

PROYECTO FIN DE CICLO

Lucas Chacon Langa



RIVALS

Desarrollo de aplicaciones web - IES Francisco de Goya

1. Introducción.....	4
1.1 Objetivos del Proyecto.....	4
1.1.1 Objetivos Técnicos.....	4
1.1.2 Inspiración y Justificación del Proyecto.....	5
1.2 Metodología de Trabajo.....	6
1.2.1 Organización Personal del Desarrollo.....	6
1.2.2 Notion.....	6
1.2.3 Ciclos Iterativos y Tiempos.....	6
1.2.4 Aplicación de Principios Ágiles (Scrum).....	6
1.3 Buenas Prácticas en el Desarrollo.....	7
1.3.1 Clean Code.....	7
1.3.2 Control de Errores y Excepciones.....	7
1.4 Tecnologías Utilizadas.....	8
1.4.1 Frontend.....	8
1.4.1.1 HTML5.....	8
1.4.1.2 CSS3.....	8
1.4.1.3 Tailwind CSS.....	8
1.4.1.4 JavaScript.....	8
1.4.1.4.1 Chart.js.....	8
1.4.1.4.2 SweetAlert2.....	9
1.4.1.5 Google Fonts.....	9
1.4.1.6 Fuente externa: Sonic Extra Bold.....	9
1.4.2 Backend.....	9
1.4.2.1 Java.....	9
1.4.2.2 Spring Boot.....	9
1.4.2.3 OpenAI API.....	9
1.4.2.4 Google Auth.....	10
1.4.3 Base de Datos.....	10
1.4.3.1 H2 (Desarrollo).....	10
1.4.3.2 PostgreSQL (Producción).....	10
1.4.4 Herramientas y Entornos.....	10
1.4.4.1 Git & GitHub.....	10
1.4.4.2 Maven.....	10
1.4.4.3 IntelliJ IDEA.....	11
1.4.4.4 Heroku.....	11
1.4.4.5 Servidor Personal (Desarrollo).....	11
2. Análisis de Requisitos.....	12
2.1 Análisis Funcional y No Funcional.....	12
2.2 Casos de Uso y Experiencia de Usuario.....	13
2.3 Mapa de Pantallas y Navegación.....	16
2.4 Manual de Usuario (UX/UI).....	18
2.4.1 Landing Page.....	19
2.4.2 Registro e Inicio de Sesión.....	20
2.4.3 Ranking.....	22
2.4.4 Fighters.....	23
2.4.5 Simulador.....	24

2.4.6	Página de Usuario.....	26
2.4.7	Gestión de Favoritos.....	27
2.4.8	Política de Privacidad y Seguridad.....	28
2.4.9	Páginas de Error.....	29
3.	Configuración del Entorno de Desarrollo.....	31
3.1	Creación del Repositorio y Estructura de Carpetas.....	31
3.2	Git: Flujo de Trabajo y Comandos.....	31
3.2.1	Git Init / Clone / Push / Pull / Restore.....	31
3.2.2	Buenas Prácticas de Commits.....	31
3.3	Instalación de Herramientas.....	33
3.3.1	Java (Windows y Linux).....	33
3.3.2	Maven (Windows y Linux).....	33
3.3.3	IntelliJ IDEA.....	33
3.3.4	Navegadores y Testing.....	34
3.4	Gestión de Dependencias con Maven.....	34
3.5	CDNs.....	35
4.	Diseño y Desarrollo de la Base de Datos.....	36
4.1	Estructura de la Base de Datos.....	36
4.1.1	Diagrama.....	36
4.1.2	Tablas y Campos.....	37
4.1.2.1	fighter.....	37
4.1.2.2	user_app.....	37
4.1.2.3	fav_fighter.....	37
4.1.2.4	fav_fight.....	38
4.1.3	Claves Primarias, Externas y Relaciones.....	38
4.2	Estrategia de Transición: H2 → PostgreSQL.....	38
4.3	Seguridad e Integridad de Datos.....	38
4.4	Testing de Persistencia y Validaciones.....	38
5.	Arquitectura del Proyecto y Desarrollo.....	40
5.1	Estructura del Proyecto Spring Boot.....	40
5.2	Patrón MVC Aplicado.....	41
5.3	Controladores (Controllers).....	42
5.3.1	MainController.....	42
5.3.2	ErrorController.....	42
5.3.3	RankingController.....	42
5.3.4	FighterController.....	42
5.3.5	SimulatorController.....	42
5.3.6	UserController.....	43
5.3.7	FavoriteController.....	43
5.3.8	OpenAIController.....	43
5.4	Data Transfer Objects (DTOs).....	44
5.4.1	FighterDto.....	44
5.4.2	UserAppDto.....	44
5.4.3	FavFighterDto.....	44
5.4.4	FavFightDto.....	45
5.5	Repositorios (Repositories).....	45

5.5.1 FighterRepository.....	45
5.5.2 UserAppRepository.....	46
5.5.3 FavFighterRepository.....	46
5.5.4 FavFightRepository.....	46
5.6 Servicios (Services) y Lógica de Negocio.....	47
5.6.1 FighterService.....	47
5.6.2 UserAppService.....	47
5.6.3 FavFighterService.....	48
5.6.4 FavFightService.....	48
5.6.5 CustomOAuth2UserService.....	49
5.6.6 OpenAIClientService.....	49
5.7 Plantillas Thymeleaf y Recursos Estáticos.....	50
5.7.1 Estructura HTML y Fragments.....	50
5.7.2 Tailwind: Responsividad.....	50
5.7.3 JavaScript y librerías Chart.js y SweetAlert 2.....	50
5.7.4 CSS y fuentes.....	51
5.7.5 Imágenes y logos.....	51
5.8 Seguridad.....	52
5.8.1 SecurityConfig (SpringSecurity).....	52
5.8.2 CustomOAuth2UserService.....	52
5.8.3 UserAppInfo.....	53
5.9 Integración con OpenAI.....	53
5.9.1 Prompt Design.....	53
5.9.2 Llamadas a la API y Control de Errores.....	54
5.10 Flujo de Datos.....	54
5.10.1 De Frontend a Backend.....	54
5.10.2 De Backend a la Base de Datos.....	54
5.11 Testing y Validación del Código y Base de datos.....	55
5.11.1 Testing Manual.....	55
6. Despliegue y Producción.....	56
6.1 Despliegue en Heroku.....	56
6.1.1 Configuración del application.properties.....	56
6.1.2 Variables de Entorno y Seguridad.....	57
6.2 Despliegue en Servidor Personal (Desarrollo).....	57
7. Conclusiones Finales.....	57
7.1 Lecciones Aprendidas durante el Proyecto.....	57
7.2 Mejoras Futuras e Ideas de Escalabilidad.....	58
8. Bibliografía y Recursos.....	58
8.1 Documentación Oficial.....	58
8.2 Recursos y Tutoriales Utilizados.....	59
8.3 Créditos a APIs y Librerías Externas.....	59

1. Introducción

El desarrollo del presente proyecto se llevó a cabo con el objetivo de diseñar y construir una aplicación web full stack centrada en el ámbito de las artes marciales mixtas (MMA), un nicho con escasa representación digital. A través de este trabajo, se buscó integrar distintos aspectos del desarrollo de software moderno: desde la organización metodológica y la aplicación de buenas prácticas de programación, hasta el uso de tecnologías actuales, así como la incorporación de inteligencia artificial para enriquecer la experiencia del usuario.

Durante todo el proceso se tuvo en cuenta tanto la dimensión técnica como la experiencia de usuario, priorizando la claridad, escalabilidad y mantenibilidad del código, junto con una navegación fluida y una interfaz atractiva. Este apartado describe los objetivos planteados, la metodología de trabajo adoptada, las buenas prácticas aplicadas y las tecnologías utilizadas a lo largo del desarrollo del proyecto. Estos son los enlaces donde está el proyecto accesible:

Repositorio github: <https://github.com/lucaschacon3/UFC-Rivals>

Despliegue en Heroku: <https://ufc-rivals-cd6647701d82.herokuapp.com>

Despliegue Servidor personal: <http://ufc.rivals.mo00.com>

1.1 Objetivos del Proyecto

En esta sección se presentan los objetivos que guiaron el desarrollo del proyecto, tanto desde una perspectiva técnica como desde su motivación conceptual. Se establecieron dos enfoques: uno centrado en los logros técnicos a alcanzar, y otro en la inspiración que dio origen a la iniciativa.

1.1.1 Objetivos Técnicos

Durante el desarrollo del proyecto, se plantearon y alcanzaron los siguientes objetivos técnicos:

- Se diseñó y construyó una aplicación web con arquitectura cliente-servidor, utilizando **Spring Boot** en el backend y **Thymeleaf** para la generación dinámica de vistas.
- Se desarrolló una base de datos en **PostgreSQL** con gran cantidad de atributos y características para ofrecer unas simulaciones competitivas, además de poder gestionar a usuarios.
- Se implementó un sistema de autenticación seguro mediante **OAuth 2.0**, integrando el servicio de inicio de sesión de **Google**.
- Se desarrolló un **simulador de peleas** interactivo que permite al usuario visualizar resultados de enfrentamientos ficticios, haciendo uso de gráficos dinámicos con **Chart.js**.

- Se diseñó una interfaz basada en principios UX/UI, dando un resultado dinámico y responsivo, además de la implementación de alertas con **SweetAlert 2**.
- Se integró la **API de OpenAI** para generar contenido basado en inteligencia artificial, en concreto una simulación de combate por rounds.
- Se habilitó una gestión de usuarios con posibilidad de marcar **favoritos** y acceder a una página de perfil personalizada.
- Se desplegó el proyecto en entornos de desarrollo y producción, utilizando **Heroku** como plataforma de publicación.
- Se aplicaron prácticas de **Clean Code** y control de errores para asegurar la mantenibilidad y calidad del código.

1.1.2 Inspiración y Justificación del Proyecto

La idea del proyecto surgió del interés personal en abordar un nicho actualmente poco explorado: las artes marciales mixtas. Se planteó como objetivo desarrollar una aplicación web full stack que integrara inteligencia artificial y ofreciera experiencias interactivas. El enfoque se orientó hacia un propósito tanto formativo como demostrativo, con la intención de llevar a cabo:

- Implementación de tecnologías modernas dentro de un entorno completo y funcional.
- Interacción fluida para el usuario, combinando datos estructurados con elementos visuales y simulaciones.
- Poner en práctica conocimientos adquiridos durante la formación, abordando aspectos como desarrollo de software, gestión de datos, diseño de interfaces y despliegue.
- Generar un producto que funcionara como portafolio profesional, demostrando competencias técnicas, organizativas y metodológicas.
- Desarrollar la aplicación íntegramente en inglés, tanto a nivel de código como en la interfaz de usuario, con el objetivo de alcanzar un público más internacional. Esta decisión respondió a que la mayoría de los seguidores de las artes marciales mixtas pertenecen a un entorno global, y además, permitió adoptar buenas prácticas como desarrollador orientado a estándares profesionales y escalabilidad del proyecto.
- Reflexionar sobre la capacidad de la IA como complemento para enriquecer interfaces y funcionalidades web.

