribirá en dicha casilla, estuviese vacía o no. Después de escribir una letra, la casilla seleccionada será la primera disponible.

En segundo lugar, cada letra tendrá un estado en función de:

* Si alguna de las letras escrita se encuentra en la palabra asignada en la partida se mostrará la casilla en amarillo y también en el teclado virtual.
* Si alguna letra se encuentra asignada en la partida y además coincide en posición en la casilla se mostrará en verde y en el teclado también.
* Si alguna letra no se encuentra en la palabra asignada en la partida se mostrará en gris oscuro y en el teclado también.

Para que el usuario pueda rectificar las letras que pulso a continuación se describe el funcionamiento del botón borrar del teclado:

* Si no hay ninguna casilla seleccionada se borrará el contenido de la última casilla y ésta será la seleccionada.
* Si hay una casilla seleccionada y ésta tiene una letra, se borrará y seguirá siendo la casilla seleccionada.
* Si la casilla seleccionada no tiene contenido y no es la primera letra, se debe borrar la casilla anterior y será esa la casilla seleccionada.
* Si la casilla seleccionada no tiene contenido y es la primera letra, entonces esta casilla sigue seleccionada y no ocurre nada más.

**Criterios aceptación**

1. Dado que soy usuario, cuando entre al juego, entonces veo 5 casillas en vez de uno.
2. Dado que estoy jugando una nueva partida, cuando pulso una letra, entonces se escribe en la casilla correspondiente según el orden establecido en la descripción.
3. Dado que estoy escribiendo una palabra, cuando pulso el botón borrar, entonces me borra las letras escritas según el orden que aparece en la descripción.
4. Dado que estoy jugando una nueva partida, cuando envío una palabra, entonces se comprueba cuantas letras he acertado y se marcan en los estados que aparecen en la descripción.

#5 Fin de la partida – MUST

**Como** usuario

**Quiero** ganar una partida

**Para** demostrar que no soy un looser

**Descripción**:

En esta HU se desarrollará cuando se gana una partida, que básicamente es cuando se ha acertado la palabra asignada en la partida. Y a continuación se mostrará un mensaje con el siguiente texto "Enhorabuena has acertado la palabra, pero... ¿podrás con la siguiente?". Además, se desactivarán todas las acciones.

En el modal del mensaje habrá un botón para generar una nueva partida con el siguiente texto: ¿Te atreves a otra partida piltrafilla?

**Criterios aceptación**

1. **Dado** que estoy jugando, **cuando** acierto la palabra asignada en la partida, **entonces** se me muestra el mensaje que aparece en la descripción.
2. **Dado** que he ganado, **cuando** veo el modal de haber ganado, **entonces** si pulso en el botón de nueva partida, se me genera una nueva.
3. **Dado** que estoy jugando, **cuando** acierto la palabra asignada en la partida, **entonces** se desactivan todas las acciones de la página.

#6 Limitar la partida a 5 intentos – MUST

**Como** usuario que está jugando

**Quiero** ganar antes de 5 intentos

**Para** ser un crack

**NOTA**

Esta HU es evolución de la #4, por lo tanto, tendrá que desarrollarse después.

**Descripción**:

Una vez que el usuario ya puede jugar con intentos ilimitados, ahora solo podrá hacerlo en 5 intentos.

Para ello cada vez que el usuario envié una palabra que exista y se comprueben las letras, le mostrara una fila nueva de casillas vacías, hasta un máximo de 5 filas.

En cada intento se mostrarán las letras del intento anterior en los estados descritos en la HU #4, para darle pistas de cuantas letras ha acertado en el intento anterior.

Si el jugador ha fallado un total de 5 intentos, se mostrará el mensaje “Has perdido una partida más looser, ¡¡¡espabila!!!", informando que ha perdido la partida, se recuperara la palabra correcta y se mostrará y se le bloquearan las acciones de la pantalla.

Cuando se muestre le modal con el mensaje de haber perdido, habrá un botón con el siguiente texto: ¿Que tal looser, lo vuelves a intentar?, para generar una nueva partida.

**Criterios aceptación**

1. **Dado** que estoy jugando, **cuando** envío una palabra, **entonces** se me muestran las letras en sus diferentes estados.
2. **Dado** que estoy jugando, **cuando** envío una palabra, **entonces** se me muestra una nueva fila para introducir una nueva palabra hasta un máximo de 5 filas.
3. **Dado** que estoy jugando, **cuando** he agotado los 5 intentos sin acertar la palabra, **entonces** veo el mensaje de la descripción, la palabra del correcta, además de bloquearle todas las acciones.
4. **Dado** que estoy jugando, **cuando** he agotado los 5 intentos sin acertar la palabra, **entonces** puedo crear una nueva partida pulsando el botón del modal.

#7 Login (validar usuario) – SHOULD

**Como** usuario

**Quiero** que el juego tenga una página inicial de login que me permita identificarme

**Para** poder guardar los resultados de las partidas que juego

**Descripción**:

En esta historia vamos a desarrollar un formulario de Login que nos permita identificarnos.