1.2 Metodología de Trabajo

Durante el desarrollo del proyecto se adoptó una metodología que permitió organizar las tareas de manera estructurada, flexible y enfocada en la mejora continua. Se recurrió a herramientas digitales de planificación, se aplicaron ciclos iterativos de trabajo y se integraron principios de metodologías ágiles, como Scrum, para gestionar el avance del desarrollo.

1.2.1 Organización Personal del Desarrollo

Desde el inicio, se estructuró el trabajo en fases bien definidas: planificación, desarrollo, pruebas y despliegue. Se establecieron objetivos semanales y se priorizaron tareas en función de su impacto en la funcionalidad general del sistema.

Esta organización personal facilitó un flujo de trabajo constante, evitando acumulaciones y reduciendo la posibilidad de errores críticos.

1.2.2 Notion

Para la gestión de tareas, documentación y seguimiento del proyecto se utilizó **Notion**, una herramienta digital que permitió centralizar la información de manera clara y accesible.

Se crearon paneles organizados por módulos y prioridades, donde se documentaron los avances, incidencias, ideas de mejora y retrospectivas. Además, se emplearon checklists para mantener el control del desarrollo, permitiendo una visión global del estado del proyecto en todo momento.

1.2.3 Ciclos Iterativos y Tiempos

El desarrollo se dividió en ciclos iterativos cortos, similares a sprints, de aproximadamente una semana de duración. Al inicio de cada ciclo se definieron tareas clave y al final se evaluaron los resultados obtenidos.

Este enfoque permitió refinar funcionalidades progresivamente, mantener un ritmo constante de trabajo y realizar entregas incrementales con mejoras continuas.

También se reservaron espacios de tiempo para refactorización del código, pruebas, documentación técnica y mejoras visuales en la interfaz.

1.2.4 Aplicación de Principios Ágiles (Scrum)

Aunque se trató de un proyecto individual, se tomaron como referencia principios de **Scrum** para organizar el trabajo de forma ágil y eficiente. Se llevó un backlog priorizado con tareas pendientes y se realizaron revisiones periódicas para evaluar el avance.

Las decisiones sobre funcionalidades se adaptaron en función de los resultados obtenidos y de nuevas ideas surgidas durante el proceso, fomentando la flexibilidad y la capacidad de respuesta ante cambios.

Esta metodología permitió enfocarse en la entrega de valor constante, garantizando que cada iteración aportara mejoras concretas y funcionales a la aplicación.

1.3 Buenas Prácticas en el Desarrollo

Durante el desarrollo del proyecto se priorizó la calidad del código, la mantenibilidad y la claridad en la estructura del software. Para ello, se implementaron buenas prácticas ampliamente aceptadas en el ámbito del desarrollo profesional, con el fin de asegurar un sistema robusto, escalable y fácil de comprender por otros desarrolladores en el futuro.

1.3.1 Clean Code

Se siguieron los principios de **Clean Code**, priorizando la legibilidad, la simplicidad y la coherencia en la escritura del código fuente.

Se utilizaron nombres descriptivos en variables, funciones y clases, y se evitó la duplicación innecesaria. Asimismo, se estructuraron los métodos para que cumplieran con una única responsabilidad y fueran de longitud reducida, favoreciendo la comprensión y el mantenimiento.

Se desarrollaron clases con funciones reutilizables, complementadas por un archivo común que centraliza funciones empleadas a lo largo de toda la aplicación.

Se diseñaron servicios y controladores, separando la lógica de negocio de la presentación y el acceso a datos. Además, se estructuraron las interfaces para evitar dependencias rígidas, lo que facilitó la evolución del sistema sin comprometer la funcionalidad existente.

Además, se eliminó código comentado o innecesario, y se mantuvo una estructura consistente en cuanto al formato y estilo, lo cual contribuyó a reducir la probabilidad de errores y facilitar la colaboración en caso de futuras ampliaciones.

1.3.2 Control de Errores y Excepciones

Se estableció una estrategia clara para la gestión de errores y excepciones en todo el sistema. Se implementaron bloques **try-catch** en los puntos críticos del backend para capturar posibles fallos y manejar respuestas adecuadas tanto a nivel del servidor como del cliente.

Se diseñaron mensajes de error informativos que fueron mostrados mediante modales con **SweetAlert 2**, mejorando la experiencia de usuario y evitando comportamientos inesperados.

Asimismo, se controlaron errores comunes como inputs inválidos, recursos no encontrados o fallos en la comunicación con APIs externas, especialmente la integración con OpenAI.

1.4 Tecnologías Utilizadas

1.4.1 Frontend

Para la construcción del frontend del proyecto se emplearon tecnologías modernas orientadas al desarrollo web responsivo, accesible y estéticamente atractivo. Se priorizó la separación entre estructura, estilos y comportamiento, y se utilizaron herramientas que permitieran una experiencia de usuario fluida e interactiva.

1.4.1.1 HTML5

Se utilizó **HTML5** como lenguaje base para estructurar el contenido de las vistas. Además, se implementó Thymeleaf, integrado en Spring Boot, lo que permitió el uso de fragments para reutilizar código de manera eficiente. Estos fragments se emplearon principalmente para el header, footer y el head del HTML, donde se incluyen los imports necesarios para el correcto funcionamiento de la aplicación. Asimismo, Thymeleaf facilitó la integración entre frontend y backend mediante la gestión de variables en el modelo, lo que permitió un intercambio dinámico de datos entre ambas capas.

1.4.1.2 CSS3

Se aplicó **CSS3** para mejorar la presentación visual de la aplicación mediante la implementación de animaciones en diversos componentes, lo que contribuyó a una experiencia de usuario más fluida e interactiva. Además, se personalizó la tipografía utilizando fuentes externas, integradas a través de Google Fonts y fuentes adicionales como *Sonic Extra Bold*.

1.4.1.3 Tailwind CSS

Se integró **Tailwind CSS** como framework de utilidades para agilizar el desarrollo del diseño visual. Gracias a su enfoque "utility-first", se logró aplicar estilos directamente desde el HTML de forma estructurada y coherente. Tailwind facilitó la implementación de diseños responsivos ya que es "mobile-first" por lo que la maquetación comienza desde la interfaz móvil.

1.4.1.4 JavaScript

El comportamiento dinámico del sitio se desarrolló utilizando **JavaScript** Vanilla. Se escribieron scripts personalizados para gestionar la interacción con la interfaz y enviar solicitudes al backend mediante funciones fetch.

1.4.1.4.1 Chart.js

Para la representación visual de datos estadísticos, se utilizó la librería **Chart.js**, que permitió generar gráficos radiales con una carga mínima en el frontend. Se configuraron opciones personalizadas para adaptar los gráficos al diseño general del proyecto y hacerlos comprensibles para el usuario.

1.4.1.4.2 SweetAlert2

Se implementó **SweetAlert 2** como herramienta de alertas modales, lo cual mejoró la experiencia de usuario en comparación con los diálogos nativos del navegador. Se usaron alertas para confirmar acciones, notificar errores o advertencias y proporcionar retroalimentación visual inmediata

1.4.1.5 Google Fonts

Se incluyó el servicio de **Google Fonts** para integrar una tipografía personalizada que mejora la identidad visual del proyecto. Se eligió una fuente legible, moderna y compatible con distintos navegadores y dispositivos.

1.4.1.6 Fuente externa: Sonic Extra Bold

Adicionalmente, se utilizó la fuente **Sonic Extra Bold** como recurso tipográfico distintivo para el logo. Esta elección contribuyó a reforzar la personalidad visual del proyecto.

1.4.2 Backend

Para el desarrollo del backend se eligieron tecnologías robustas, escalables y ampliamente utilizadas en entornos empresariales. Se priorizó la modularidad, la seguridad y la integración con servicios externos.

1.4.2.1 Java

Se desarrolló la lógica de servidor utilizando el lenguaje **Java**, debido a su tipado fuerte, su madurez en entornos empresariales y su compatibilidad con frameworks modernos como Spring Boot.

1.4.2.2 Spring Boot

Se empleó **Spring Boot** como framework principal para facilitar la configuración automática del entorno y el desarrollo del backend. Se estructuró el proyecto bajo el patrón **Modelo-Vista-Controlador (MVC)**, lo que permitió una separación clara de responsabilidades. Las dependencias se gestionaron mediante **Maven**, incorporando módulos clave como **Spring Web** para el manejo de rutas y controladores, **Spring Security** para la autenticación y autorización de usuarios, y **Spring JDBC** para la interacción directa con la base de datos, permitiendo mapear los resultados en objetos DTO de forma controlada.

1.4.2.3 OpenAI API

Se integró la **API de OpenAI** para generar respuestas automatizadas dentro del simulador de combates. Para ello, se diseñaron prompts específicos y se gestionaron las solicitudes desde el backend, controlando errores y validando la respuesta antes de enviarla al frontend. Esta integración añadió una capa de inteligencia artificial al proyecto.

1.4.2.4 Google Auth

Como extra a la autenticación de usuarios, se utilizó el sistema **OAuth 2.0 de Google**, permitiendo iniciar sesión con cuentas de Google de manera segura. Se configuró la integración con Spring Security para proteger rutas, gestionar sesiones y almacenar la información mínima necesaria del usuario, respetando las normativas de privacidad.

1.4.3 Base de Datos

La gestión y persistencia de los datos se abordó mediante el uso de dos bases de datos diferenciadas para entornos de desarrollo y producción. Se aplicaron relaciones entre tablas para estructurar de forma eficiente la información y guardarla de forma clara, ordenada y fácil de usar.

1.4.3.1 H2 (Desarrollo)

Durante el desarrollo, se utilizó la base de datos en memoria **H2**. Esta elección permitió una configuración rápida, pruebas inmediatas y una limpieza automática de datos en cada reinicio, lo que facilitó la iteración continua sin necesidad de herramientas externas.

1.4.3.2 PostgreSQL (Producción)

Para el entorno de producción, se implementó **PostgreSQL**, una base de datos relacional potente, segura y de código abierto. Se configuraron las credenciales, variables de entorno y conexión mediante Spring Boot, asegurando la persistencia real de los datos. Además, se diseñaron estrategias de migración para facilitar la transición desde H2. El motivo del uso de PostgreSQL fue de cara al hosting Heroku ya que tienen una alta compatibilidad.

1.4.4 Herramientas y Entornos

Durante el desarrollo del proyecto se utilizaron diversas herramientas y entornos que permitieron gestionar el ciclo de vida del software de forma eficiente, desde la programación y el control de versiones hasta el despliegue y la ejecución en diferentes entornos.

1.4.4.1 Git & GitHub

Se utilizó **Git** como sistema de control de versiones distribuido. Se logró mantener un historial limpio y trazable del desarrollo. Para alojar el repositorio, se empleó **GitHub**, lo que facilitó la colaboración remota entre diferentes dispositivos. Se implementó una guía informativa y de despliegue local del código mediante el README.md.