Sobre los **campos**:

* El formulario dispondrá de dos campos: uno para el usuario y otro campo para la contraseña. Y también se añadirá un botón para poder validar los datos introducidos contra un servicio de back que comprobará la información.
* El campo de usuario es de tipo texto. El campo será requerido y se exigirá un mínimo de 4 caracteres y sin caracteres especiales.
* El campo de contraseña es de tipo password y por tanto cuando se escriba en él no debe aparecer el texto escrito sino caracteres “\*”. El campo es requerido y no admitirá menos de 6 caracteres ni más de 12.

Si hay algún error en los campos, se mostrará en la parte inferior de dicho campo un error que indique qué validación no se cumple:

* El campo es requerido.
* Debe tener al menos N caracteres.
* No debe superar N caracteres.
* No se admiten caracteres especiales.

Sobre la **encriptación** de datos:

* Los datos del formulario serán enviados encriptados en base64 al servicio de back para que éste lo valide.

Sobre el **token**:

* Si al enviar el formulario el servicio nos devuelve los datos correctamente, se recibirá un token que se deberá de utilizar en la cabecera de autenticación del frontend para el resto de las peticiones a servicios del juego.
* En este token debe ir la info necesaria del usuario codificada para poder realizar las consultas pertinentes y almacenar los datos necesarios de las partidas, ya que en el body de las peticiones del resto de servicios no se mandará nada relacionado con el usuario.
* El token tendrá una caducidad de 15 minutos. Si desde el front se realiza una petición con un token caducado, recibirá un error de status 401 y el usuario será redirigido al Login de nuevo para que vuelva a identificarse y, por tanto, obtener un nuevo token válido.

**Criterios aceptación**

1. **Dado** un usuario , **cuando** accedo al juego y no tengo un token activo, **entonces** el juego me redirige a la página de login.
2. **Dado** un usuario , **cuando** accede a la página de login, **entonces** ve un formulario con un campo "usuario", un campo "password" y un botón "enviar" (deshabilitado)
3. **Dado** un usuario , **cuando** tenga los dos campos de forma correcta, **entonces** el botón "enviar" se habilita
4. **Dado** un usuario , **cuando** rellena un campo de formulario Y el dato es erróneo **entonces** se mostrará en la parte inferior de dicho campo un error que indique qué validación no se cumple (errores descritos en la descripción).
5. **Dado** un usuario , **cuando** accede a un campo de formulario que daba error para solucionarlo **entonces** dejará de mostrarse el error hasta que acabe el proceso y se vuelva a comprobar.
6. **Dado** un usuario , **cuando** pulsa en el botón enviar y está deshabilitado **entonces** no pasa nada.
7. **Dado** un usuario , **cuando** el botón "enviar" está habilitado y pulsa en él **entonces** se envía la información para que sea validada.
8. **Dado** un usuario , **cuando** pulsa sobre la tecla de “enter” y el formulario no es válido **entonces** no ocurrirá nada.
9. **Dado** un usuario , **cuando** pulsa sobre la tecla de “enter” y el formulario es válido **entonces** se comportará de la misma manera que si se pulsa el botón de enviar.
10. **Dado** un usuario , **cuando** envía el formulario y ocurre un error 401 **entonces** debe mostrarse un mensaje indicando al usuario que la contraseña o el usuario no son correctos.
11. **Dado** un usuario , **cuando** envía el formulario y ocurre un error diferente de 401 **entonces** y debe mostrar un mensaje al usuario indicando que ha ocurrido un error inesperado y que lo intente de nuevo pasados unos minutos.
12. **Dado** un usuario , **cuando** envía el formulario y el servicio devuelve los datos correctamente **entonces** se recibirá un token que se deberá de utilizar en la cabecera de autenticación del frontend para el resto de las peticiones a servicios del juego (explicación completa en la descripción).

#9 Histórico de las 10 últimas partidas - SHOULD

**Como** usuario

**Quiero** tener una página de información de resultados

**Para** poder ver el resultado de mis últimas 10 partidas

**NOTA**

Esta HU es evolución de la #7, por lo tanto, tendrá que desarrollarse después.

**Descripción:**

Para consultar el histórico de partidas del usuario deberá de añadirse un enlace en la barra de navegación.

Cuando el usuario pulsa sobre el enlace se redirigirá a la página de histórico donde podrá consultar las partidas que ha jugado, en concreto mostraremos el listado de las últimas 10 partidas con fecha y hora.

**Criterios aceptación**

1. **Dado** un usuario, **cuando** accede al juego **entonces** tendrá disponible un enlace en la barra de navegación "Histórico".
2. **Dado** un usuario, **cuando** accede a la página de histórico **entonces** verá un listado con los resultados de las últimas 10 partidas
3. **Dado** un usuario, **cuando** accede al juego y ve en el litado una partida ganada **entonces** la partida debe verse en verde
4. **Dado** un usuario, **cuando** accede al juego y ve en el litado una partida perdida **entonces** la partida debe verse en rojo

#10 Ver todas las partidas - SHOULD

**Como** usuario

**Quiero** ampliar la información de la página de "Histórico" añadiendo un enlace "ver todas las partidas"

**Para** poder ver en una ventana modal mi listado completo de partidas

**NOTA**

Está HU tiene una dependencia con la HU9: Histórico de las 10 últimas partidas

**Descripción**:

Debajo del listado de mis 10 últimas partidas se añadirá un enlace de “ver todas las partidas” que al pulsarlo abrirá una ventana modal con el listado completo de las partidas del usuario.