1.4.4.2 Maven

Se gestionaron las dependencias del proyecto con **Maven**, una herramienta de construcción ampliamente utilizada en el ecosistema Java. Se configuró el archivo `pom.xml` para declarar librerías necesarias para el funcionamiento del backend. Además, Maven permitió compilar, empaquetar de forma estandarizada.

1.4.4.3 IntelliJ IDEA

Como entorno de desarrollo integrado (IDE), se utilizó **IntelliJ IDEA**. Esta herramienta ofrece soporte avanzado para Java y Spring Boot, incluyendo autocompletado inteligente, refactorización asistida, integración con Maven, gestión de control de versiones y herramientas de inspección de código. Su uso optimizó el tiempo de desarrollo y redujo errores.

1.4.4.4 Heroku

Para el despliegue del entorno de producción, se configuró una aplicación en **Heroku**, una plataforma como servicio (PaaS) que facilitó el despliegue de la aplicación Spring Boot con PostgreSQL. Se gestionaron las variables de entorno desde el panel de Heroku y se automatizó el despliegue con GitHub, permitiendo una integración continua sencilla.

1.4.4.5 Servidor Personal (Desarrollo)

Durante las primeras fases del desarrollo, se ejecutó la aplicación en un **servidor local** para validar su comportamiento antes del despliegue. Se configuró la red local y el puerto de escucha (80). Se hizo una traducción de DNS con FreeDNS para no acceder directamente con la IP. Esta práctica facilitó la iteración constante antes de migrar a entornos más formales como Heroku.

2. Análisis de Requisitos

Para garantizar un desarrollo coherente con los objetivos del proyecto, se realizó un análisis detallado de los requisitos que debían cumplir tanto la interfaz como la lógica del sistema. Este análisis contempló aspectos funcionales que determinan lo que el sistema debe hacer y no funcionales que establecen cómo debe comportarse en términos de rendimiento, seguridad, usabilidad y escalabilidad.

Además, se consideraron las necesidades del usuario final a través del diseño de casos de uso representativos, la planificación de la experiencia de usuario (UX) y la definición de una arquitectura de pantallas clara y coherente. Todo ello permitió orientar el desarrollo hacia una aplicación funcional, accesible y bien estructurada.

2.1 Análisis Funcional y No Funcional

Para garantizar que el desarrollo de la aplicación cumpliera con las expectativas y necesidades del usuario final, se llevó a cabo un exhaustivo análisis de los requisitos funcionales y no funcionales.

El **análisis funcional** se centró en identificar y definir las acciones y operaciones que la aplicación debía ser capaz de realizar. Entre los principales requisitos funcionales se contemplaron funcionalidades como el registro y autenticación de usuarios, la gestión de perfiles, la visualización de rankings y estadísticas de peleadores, la simulación de combates y la integración con sistemas de autenticación externa como Google Auth. Además, se contempló la necesidad de incorporar funcionalidades interactivas que

mejoraran la experiencia del usuario, tales como alertas visuales y gráficos dinámicos generados con librerías como Chart.js y SweetAlert2. Además de una simulación de peleadores realista con inteligencia artificial.

Por otro lado, el **análisis no funcional** abordó aspectos relacionados con la calidad, el rendimiento y la seguridad de la aplicación. Se establecieron criterios de usabilidad para asegurar una interfaz intuitiva y accesible, así como la implementación de medidas de seguridad robustas mediante Spring Security y OAuth2 para proteger la información personal y garantizar la integridad de los datos. Asimismo, se definieron objetivos de escalabilidad y mantenibilidad que facilitarían futuras mejoras y adaptaciones del sistema, además de establecer tiempos de respuesta adecuados para ofrecer una experiencia fluida y eficiente.

Se llevó a cabo un análisis de los casos de uso para identificar y modelar las interacciones principales entre los usuarios y la aplicación. Este proceso permitió comprender cómo los usuarios finales utilizarían las distintas funcionalidades, facilitando el diseño de flujos de trabajo intuitivos y eficientes que respondieran a sus necesidades.

Los casos de uso identificados incluyeron, entre otros, el registro y autenticación de usuarios, la navegación por las diferentes vistas, la gestión de favoritos, la simulación de combates y la visualización de estadísticas personalizadas.

En paralelo, se trabajó en la experiencia de usuario (UX) para asegurar que la aplicación no solo cumpliera con los requisitos funcionales, sino que también ofreciera una interfaz amigable, accesible y atractiva. Se prestó especial atención a la usabilidad, facilidad de navegación, tiempos de carga y respuestas visuales que brindaran retroalimentación clara y coherente. Herramientas como SweetAlert 2 y Chart.js contribuyeron a mejorar la interacción y la presentación de datos, haciendo que la experiencia fuera más dinámica y envolvente.

Además, se consideraron aspectos de accesibilidad y adaptabilidad, garantizando que la aplicación fuera usable en diferentes dispositivos y tamaños de pantalla mediante el uso de Tailwind CSS para un diseño responsive.

Este enfoque centrado en los casos de uso y la experiencia de usuario fue clave para diseñar una aplicación funcional, eficiente y agradable que cumpliera con las expectativas del público objetivo.

Caso de uso completo versión escritorio y móvil (grabado):

- Entrar a la raíz del proyecto “home”.
 - Recargar para comprobar que cambia cada imagen y frase.
 - Hacer hover por el menú.
 - Hacer hover en botones y probarlos.
 - Comprobar enlaces de header y footer
- Entrar en “Privacy Policy”.
- Buscar URL no existente (logueado y sin loguear).
- Entrar a “Ranking”.
 - Desplegar la carta peleador.
 - Pulsar ver más y ver menos.
- Entrar en “Fighters”, “Simulator” y “User”.
 - Ver que te pide estar logueado.
 - Crear cuenta
 - Poner usuario que ya existe.
 - Poner email que ya existe.
 - Poner contraseña < 8 caracteres.
 - Poner contraseña sin mayúsculas.
 - Poner contraseña sin minúsculas.
 - Poner contraseña sin números.
 - Poner contraseña sin caracteres especiales.
 - Iniciar sesión
 - Poner contraseña mal.
 - Iniciar sesión correctamente.
- Entrar en “Fighters”
 - Comprobar cada filtro, la búsqueda y paginación.

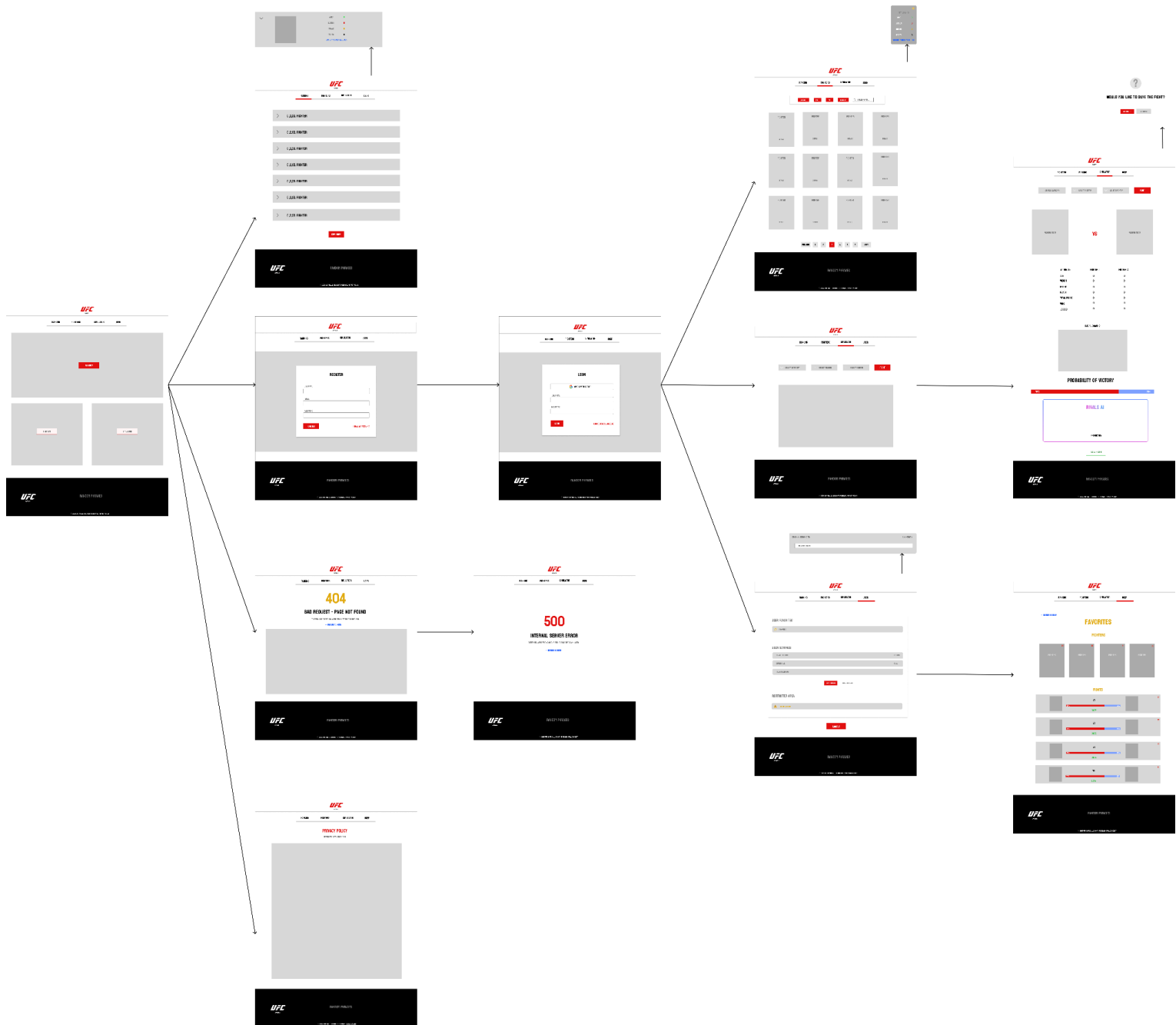
- Hacer hover sobre la carta.
- Añadir peleador a favoritos con animación y notificación y quitarlo.
- Entrar en “Simulator”
 - Seleccionar categoría con los inputs de peleadores “disabled”.
 - Seleccionar peleadores de la misma categoría con peleador 1 “disabled”.
 - Simular combate, ver estadísticas y respuesta de la IA.
 - Guardar pelea con alerta.
 - Comprobar botón de guardar disabled.
- Entrar en “User”.
 - Ver favoritos
 - Eliminar favoritos y probar enlace de añadir más.
 - Ajustes de usuario
 - Cambiar el usuario, email y contraseña (todos e individual).
 - Probar la validación de crear cuenta.
 - Probar a enviar vacío.
 - Poner contraseña al hacer cambios.
 - Ver alerta de error y confirmación.
 - Rellenar todos los campos y limpiar
 - Eliminar cuenta con alerta.

2.3 Mapa de Pantallas y Navegación

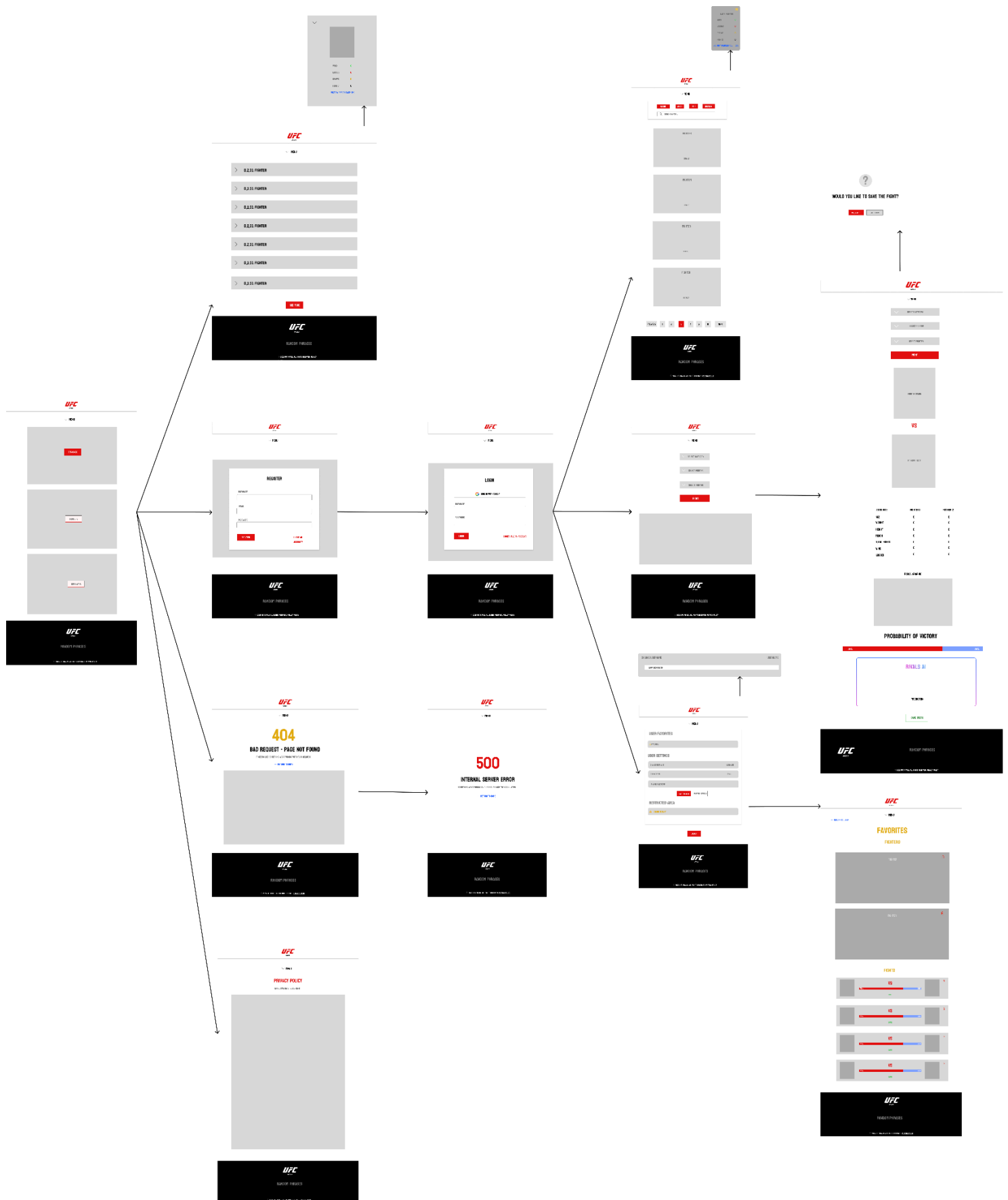
Se puede visualizar en el siguiente enlace:

<https://www.figma.com/design/ZfQr0KU8GpSmB5Fi68ID90/UFC-Rivals?node-id=165-2&t=1VHyo4bZtIdxf5d-0> se puede acceder tanto a la versión móvil como a la versión escritorio.

Esta es la versión escritorio:



Esta es la versión móvil:



2.4 Manual de Usuario (UX/UI)

Durante el desarrollo de la aplicación, se priorizó la experiencia del usuario (UX) como un pilar esencial para garantizar la accesibilidad, claridad y fluidez en la navegación. El diseño de la interfaz de usuario (UI) fue concebido con un enfoque minimalista e intuitivo, orientado a ofrecer una interacción ágil y coherente en todas las pantallas del sistema.

Se empleó una paleta cromática pensada para combinar impacto visual y legibilidad. Los colores principales fueron **rojo (#E50F0F)**, **blanco** y **negro**, los cuales establecen un contraste fuerte y dinámico que remite al mundo competitivo y enérgico de las artes marciales mixtas. El rojo fue utilizado como color base para resaltar elementos clave como botones de acción, títulos o indicadores importantes. El blanco y negro sirvieron como equilibrio para proporcionar claridad en los fondos y los textos, facilitando la lectura y la navegación.

Se utilizaron iconos proporcionados por Icons8, seleccionados por su coherencia visual con el estilo general, estos contribuyeron a una interfaz limpia y moderna.

Como colores secundarios se incorporaron:

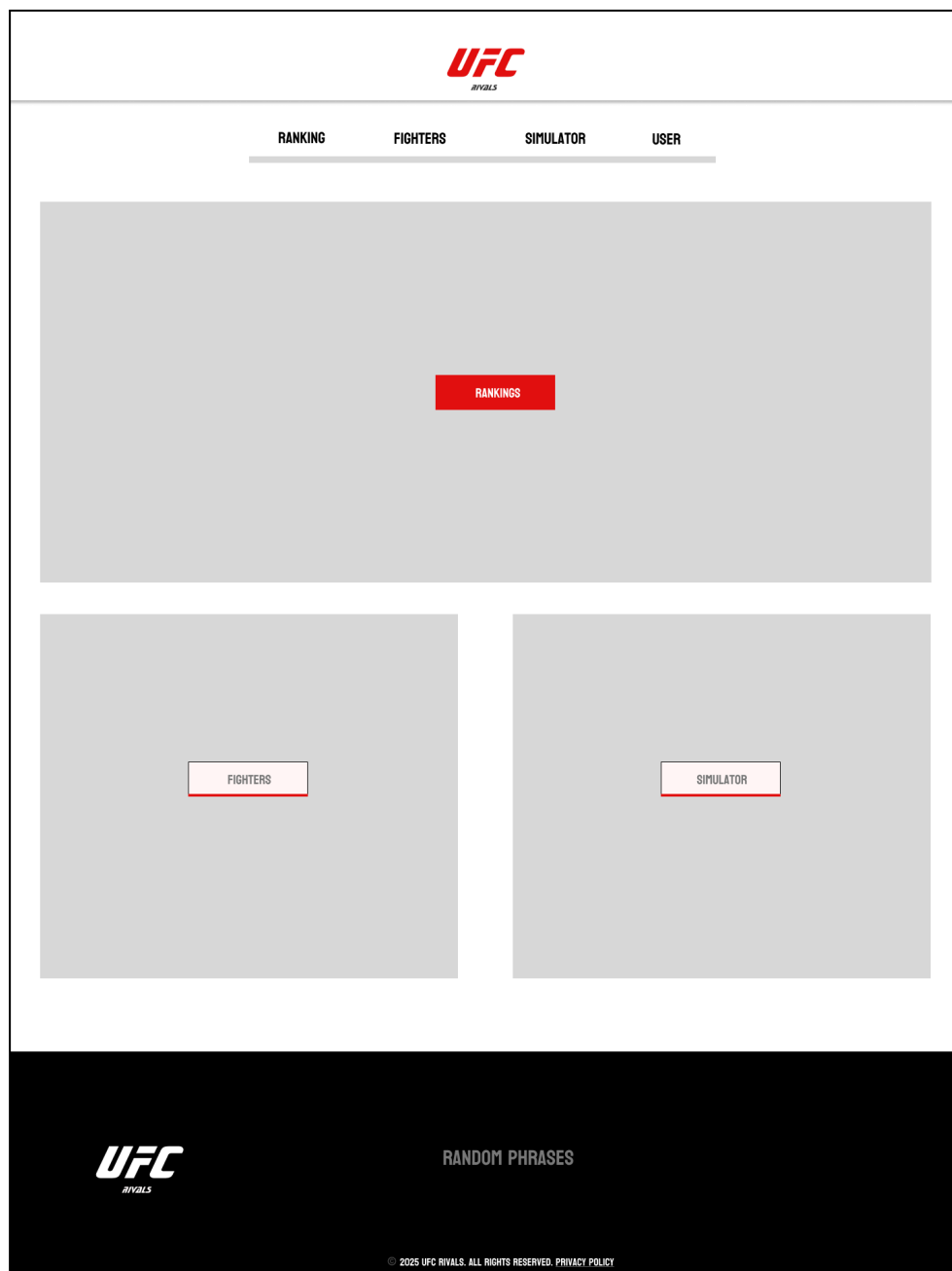
- **Amarillo (#E5AC0F)**, usado para destacar información o llamados a la atención (call to action).
- **Gris claro (#D9D9D9)**, aplicado en elementos neutros como fondos secundarios o separadores.
- **Azul (#2F6AFF)** y **verde (#3AC54A)**, utilizados de manera puntual para representar acciones positivas (confirmaciones, éxito) o información complementaria.

Cada vista fue diseñada para mantener una coherencia visual, con componentes reutilizados como el encabezado, pie de página y menús, garantizando así una experiencia homogénea en toda la aplicación. Se incorporaron principios de diseño responsivo para asegurar la correcta visualización tanto en dispositivos de escritorio como móviles.

2.4.1 Landing Page

La **Landing Page** actúa como la página de inicio de la aplicación y comparte estructura con el resto de vistas a través de un **header** y un **footer** comunes. El header incluye un menú con efecto *hover* dinámico, mientras que el footer muestra frases aleatorias en cada visita y contiene un enlace a la **Política de Privacidad**.

La página tiene un diseño visual atractivo y dinámico: las **imágenes principales** cambian aleatoriamente en cada carga y los **botones** presentan animaciones interactivas al pasar el cursor. Esta combinación de elementos ofrece una experiencia de usuario fluida, moderna y coherente con la identidad visual de la plataforma.



2.4.2 Registro e Inicio de Sesión

El sistema principal de autenticación de la aplicación se basa en un formulario propio de registro e inicio de sesión, diseñado con validaciones estrictas para garantizar la seguridad de los datos introducidos. Durante el registro, se verifica que el nombre de usuario y el correo electrónico no estén duplicados, y que la contraseña cumpla con requisitos mínimos de seguridad: al menos ocho caracteres, inclusión de mayúsculas, minúsculas, números y símbolos especiales.