Si el usuario no dispone todavía de más de 10 partidas jugadas, no se mostrará ese enlace.

Al pinchar en el enlace:

* Se realizará una petición para recuperar todas las partidas del usuario.
  + Si falla, y no es un error con status 401, se mostrará un mensaje informando del error al usuario, este error debe venir informado en el campo message del backend.
  + Si la petición funciona correctamente, se obtendrá el listado que debe mostrarse en una modal del mismo modo que se mostraban las 10 últimas partidas (mismos estilos).

**Criterios aceptación**

1. **Dado** un usuario **cuando** accede a la página de histórico y no ha jugado más de 10 partidas **entonces** no se verá el enlace "ver todas las partidas"
2. **Dado** un usuario **cuando** accede a la página de histórico y ha jugado más de 10 partidas **entonces** aparecerá un enlace "ver todas las partidas" debajo del listado
3. **Dado** un usuario **cuando** pulsa el enlace "ver todas las partidas" y se produce un error **entonces** se mostrará un mensaje informando del error al usuario, este error debe venir informado en el campo message del backend.
4. **Dado** un usuario **cuando** pulsa el enlace "ver todas las partidas" y todo es correcto **entonces** se abrirá una ventana modal mostrando el listado completo de partidas del usuario (con el mismo estilo del listado de 10 últimas partidas definido en la HU8)

#12 Mostrar en el histórico las 3 mejores partidas - SHOULD

**Como** usuario

**Quiero** ampliar la información de la página de "Histórico"

**Para** ver el resultado de mis 3 mejores partidas

**NOTA**

Está HU tiene una dependencia con la HU8: Histórico de las 10 últimas partidas

**Descripción**:

Añadiremos un bloque con las 3 mejores partidas del usuario -> Aciertos con menor número de intentos.

**Criterios aceptación**

1. **Dado** un usuario **cuando** accede a la página de histórico **entonces** verá un bloque con sus 3 mejores partidas.

ESTIMACIÓN Y CREACIÓN DE TAREAS

Cuando ya se han creado todas las HU necesarias para poder desarrollar la aplicación según los requisitos del cliente, se estima cada una de ellas según el esfuerzo que puede costar cada una, y se acuerdan las tareas que tendrá cada historia de usuario.

*\* El esfuerzo se mide según la dificultad, la incertidumbre y experiencia.*

#HU1 - EN-crear seed

**Estimación** 🡪 1

**Tareas** 🡪

* [**Back**] Crear repositorio y README.
* [**Back**] Crear proyecto Java e instalar dependencias.
* [**Front**] Crear repositorio y README.
* [**Front**] Crear proyecto Angular e instalar dependencias.

#HU2 – Verificar existencia palabra

**Estimación** 🡪 3

**Tareas** 🡪

* [**Back**] Plantear y crear base de datos.
* [**Back**] Crear endpoint para verificar la palabra de tipo GET.
* [**Back**] Gestión de errores.
* [**Front**] Crear página principal.
* [**Front**] Crear teclado virtual.
* [**Front**] Crear servicio de validación de la palabra.
* [**Front**] Gestión de errores.

#HU3 – Nueva partida

**Estimación** 🡪 1

**Tareas** 🡪

* [**Back**] Crear tabla “game”.
* [**Back**] Crear endpoint para crear una nueva partida de tipo POST.
* [**Back**] Gestión de errores.
* [**Front**] Crear modal para los mensajes de error.
* [**Front**] Crear servicio de nueva partida.
* [**Front**] Gestión de errores.

#HU4 – Validación de la palabra

**Estimación** 🡪 13

**Tareas** 🡪

* [**Back**] Crear endpoint para validar las posiciones de las letras de la palabra
* [**Back**] Implementar servicio para validar las posiciones de las letras de la palabra.
* [**Back**] Gestión de errores.
* [**Front**] Crear servicio para enviar las letras.
* [**Front**] Implementar la funcionalidad para borrar letras.
* [**Front**] Implementar la funcionalidad para escribir letras.
* [**Front**] Implementar la funcionalidad para establecer el estado de las letras.
* [**Front**] Crear componente de casilla.
* [**Front**] Gestión de errores.

#HU5 – Fin de partida

**Estimación** 🡪 2

**Tareas** 🡪

* [**Front**] Crear modal con el mensaje de fin de partida y botón de nueva partida.
* [**Front**] Desactivar las opciones de la página.