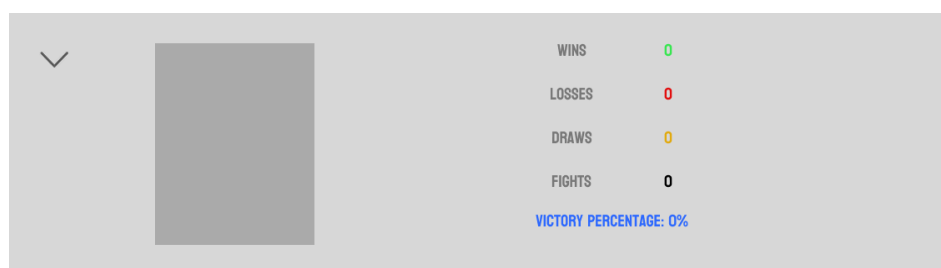
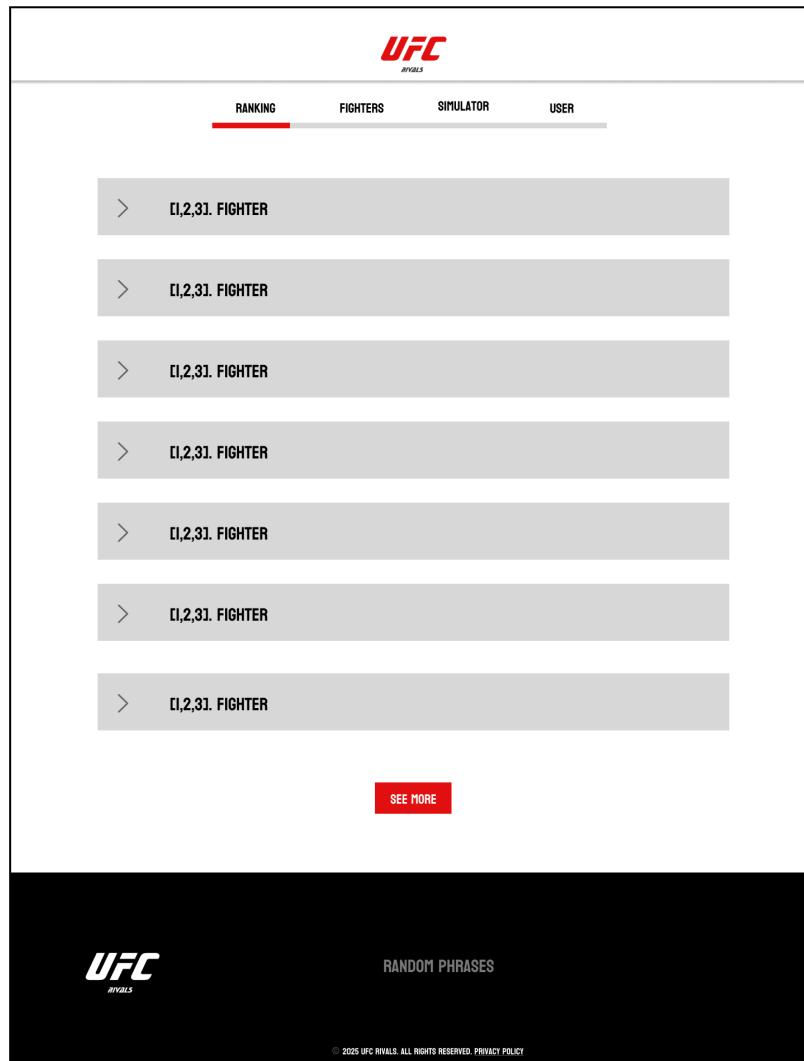
Una vez registrado, el usuario puede iniciar sesión mediante este mismo sistema. La interfaz se mantuvo clara, accesible y centrada en la acción, eliminando elementos innecesarios para facilitar el uso desde cualquier dispositivo.

Como funcionalidad adicional, se integró el inicio de sesión mediante cuentas de Google utilizando OAuth2, lo que ofrece una vía alternativa rápida y segura, pero sin sustituir el sistema propio desarrollado.

The screenshot displays the login interface for the UFC Rivals application. At the top, the UFC Rivals logo is centered. Below it, a navigation bar contains links for RANKING, FIGHTERS, SIMULATOR, and USER. The main content area is a light gray rectangle containing a white login form. The form is titled 'LOGIN' and features a 'SIGN IN WITH GOOGLE' button with the Google logo. Below this are input fields for 'USERNAME' and 'PASSWORD'. At the bottom of the form, there is a red 'LOGIN' button and a link that says 'I DON'T HAVE AN ACCOUNT'. The footer is a black bar with the UFC Rivals logo on the left, 'RANDOM PHRASES' in the center, and a small copyright notice '© 2025 UFC RIVALS. ALL RIGHTS RESERVED. PRIVACY POLICY' on the right.

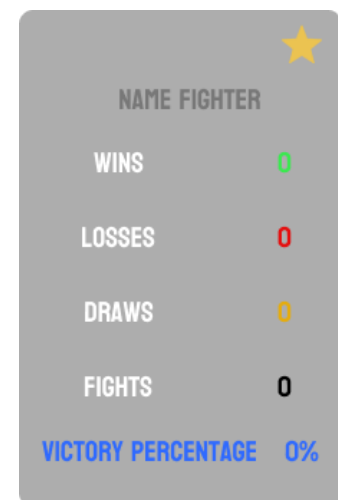
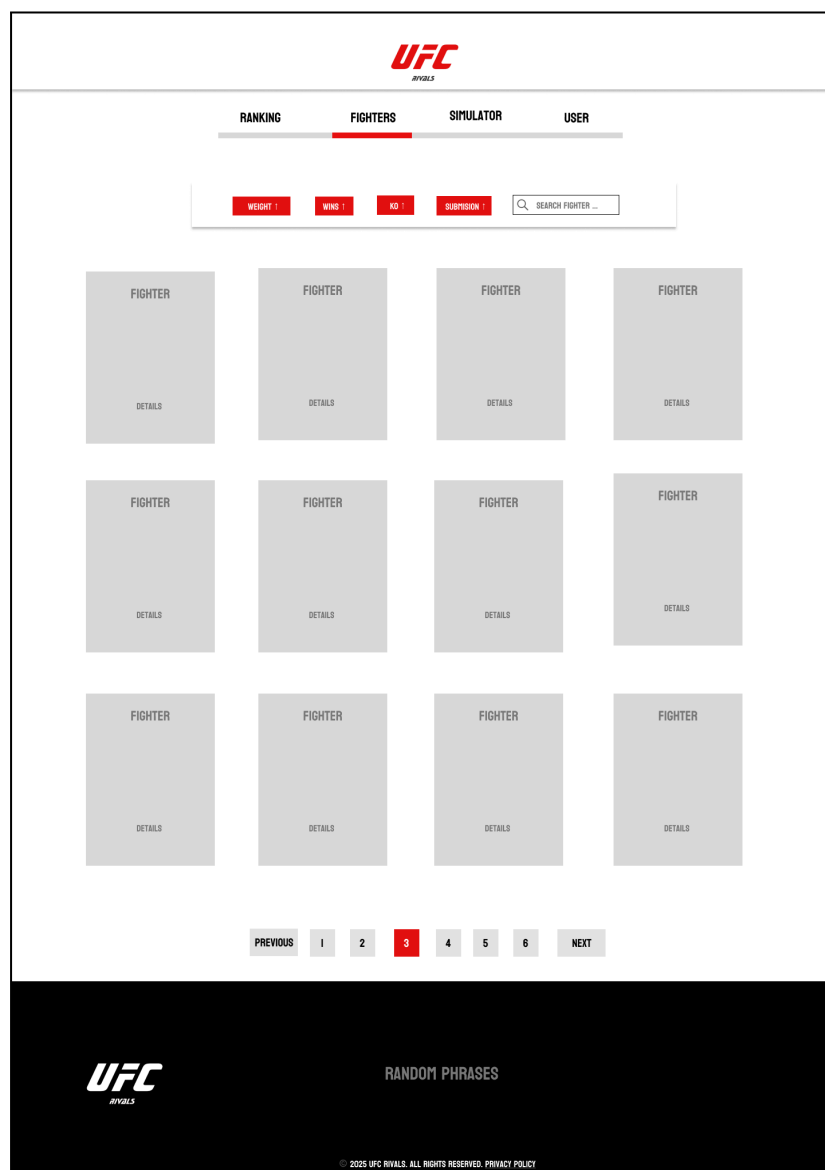
2.4.3 Ranking

En esta sección se muestra una clasificación dinámica de los 15 peleadores mejor posicionados. Inicialmente, se presentan los 9 primeros, con la opción de *ver más* para desplegar los restantes y *ver menos* para contraer la vista. Al seleccionar un peleador, se despliega una tarjeta con sus estadísticas detalladas: número de victorias, derrotas, empates, total de combates y porcentaje de victorias. Esta funcionalidad mejora la experiencia de usuario al ofrecer información completa de forma accesible y ordenada.



2.4.4 Fighters

Esta sección presenta el listado completo de peleadores, permitiendo la aplicación de filtros por peso, victorias, nocauts y sumisiones, así como la búsqueda por palabra clave. Los peleadores se muestran de forma paginada, con una disposición de cuatro por fila en escritorio y uno por fila en dispositivos móviles. Al pasar el cursor sobre la tarjeta de un peleador, se ejecuta una animación que realiza un flip de la carta; en el anverso se visualiza una fotografía junto a su nombre, nickname y país, mientras que en el reverso se despliegan las estadísticas detalladas: victorias, derrotas, empates, total de combates y porcentaje de victorias. Además, en el reverso se incorpora un icono de favoritos que, al pulsarse, añade el peleador a la lista de favoritos del usuario y resalta la tarjeta con un borde dorado además de una notificación dinámica.



2.4.5 Simulador

El simulador permite comparar dos peleadores dentro de una misma categoría de peso. Para comenzar, se deben seleccionar tres elementos mediante listas desplegables: primero la categoría (requisito obligatorio), lo que habilita los otros dos selectores, que muestran únicamente peleadores pertenecientes a dicha categoría. Una vez seleccionado el primer peleador, este queda deshabilitado en el segundo selector para evitar duplicados. Mientras se realiza la selección, se muestra una imagen animada para acompañar la experiencia visual.

Al pulsar el botón "Luchar", se genera una tabla comparativa con atributos clave como peso, altura, alcance, victorias y derrotas. A continuación, se despliega un gráfico radial dinámico que compara el desempeño de ambos peleadores en victorias y derrotas por nocaut, decisión, sumisión y empates. Además, se presenta una barra de estimación de victoria con porcentajes orientativos.

Finalmente, se incluye un apartado de inteligencia artificial que genera una simulación escrita del combate, y un botón de "Guardar", que al pulsarlo activa una alerta dinámica confirmando la acción.