#HU6 – Limitar la partida a 5 intentos

**Estimación** 🡪 5

**Tareas** 🡪

* [**Back**] Comprobar que la partida no supera los 5 intentos.
* [**Back**] Devolver palabra correcta cuando pierde.
* [**Back**] Gestión de errores.
* [**Front**] Crear nueva línea de letras cuando termina un intento.
* [**Front**] Crear modal de partida perdida y creación de nueva partida.
* [**Front**] Gestión de errores.

#HU7 – Login (validar usuario)

**Estimación** 🡪 5

**Tareas** 🡪

* [**Back**] Crear tabla y entidad “user”.
* [**Back**] Configurar Spring Security.
* [**Back**] Gestión de tokens JWT.
* [**Back**] Implementar servicio y controller de autenticación.
* [**Back**] Gestión de errores.
* [**Front**] Crear componente login.
* [**Front**] Crear servicio de tokens JWT.
* [**Front**] Crear interceptor de login
* [**Front**] Gestión de errores.

#HU9 – Histórico de las 10 últimas partidas

**Estimación** 🡪 3

**Tareas** 🡪

* [**Back**] Crear endpoint que devuelva las 10 últimas partidas con fecha, y estado.
* [**Back**] Gestión de errores.
* [**Front**] Crear componente histórico.
* [**Front**] Gestión de errores.

#HU10 – Histórico de todas las partidas

**Estimación** 🡪 3

**Tareas** 🡪

* [**Back**] Crear endpoint que devuelva todas las partidas del usuario.
* [**Back**] Gestión de errores.
* [**Front**] Crear implementación para ver todas las partidas.
* [**Front**] Gestión de errores.

#HU12 – Mostrar en el histórico las 3 mejores partidas

**Estimación** 🡪 2

**Tareas** 🡪

* [**Back**] Crear endpoint que devuelva las 3 mejores partidas del usuario
* [**Back**] Gestión de errores.
* [**Front**] Añadir tabla con las 3 mejores partidas del usuario.
* [**Front**] Gestión de errores.

### 3.2.2 DISEÑO

Una vez tenemos definidos los requisitos de la aplicación y como se desarrollará, pasamos a analizar estos y diseñar como será nuestra aplicación.

ACTORES DEL SISTEMA

Primero definiremos que posibles actores podrá tener la aplicación:

Forma

Descripción generada automáticamente con confianza baja

**Usuario identificado** 🡪 El usuario identificado es un usuario que ha iniciado sesión en la aplicación con su nombre de usuario y contraseña y puede acceder a todos los recursos de esta.

Forma

Descripción generada automáticamente con confianza baja

**Usuario no identificado** 🡪 El usuario no identificado es un usuario que no ha iniciado sesión en la aplicación por lo tanto no podrá ni jugar ni acceder a ningún recurso de la aplicación.

DIAGRAMA DE CASOS DE USO

Cuando ya hemos definido que actores tendrá nuestra aplicación pasamos a realizar el diagrama de casos de uso, donde contemplaremos todas las casuísticas que pueden pasar, según la acción que realice algún actor de nuestro sistema.

Diagrama, Esquemático

Descripción generada automáticamente

CASOS DE USO

Después de realizar el diagrama de casos de uso pasaremos a explicar cada caso de uso detalladamente para que quede claro cómo se desarrolla cada acción del actor:

UC 1. INICIAR SESIÓN

|  |  |
| --- | --- |
| **Id y nombre** | UC 1. Iniciar sesión |
| **Descripción** | El usuario inicia sesión con su nombre y contraseña |
| **Actores** | Usuario no identificado |
| **Precondiciones** | El usuario no debe estar identificado. |
| **Secuencia principal** | 1. El usuario introduce su nombre de usuario y contraseña 2. El sistema valida los datos introducidos por el usuario. 3. El sistema identifica al usuario. |
| **Flujos alternativos** | UsuarioOContraseñaNoValidos  FormatoDeDatosIncorrecto |
| **Postcondiciones** | Se ha autenticado al usuario y puede acceder a los recursos |

|  |  |
| --- | --- |
| **Id y nombre** | FA 1.1 Iniciar sesión: UsuarioOContraseñaNoValidos |
| **Descripción** | El sistema informa al usuario que su usuario o contraseña no son correctos. |
| **Actores** | Usuario no identificado |
| **Precondiciones** | El usuario ha introducido un nombre o una contraseña que no son válidos. |
| **Flujos alternativos** | 1. Comienza después del paso 2 del flujo principal 2. El sistema informa al usuario que ha introducido un usuario o una contraseña no válidos. – Mostrar error de autenticación |
| **Postcondiciones** | El usuario no está autenticado |

|  |  |
| --- | --- |
| **Id y nombre** | FA 1.2 Iniciar sesión: FormatoDeDatosIncorrecto |
| **Descripción** | El sistema informa al usuario que los datos introducidos no tienen el formato correcto |
| **Actores** | Usuario no identificado |
| **Precondiciones** | El usuario ha introducido un nombre con menos de 4 caracteres, o no lo ha introducido.  El usuario ha introducido una contraseña con menos de 6 caracteres, más de 12 caracteres o no la ha introducido |
| **Flujos alternativos** | 1. Comienza después del paso 2 del flujo principal 2. El sistema informa al usuario que ha introducido un usuario o una contraseña con formato incorrecto. – Mostrar error de autenticación |
| **Postcondiciones** | El usuario no está autenticado |

UC 2. JUGAR UNA PARTIDA

|  |  |
| --- | --- |
| **Id y nombre** | UC 2. Jugar una partida |
| **Descripción** | El usuario juega una partida para adivinar la palabra correcta. |
| **Actores** | Usuario identificado |
| **Precondiciones** | El usuario está identificado. |
| **Secuencia principal** | 1. El usuario accede a la página principal. 2. Se crea una nueva partida. 3. El usuario introduce una palabra existente a través del teclado virtual. 4. Si la palabra es correcta   4.1 Se muestra un mensaje de partida ganada   1. Si no    1. Se comprueban las letras introducidas y adoptan el estado de gris (no está en la palabra), amarillo (está en la palabra, pero no en la posición correcta), verde (está en la posición correcta)    2. Se añade una nueva línea con 5 letras, para otro intento. |
| **Flujos alternativos** | MostrarErrorDePalabraInexistente  PartidaPerdida |
| **Postcondiciones** | Se termina la partida y se guarda como ganada o perdida. |

|  |  |
| --- | --- |
| **Id y nombre** | FA 2.1 Jugar una partida: MostrarErrorDePalabraInexistente |
| **Descripción** | El sistema informa al usuario de que la palabra no existe. |
| **Actores** | Usuario identificado |
| **Precondiciones** | El usuario ha introducido una palabra que no existe. |
| **Flujos alternativos** | 1. Comienza después del paso 2 del flujo principal 2. El sistema informa al usuario de que la palabra no existe |
| **Postcondiciones** | El usuario puede seguir intentado adivinar la palabra. |

|  |  |
| --- | --- |
| **Id y nombre** | FA 2.2 Jugar una partida: PartidaPerdida |
| **Descripción** | El sistema informa al usuario que ha perdido la partida |
| **Actores** | Usuario identificado |
| **Precondiciones** | El usuario ha hecho 5 intentos de adivinar la palabra y no ha acertado |
| **Flujos alternativos** | 1. Comienza después del paso 5 del flujo principal 2. El sistema informa al usuario que ha perdido la partida |
| **Postcondiciones** | Se termina la partida y se guarda la partida como partida perdida. |

UC 3. ACCEDER AL HISTORIAL DE PARTIDAS

|  |  |
| --- | --- |
| **Id y nombre** | UC 2. Acceder al historial de partidas. |
| **Descripción** | El usuario accede a su historial de partidas. |
| **Actores** | Usuario identificado |
| **Precondiciones** | El usuario está identificado. |
| **Secuencia principal** | 1. El usuario hace clic en el enlace “Historial” 2. Se listan las últimas 10 partidas. 3. Se listan los datos del usuario. 4. Se listan las 3 mejores partidas del usuario. |
| **Flujos alternativos** | ListarTodasLasPartidas |
| **Postcondiciones** | No hay. |

|  |  |
| --- | --- |
| **Id y nombre** | FA 3.1. Acceder al historial de partidas: ListarTodasLasPartidas |
| **Descripción** | El usuario muestra todas sus partidas |
| **Actores** | Usuario identificado |
| **Precondiciones** | El usuario hace clic en el botón de “Ver todas las partidas” |
| **Secuencia principal** | 1. Comienza después del paso 2 del flujo principal 2. Se listan todas las partidas del usuario. |
| **Flujos alternativos** | ErrorInsuficientesPartidas. |
| **Postcondiciones** | El usuario ha listado todas sus partidas y las puede volver a ocultar para ver solo las 10 últimas. |

|  |  |
| --- | --- |
| **Id y nombre** | FA 3.1.1 ListarTodasLasPartidas - ErrorInsuficientesPartidas. |
| **Descripción** | El sistema informa al usuario de que no tiene suficientes partidas. |
| **Actores** | Usuario identificado |
| **Precondiciones** | El usuario hace clic en el botón de “Ver todas las partidas”, y no tiene suficientes partidas. |
| **Secuencia principal** | 1. Comienza después del paso 2 del flujo alternativo - ListarTodasLasPartidas 2. El sistema muestra un error al usuario informándole de que no tiene más de 10 partidas. |
| **Flujos alternativos** | ErrorInsuficientesPartidas. |
| **Postcondiciones** | El usuario cierra el mensaje y se muestran las 10 últimas partidas. |

DIAGRAMA ENTIDAD – RELACIÓN

Al terminar el diseño de cómo será la funcionalidad de la aplicación ya podemos pasar a diseñar como será técnicamente, para esto primero definiremos el modelo de datos de la aplicación. Usaremos un modelo relacional en el cual se relacionarán entre si las entidades User, Word y Game. Para ver mejor como serán estas relaciones hemos diseñado un diagrama entidad-relación:

Diagrama

Descripción generada automáticamente

Como vemos en nuestra aplicación habrá 2 relaciones, User-Game y Word-Game.