The image shows a web application interface for 'UFC Rivals'. At the top, the 'UFC RIVALS' logo is centered. Below it is a navigation bar with four tabs: 'RANKING', 'FIGHTERS', 'SIMULATOR' (which is highlighted with a red underline), and 'USER'. The main content area contains three dropdown menus labeled 'SELECT CATEGORY', 'SELECT FIGHTER', and 'SELECT FIGHTER', followed by a red button labeled 'FIGHT'. Below these controls is a large, empty gray rectangular box, likely a placeholder for a fight simulation or results. The footer is black and contains the 'UFC RIVALS' logo on the left, the text 'RANDOM PHRASES' in the center, and a small copyright notice '© 2025 UFC RIVALS. ALL RIGHTS RESERVED. PRIVACY POLICY' on the right.

FIGHTERS

RANKING

SIMULATOR

USER

SELECT CATEGORY

SELECT FIGHTER

SELECT FIGHTER

FIGHT

FIGHTER IMAGE

VS

FIGHTER IMAGE

ATTRIBUTE	FIGHTER 1	FIGHTER 2
AGE	0	0
WEIGHT	0	0
HEIGHT	0	0
REACH	0	0
TOTAL FIGHTS	0	0
WINS	0	0
LOSSES	0	0

RADIAL GRAPHIC

PROBABILITY OF VICTORY

75%

25%

RIVALS AI

PREDICTION

SAVE FIGHTH

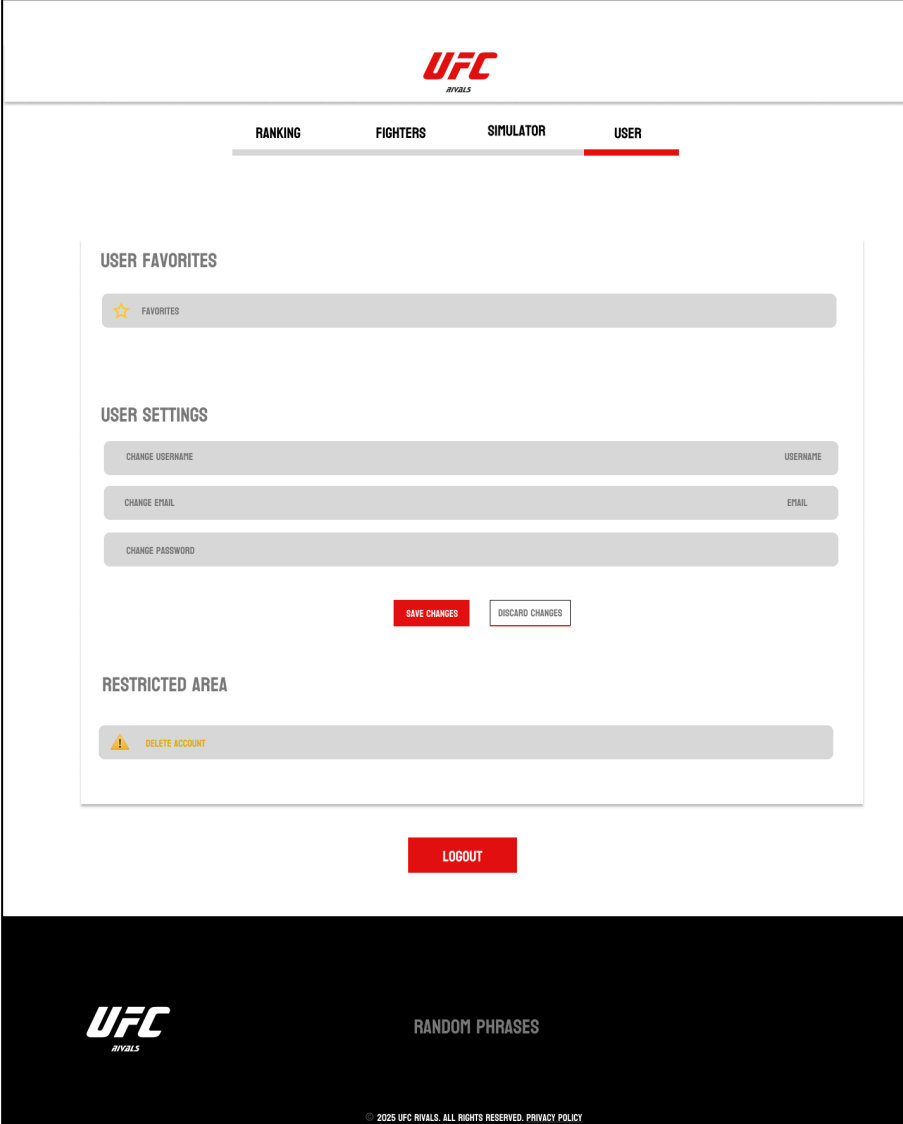
RANDOM PHRASES

© 2025 UFC RIVALS. ALL RIGHTS RESERVED. PRIVACY POLICY

2.4.6 Página de Usuario

La página de usuario centraliza las opciones de gestión personal. Incluye un apartado de ajustes donde es posible modificar el nombre de usuario, el correo electrónico o la contraseña. Cada cambio requiere la introducción de la contraseña actual como medida de seguridad, tras lo cual se cierra la sesión automáticamente, obligando a un nuevo inicio. Todas las acciones están acompañadas de alertas dinámicas que confirman o advierten sobre los resultados. Al hacer cualquier cambio se mantiene la validación de registrar usuario.

También se ofrece la posibilidad de eliminar la cuenta, lo cual igualmente exige la contraseña para proceder. Esta acción elimina los datos asociados en cascada, garantizando una gestión responsable de la información. Finalmente, se proporciona un botón de cierre de sesión (logout) para terminar la sesión manualmente.



The screenshot displays the 'USER' management interface of the UFC Rivals application. At the top, the 'UFC RIVALS' logo is centered, with a navigation bar below it containing links for 'RANKING', 'FIGHTERS', 'SIMULATOR', and 'USER' (the active page, highlighted with a red underline). The main content area is divided into three sections: 'USER FAVORITES' with a 'FAVORITES' button; 'USER SETTINGS' with buttons for 'CHANGE USERNAME', 'CHANGE EMAIL', and 'CHANGE PASSWORD', each followed by a corresponding input field; and 'RESTRICTED AREA' with a 'DELETE ACCOUNT' button. Below these sections are two buttons: 'SAVE CHANGES' (red) and 'DISCARD CHANGES' (white with a red border). At the bottom of the main content area is a red 'LOGOUT' button. The footer is a dark grey bar containing the 'UFC RIVALS' logo, the text 'RANDOM PHRASES', and a copyright notice: '© 2025 UFC RIVALS. ALL RIGHTS RESERVED. PRIVACY POLICY'.



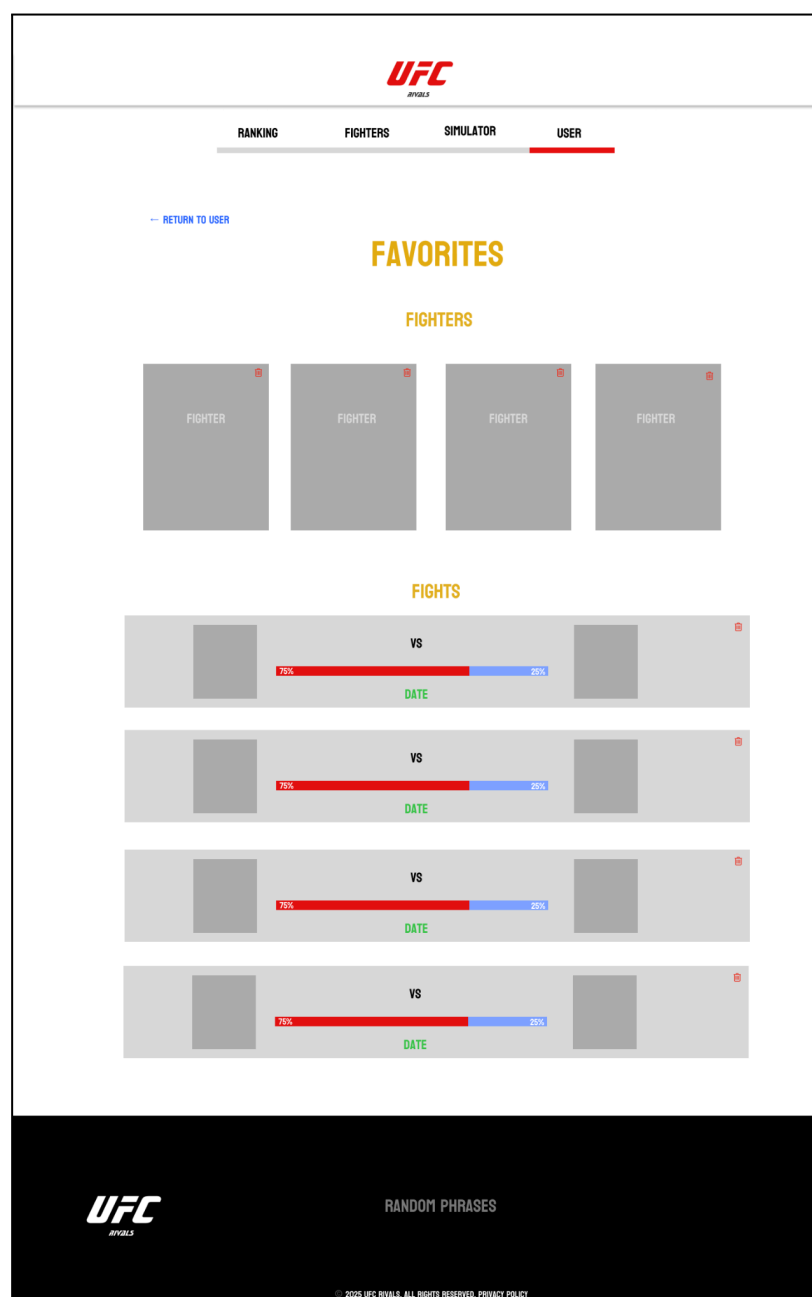
This is a close-up of the 'CHANGE USERNAME' form. It features a label 'CHANGE USERNAME' on the left and a 'USERNAME' label on the right. Below these is a text input field with the placeholder text 'NEW USERNAME'.

2.4.7 Gestión de Favoritos

En este apartado, el usuario puede visualizar y administrar sus elementos guardados. Se dividen en dos secciones: peleadores favoritos y simulaciones guardadas.

En la sección de peleadores favoritos, se muestran tarjetas con el nombre e imagen de cada peleador. Al pasar el cursor (hover) sobre una tarjeta, aparece un icono de papelera que permite eliminarlo de la lista.

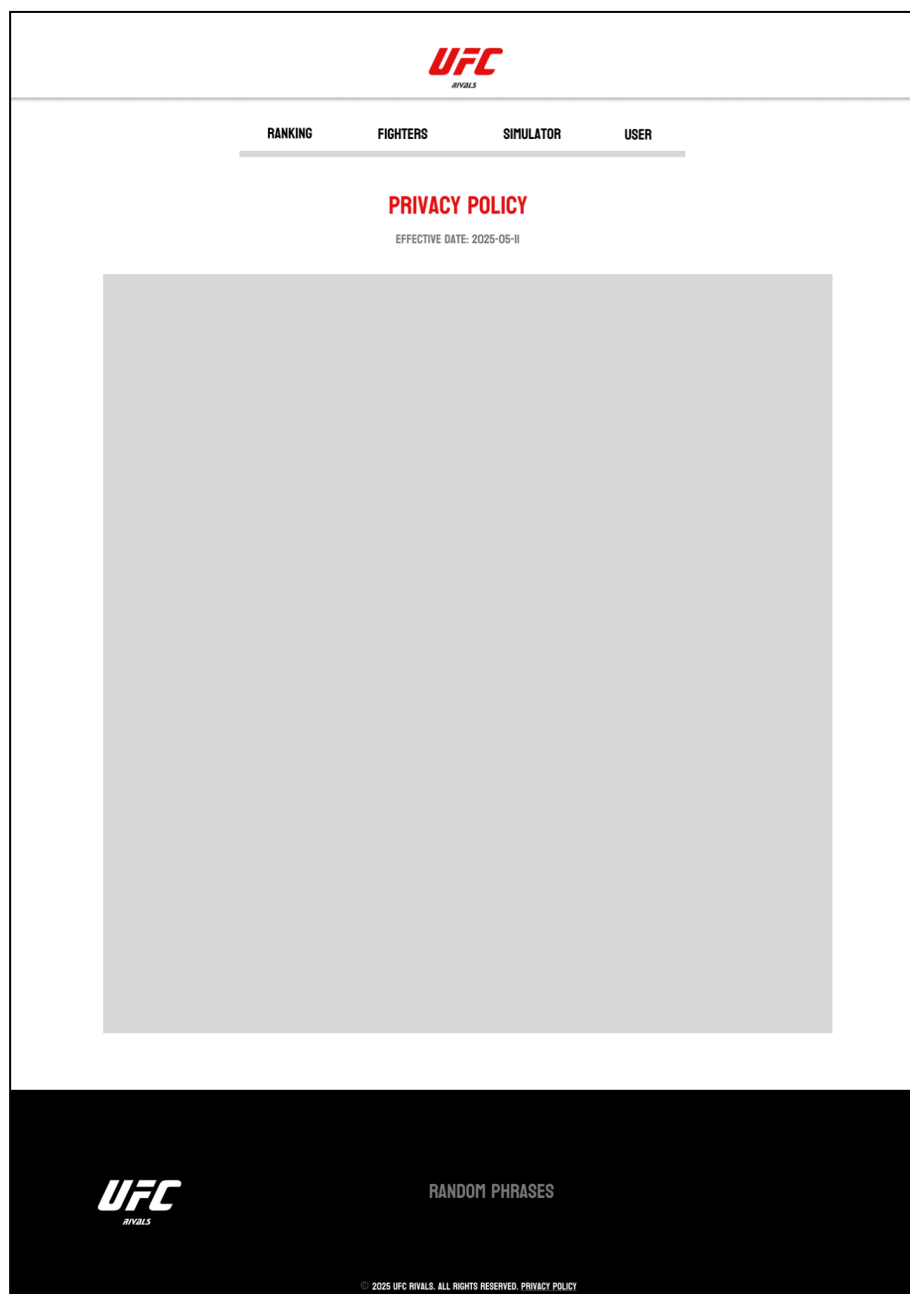
Las simulaciones guardadas muestran el enfrentamiento entre dos peleadores, incluyendo sus nombres, imágenes, una barra de porcentaje que representa la estimación de victoria y la fecha en que se realizó la simulación. Al igual que en los peleadores, es posible eliminarlas mediante el icono de papelera al hacer hover. Esta funcionalidad proporciona al usuario un control claro e intuitivo sobre sus elementos destacados.



2.4.8 Política de Privacidad y Seguridad

La aplicación cuenta con una política de privacidad accesible desde el footer, que informa sobre el uso de datos personales. Solo se recogen los datos necesarios para el registro o inicio de sesión: usuario, email y contraseña cifrada. Si se usa Google, la autenticación se gestiona mediante OAuth de forma segura, sin almacenar la contraseña ni acceder a datos sensibles.

No se comparten datos con terceros ni se utilizan cookies de seguimiento, salvo una cookie de sesión para la autenticación. Se aplican medidas de seguridad técnicas para proteger la información del usuario, y este puede acceder, modificar o eliminar sus datos en cualquier momento.

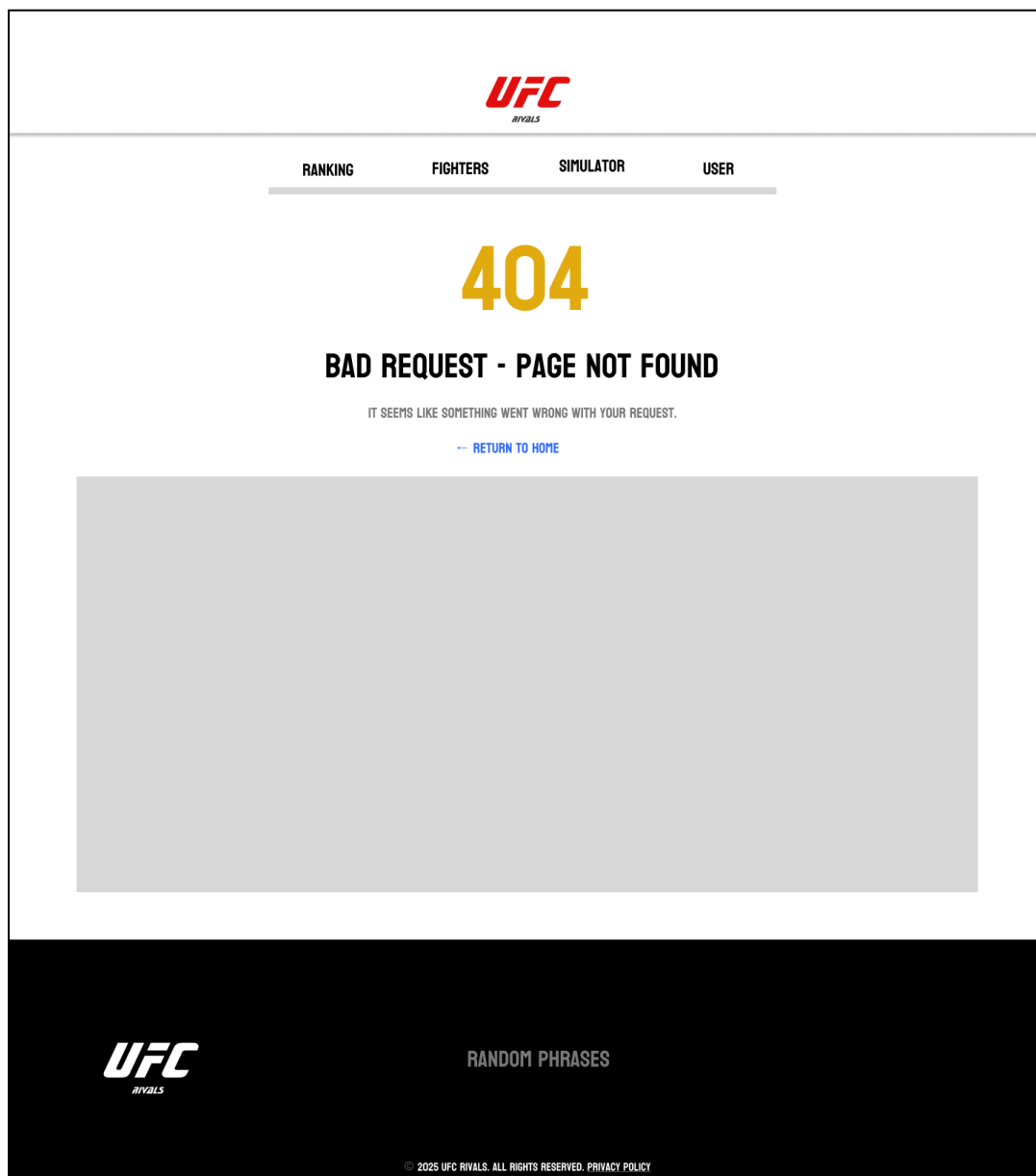


2.4.9 Páginas de Error

La aplicación dispone de páginas personalizadas para manejar errores comunes y mejorar la experiencia del usuario.

- **Error 404 – Página no encontrada:** se muestra cuando el usuario intenta acceder a una ruta inexistente. Cuenta con una imagen animada que transmite el error de forma visual y amigable.
- **Error 500 – Error interno del servidor:** aparece ante fallos inesperados del servidor. Presenta un diseño coherente con el resto del sitio, manteniendo el estilo y la claridad del mensaje.

Ambas páginas incluyen opciones para regresar a la página principal, facilitando la navegación y reduciendo la frustración del usuario.





RANKING

FIGHTERS

SIMULATOR

USER

500

INTERNAL SERVER ERROR

SOMETHING WENT WRONG ON OUR END. PLEASE TRY AGAIN LATER.

[← RETURN TO HOME](#)



RANDOM PHRASES

© 2025 UFC RIVALS. ALL RIGHTS RESERVED. [PRIVACY POLICY](#)

3. Configuración del Entorno de Desarrollo

Durante la etapa inicial del proyecto, se configuró el entorno de desarrollo para asegurar una base sólida que facilitara el trabajo continuo. Se establecieron herramientas, flujos de trabajo y una estructura organizada que permitiera mantener buenas prácticas y escalabilidad a lo largo del desarrollo.

3.1 Creación del Repositorio y Estructura de Carpetas

Se creó un repositorio en GitHub como control de versiones y respaldo del proyecto. La estructura de carpetas se organizó siguiendo el patrón de diseño MVC (Modelo-Vista-Controlador), separando claramente los componentes del backend (controladores, servicios, modelos y repositorios) y del frontend (recursos estáticos, vistas y plantillas Thymeleaf). Esta organización permitió mantener el código limpio y modular.

3.2 Git: Flujo de Trabajo y Comandos

3.2.1 Git Init / Clone / Push / Pull / Restore

Se emplearon los comandos principales de Git para gestionar versiones:

- `git init` para inicializar el repositorio local.
- `git clone` para clonar el repositorio remoto en nuevas máquinas.
- `git push` y `git pull` para sincronizar cambios entre local y remoto.
- `git restore` para descartar cambios no deseados.

Este flujo aseguró una correcta sincronización entre las distintas etapas del desarrollo.

3.2.2 Buenas Prácticas de Commits

Se adoptaron buenas prácticas en los mensajes de commit, empleando descripciones claras, en tiempo presente, y agrupando cambios lógicos. Se evitó subir archivos temporales o innecesarios (como `.idea/`, `target/`, o archivos del sistema). También se utilizó `.gitignore` para excluir estos elementos automáticamente como un `application.properties` con secretos de las API Keys hasta que se metieron en variables de entorno.