**User** 🡪 Tabla que guardará la información de los usuarios de la aplicación.

**Game** 🡪 Tabla que guardara la información de las partidas que juegue un usuario. Entidad principal de nuestra aplicación.

* Se relaciona con User ya que una partida pertenece a un usuario, mientras que un usuario puede tener muchas partidas
* Se relaciona con Word ya que una partida tiene una palabra correcta o ganadora asociada, pero una misma palabra del diccionario, puede ser la palabra ganadora de muchas partidas.

**Word** 🡪 Tabla que define todas las palabras de nuestro diccionario, las cuales se utilizaran para poder, y saber cuál es la palabra correcta de una partida

DIAGRAMA DE TABLAS

Una vez hemos diseñado el diagrama entidad – relación de nuestra aplicación ya podemos diseñar las tablas de la base de datos SQL que usaremos para almacenar información. Siguiendo las pautas y según las relaciones creadas anteriormente, deducimos que para relacionar Game con Word añadiremos una clave foránea a Game que será word\_id, y para la relación entre Game y User deberemos de añadir una clave foránea user\_id a Game. El diagrama de tablas quedaría así:

Diagrama

Descripción generada automáticamente

*\* Además de añadir las relaciones a las tablas también hemos añadido las restricciones consideradas a cada campo de cada tabla.*

\*Descripción de cómo se trabaja con scrum en un sprint, git Flow, desarrollo de un endpoint en java, desarrollo de game service en angular\*

### 3.2.3 IMPLEMENTACIÓN

Después de tener ya diseñada la aplicación, ya sabemos cómo va a ser, por lo tanto, podemos pasar a la fase de implementación de esta.

Al trabajar con Scrum esta fase se divide en sprints, que son fases de entre 2 y 4 semanas donde se desarrolla la aplicación. En este caso cada sprint del proyecto será de dos semanas.

Cada sprint se divide en 4 fases:

* **Sprint plannig** 🡪 Ceremonia de scrum donde se planifica el sprint, se estiman las historias de usuario, se detallan las tareas que tendrá cada tarea, se eligen cuantas historias se desarrollarán en el sprint y se define el sprint goal (el objetivo del sprint)
* **Desarrollo** 🡪 Después del sprint planning se empieza a desarrollar la aplicación, implementando las HU elegidas en el sprint planning. En esta fase, se hacen reuniones diarias (dailys) donde todo el equipo pone en común que ha hecho desde la daily pasada, que va a hacer y si tiene algún problema o bloqueo.
* **Spring review** 🡪 Cuando finaliza el sprint el equipo de desarrollo presenta el trabajo hecho durante el sprint a los steak holders (clientes, miembros ajenos a la implementación), realizando una demo del resultado.
* **Sprint retrospective** 🡪 Después de terminar cada sprint se celebra una ceremonia donde todo el equipo de desarrollo pone en común que ha ido bien durante el sprint, que se puede mejorar y que ha ido mal, y como solucionarlo para trabajar mejor.

Imagen que contiene reloj

Descripción generada automáticamente

GIT FLOW Y KANBAN

Dentro de cada sprint se trabaja con el control de versiones GIT, donde se va subiendo el progreso diario, para tener guardado todo lo que se ha desarrollado y poder recuperar cualquier versión de la aplicación. Al trabajar en equipo GIT también nos permite partir desde el progreso de otro compañero, trabajar simultáneamente en el mismo proyecto y poder fusionar el trabajo.

Además de trabajar con GIT también se trabaja con la metodología Kanaban, que consiste en un tablero donde se registra el estado de las tareas de un Sprint.

**¿Cómo se trabaja con GIT y Kanban en un sprint?**

Bien, para trabajar en GIT y Kanban dentro de un sprint seguimos el GIT Flow de la empresa, es decir el flujo de trabajo con GIT que nos marca la empresa. Este flujo consiste en 3 ramas:

* **Feature**🡪 Las ramas features son ramas temporales que se crean para la implementación de una nueva funcionalidad, normalmente se crea una rama feature por tarea.
* **Develop** 🡪 La rama develop es donde hay una versión estable de la aplicación que va incrementando su valor a lo largo del sprint, se van mergeando features en esta, pero siempre debe ser una versión estable.
* **Master** 🡪 Al final de cada sprint o cada dos se sube el progreso de develop a master como en un entorno de producción, será lo que se despliegue en el servidor.

Gráfico, Gráfico de líneas

Descripción generada automáticamente

Ahora que ya sabemos que ramas se utilizan durante la fase de implementación, pondré un ejemplo de cómo sería el **desarrollo de una feature** en un sprint:

* Se elige una tarea (asignación en el tablero KANBAN / estado **NEW**), se crea la rama de la tarea, **feat\_*numero-ref-tarea****,* se trabaja en ella (pasa a estado **IN PROGRESS**), cuando está terminada y pasa todos los tests se hace un *merge request*[[1]](#footnote-2) (pasa a estado **MERGE REQUEST**), cuando se aprueba el merge request se **mergea** la rama **feature** **con** la rama **develop** y **se elimina** la rama **feature**, una vez hecho este proceso se prueba la funcionalidad (estado **READY FOR TEST**), y si todo va bien se mantiene como **versión estable en develop** y la tarea pasa a estado **CLOSED** en el tablero kanban.

Interfaz de usuario gráfica

Descripción generada automáticamente con confianza media

Imagen que contiene Forma

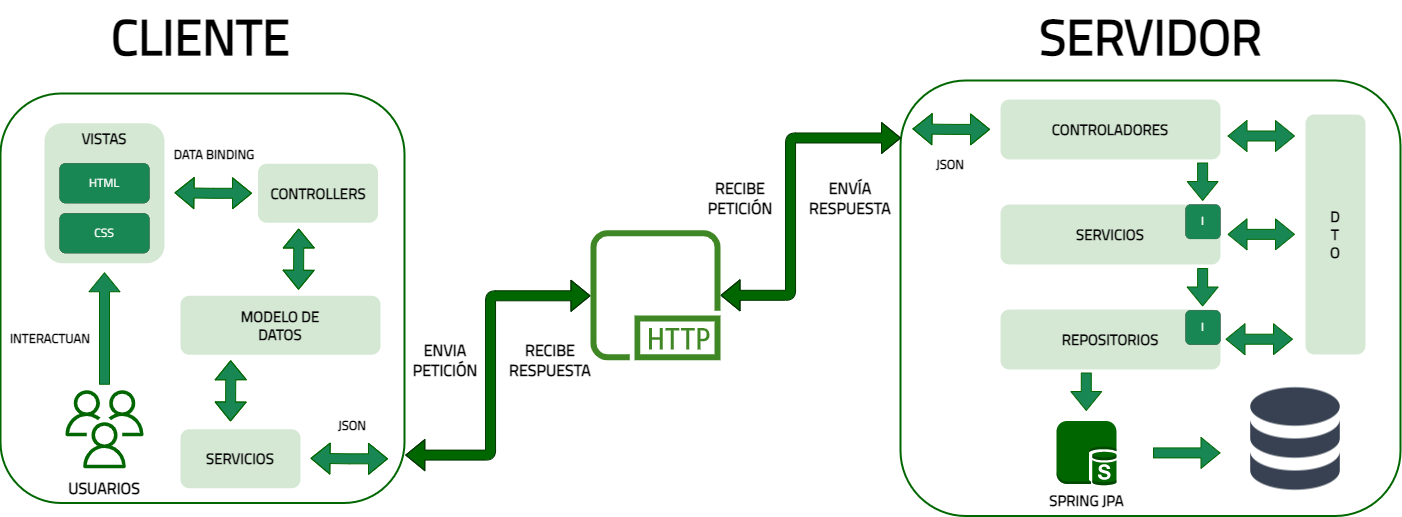
Descripción generada automáticamente

ARQUITECTURA

Una vez detallado como vamos a trabajar en el proceso de implementación, pasaremos a explicar la arquitectura que se ha usado tanto en back como en front.

Para el desarrollo de la aplicación he usado una arquitectura cliente-servidor. La arquitectura cliente-servidor es un modelo donde hay dos partes: el cliente y el servidor. El cliente es la parte visual que utiliza el usuario para interactuar con la aplicación, es la parte del navegador, la cual se comunica con el servidor atreves de peticiones mediante el protocolo HTTP. El servidor recibe la petición, y devuelve la respuesta al cliente. *\*Toda la comunicación cliente-servidor se ha hecho mediante objetos JSON*

Por la parte cliente se ha utilizado una arquitectura MVC (Modelo-Vista-Controlador) trabajando como SPA (Single Page Application), esto quiere decir que en la parte del navegador solo se trabaja con una página, que va cambiando el contenido según las interacciones del usuario. El MVC con Angular consiste en el Modelo que especifica cómo van a ser los datos con los que se trabajen, la vista que es el HTML y CSS donde se muestra la parte visual de la aplicación, los controladores que trabajan con los datos introducidos por el usuario para enviarlos al servicio donde se hacen las peticiones HTTP al servidor.

En relación con la parte servidor se ha utilizado una estructura monolítica, con una arquitectura por capas. Esta arquitectura consiste en dividir la API REST del servidor, en 3 capas, Controllers, Services y Respositories. Esta arquitectura utiliza DTO (Data Transfer Object) para comunicarse entre capas. La capa del controlador recibe la petición y transforma el JSON que recibe un objeto DTO, una vez convertido se lo pasa al servicio donde se hace toda la lógica de negocio, y este servicio utiliza métodos de los repositorios para la persistencia de datos que en este caso es en una base de datos SQL.

SCAFFOLDING

Ahora que ya sabemos la arquitectura que he usado pasare a explicar la estructura de carpetas del proyecto, explicaré la estructura general de la aplicación y después explicaré la estructura de cada parte. SERVIDOR - CLIENTE

Imagen que contiene Texto

Descripción generada automáticamenteGENERAL

* **/db** 🡪 Ficheros ".sql" de creación de la base de datos, etc. Es todo lo necesario para tener la base de datos preparada desde cero. Incluye insertar datos que siemrpe deben estar , como por ejemplo el diccionario de palabras.
* **/design** 🡪 Imágenes en formato Photshop o similares que estamos visualmente diseñando
* **/dist** 🡪 El código "compilado" de la aplicación. Eso incluye transformar de TypeScript a JavaScript y de SASS a CSS,etc.
* **/docs** 🡪 Documentación de la aplicación. Es la documentación con los diagramas UML, el JavaDoc, etc.
* **/scripts** 🡪 Fichero de ejecución para desplegar la aplicación.
* **/src** 🡪 Código fuente de la aplicación.
  + **/back** 🡪 Código de la API REST de la aplicación, parte del servidor.
  + **/front** 🡪 Código de la parte visual de la aplicación, parte del navegador.
* **.gitignore** 🡪 Fichero GIT donde se indica que fichero no deben de añadirse al hacer un git add
* **LICENSE** 🡪 Licencia del proyecto
* **README.md** 🡪 Archivo md donde esta la presentación del proyecto y como usarlo.

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamenteCLIENTE

* **.angular** 🡪
* **/dist** 🡪
* **/node\_modules** 🡪
* **/src** 🡪
  + **/app** 🡪
    - **/components** 🡪
    - **/interceptors** 🡪
    - **/interfaces** 🡪
    - **/modules** 🡪
    - **/services** 🡪
    - **app-routing-module.ts** 🡪
    - **app.component.html** 🡪
    - **app.component.scss** 🡪
    - **app.component.spect.ts** 🡪
    - **app.component.ts** 🡪
    - **app.moudle.ts** 🡪
  + **/assets** 🡪
  + **favicon.ico** 🡪
  + **index.html** 🡪
  + **main.ts** 🡪
  + **style.scss** 🡪
  + **angular.json** 🡪
  + **package-lock.json** 🡪
  + **package.json** 🡪
  + **tsconfig.app.json** 🡪
  + **tsconfig.json** 🡪
  + **tsconfig.spec.json** 🡪

Texto

Descripción generada automáticamenteSERVIDOR

* **.mvn** 🡪
* **/src** 🡪
  + **/main** 🡪
    - **/java** 🡪
      * **/config** 🡪
      * **/controllers** 🡪
      * **/dto** 🡪
      * **/entities** 🡪
      * **/exceptions** 🡪
      * **/repositories** 🡪
      * **/request** 🡪
      * **/security** 🡪
      * **/services** 🡪
      * **/utilities** 🡪
      * **AdivinaLaPalabraApplication** 🡪
    - **/resources** 🡪
      * **application.yml** 🡪
  + **/test** 🡪
    - **/java** 🡪
      * **/controllers** 🡪
      * **/exceptions** 🡪
      * **/helpers** 🡪
      * **/security** 🡪
      * **/services** 🡪
      * **/utilities** 🡪
      * **AdivinaLaPalabraApplicationTest** 🡪
    - **/resources** 🡪
      * **application.yml** 🡪
    - **/target** 🡪
    - **mvnw** 🡪
    - **mvnw.cmd** 🡪
    - **pom.xml** 🡪

DESARROLLO

Una vez tenemos claro cómo se va a organizar la aplicación pasaré a explicar cómo ha sido desarrollada.

Para esto explicaré el proceso de jugar de una partida, desde que se crea hasta que se gana o se pierde. Así se podrá ver el paso de información entre todas las capas de la arquitectura explicada antes, y como es la implementación de los endpoints de la aplicación el cual implica:

* Diseño de interfaz gráfica
* Lógica en la parte del navegador
* Envío de peticiones HTTP
* Enrutamiento de endpoints en back, y retorno de información mediante controladores.
* Paso de información al servicio e implementación de lógica de negocio en este.

IMPLEMENTACIÓN

LOGIN

CREACIÓN DE UNA PARTIDA

INTENTO DE UNA PALABRA

PARTIDA GANADA O PERDIDA

HISTORIAL DE PARTIDAS

### 3.2.4 PRUEBAS

TDD

COBERTURA DE TEST

TEST BACKEND

TEST EN CONTROLADORES

TEST EN SERVICIOS

TEST DE EXCEPCIONES

FRONTEND

TEST DE COMPONENTES

## 3.3 PROPUESTAS DE MEJORA

\*HU de usuario pendientes, limitar una partida al día (atraer al cliente), deuda técnica, explicación de la palabra al terminar la partida\*

# 4. MANUAL DE USO

# 5. VALORACIÓN PERSONAL

# 6. FUENTES BIBLIOGRÁFICAS

1. *Merge request* 🡪 Un merge request o pull request es la petición de combinación entre dos ramas, normalmente las pull request se realizan cuando se ha terminado una tarea, esta testeada y se quiere pasar como versión estable a develop, se mergea la rama feature con la rama develop. [↑](#footnote-ref-2)