Commits on May 14, 2025	
Página de error 500 lucaschacon3 committed 3 weeks ago	ba80eb9
Pequeños errores de maquetacion y configuracion de usuario lucaschacon3 committed 3 weeks ago	09ffa7b
mejora de responsividad y añadir confirmar contraseña para cambios en la cuenta del usuario lucaschacon3 committed 3 weeks ago	af6ed71
Commits on May 13, 2025	
Añadir diseño para la IA, y mejora del diseño lucaschacon3 committed 3 weeks ago	cc62e78
Añadir ajustes de usuario lucaschacon3 committed 3 weeks ago	015b65f
Commits on May 12, 2025	
Añadir política de privacidad, y mejorar diseño lucaschacon3 committed 3 weeks ago	642bb38
Solución de errores de diseño e implantación del read.me lucaschacon3 committed 3 weeks ago	ee81bee
Commits on May 8, 2025	
Desarrollo de las funcionalidades de usuario lucaschacon3 committed last month	eb2a26c
Acabar con la gestión de favoritos y mejorar el diseño general de la aplicación lucaschacon3 committed last month	b0b9ac5
Commits on May 7, 2025	
añadir la funcionalidad de peleas y peleadores favoritos lucaschacon3 committed last month	3c23956
Commits on May 6, 2025	
Añadir peleas favoritas lucaschacon3 committed last month	cde982a
Solucion de errores de maquetación y añadir peleas favoritas lucaschacon3 committed last month	783e4df
Cambio de estilos y mejora en la maquetación lucaschacon3 committed last month	6e38d49
Commits on May 5, 2025	
pasar de JPA a JDBC lucaschacon3 committed last month	c8aa986
solucion de pequeños detalles de diseño lucaschacon3 committed last month	a70e521
Commits on Apr 30, 2025	

3.3 Instalación de Herramientas

3.3.1 Java (Windows y Linux)

Se utilizó **Java 21** como versión base para el desarrollo del proyecto, aprovechando las mejoras en rendimiento, seguridad y nuevas características del lenguaje.

- En **Windows**, se descargó el JDK desde el sitio oficial de Oracle, configurando las variables de entorno (`JAVA_HOME`) y añadiendo Java al `PATH`.
- En **Linux**, se instaló mediante el gestor de paquetes “apt” (`sudo apt install openjdk-21-jdk`).

Se comprobó la instalación ejecutando en terminal: `java -version`

Esto aseguró que el entorno estaba preparado para ejecutar aplicaciones Spring Boot con compatibilidad total.

3.3.2 Maven (Windows y Linux)

Maven es el gestor de dependencias y sistema de construcción utilizado.

- En **Windows**, se descargó del sitio oficial de Apache Maven, se descomprimió y se añadió al `PATH` del sistema y a las variables de entorno (`MAVEN_HOME`).
- En **Linux**, se instaló vía gestor de paquetes (`sudo apt install maven`).

Se verificó su correcta instalación con: `mvn -v`

Maven permitió definir todas las dependencias del proyecto desde un único archivo (`pom.xml`), facilitando su instalación, actualización y gestión.

3.3.3 IntelliJ IDEA

Se utilizó **IntelliJ IDEA**, ya que es un entorno de desarrollo optimizado para proyectos Java y Spring Boot.

- Se creó el proyecto como aplicación Maven, lo que permitió la detección automática del `pom.xml` y la descarga de todas las dependencias.
- Se habilitaron complementos como Spring Boot, Lombok y soporte para Thymeleaf.
- Se configuraron atajos de teclado, inspección de código y herramientas de refactorización para agilizar el desarrollo y mantener buenas prácticas.

3.3.4 Navegadores y Testing

La interfaz fue probada en varios navegadores modernos como **Google Chrome**, **Mozilla Firefox** y **Safari** para asegurar una correcta visualización y compatibilidad cross-browser.

- Se utilizaron herramientas como **DevTools** para inspeccionar elementos, ajustar estilos CSS en tiempo real y comprobar el comportamiento responsive en diferentes resoluciones.
- Estas pruebas garantizaron una experiencia fluida tanto en escritorio como en dispositivos móviles.

3.4 Gestión de Dependencias con Maven

La totalidad de dependencias del proyecto se gestionó a través del archivo `pom.xml`. Algunas de las más relevantes fueron:

- **Spring Boot Starter Web** (para el backend y el enrutamiento)
- **Spring Security** (autenticación y protección de rutas)
- **Thymeleaf** (motor de plantillas HTML)
- **Spring Boot Starter JDBC** (conexión a base de datos)
- **Google OAuth2 Client** (inicio de sesión con Google)
- **H2 Database** (base de datos en memoria para desarrollo)
- **Postgre** (compatibilidad con la base de datos de producción)
- **WebClient** (para consumir la API de OpenAI)
- **Lombok** (reducción de código repetitivo en Java)
- **Spring Boot DevTools** (reinicio automático en desarrollo)
- **Spring Boot Starter Test** (tests unitarios y de integración)

Esto permitió automatizar la instalación, asegurar la compatibilidad de versiones y facilitar la replicación del entorno por parte de otros desarrolladores.

3.5 CDNs

Para optimizar el rendimiento y reducir el peso de las vistas cargadas desde el servidor durante el desarrollo, se incorporaron varias bibliotecas frontend mediante **CDNs**:

- **Tailwind CSS**: framework de estilos utilitario que permitió construir interfaces modernas, responsive y consistentes sin necesidad de escribir CSS desde cero.
- **SweetAlert 2**: alertas personalizadas, usadas en acciones como guardar, eliminar o errores.
- **Chart.js**: generación de gráficas dinámicas en el simulador de combates.
- **Google Fonts**: para aplicar tipografías modernas.
- **Icons8**: iconos coherentes con el estilo visual de la app.

Al utilizar CDNs, se logró una carga más rápida de los recursos, menor carga en el servidor y mayor disponibilidad. A la hora del despliegue se descargaron las librerías y se introdujeron en la carpeta de archivos estáticos ya que para producción no se debe depender de ficheros ubicados en lugares fuera de tu control.

4. Diseño y Desarrollo de la Base de Datos

4.1 Estructura de la Base de Datos

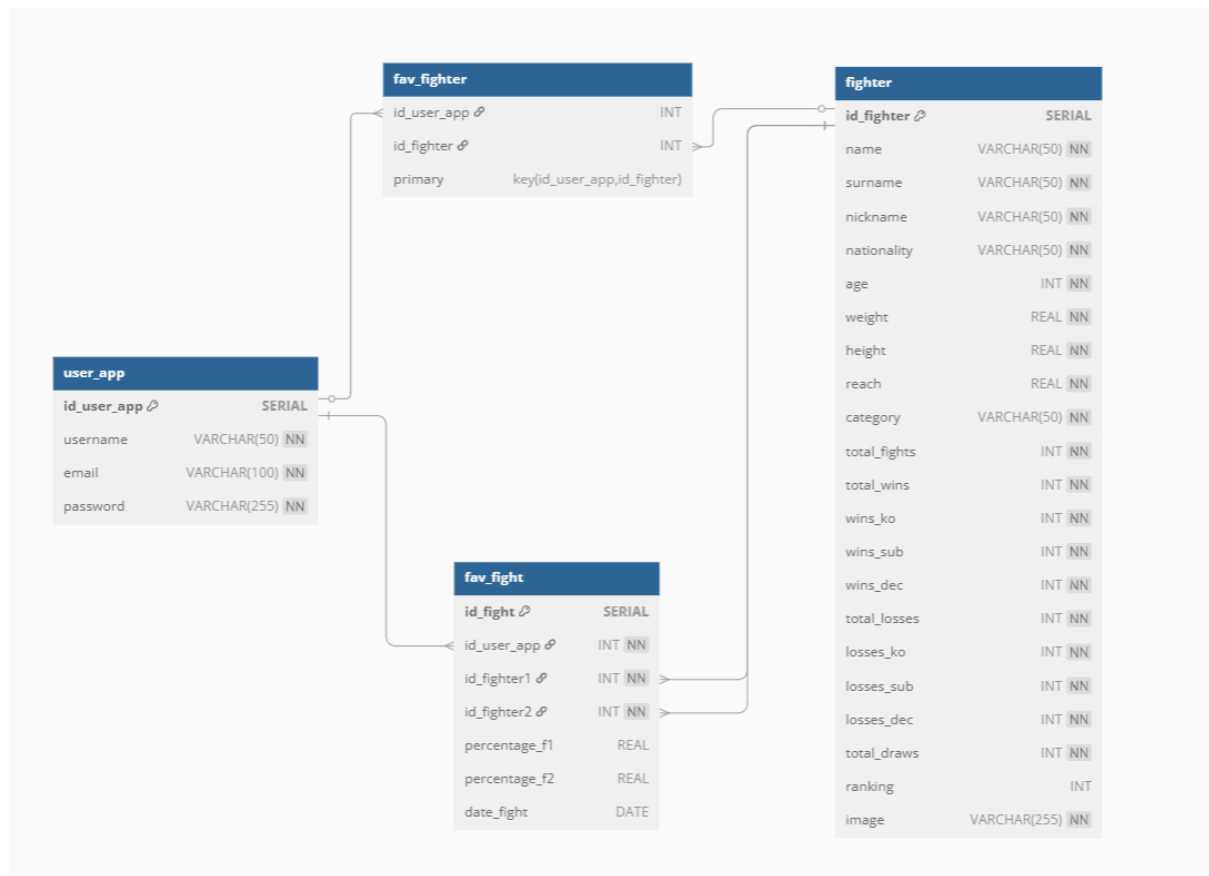
La base de datos gestiona la información de usuarios, peleadores y sus interacciones, incluyendo la selección de peleadores y combates favoritos por parte de cada usuario.

4.1.1 Diagrama

El modelo entidad-relación consta de cuatro tablas principales: **user_app**, **fighter**, **fav_fighter** y **fav_fight**.

- **user_app** almacena los datos de los usuarios registrados.
- **fighter** contiene la información detallada de cada peleador.
- **fav_fighter** vincula usuarios con sus peleadores favoritos, creando una relación muchos a muchos (N-M).
- **fav_fight** guarda peleas simuladas favoritas de los usuarios, incluyendo porcentajes de victoria y fechas.

Las relaciones entre tablas se establecen mediante claves foráneas para mantener la coherencia y permitir cascada de borrados, evitando registros huérfanos.



4.1.2 Tablas y Campos

4.1.2.1 fighter

Contiene la información detallada de cada peleador, incluyendo datos personales, estadísticas de combate y ranking.

- `id_fighter` (PK): identificador único autoincremental.
- `name`, `surname`, `nickname`: identificación y apodos.
- `nationality`, `age`, `weight`, `height`, `reach`, `category`: datos físicos y categoría.
- `total_fights`, `total_wins`, `wins_ko`, `wins_sub`, `wins_dec`: estadísticas de victorias y tipo.
- `total_losses`, `losses_ko`, `losses_sub`, `losses_dec`: estadísticas de derrotas y tipo.
- `total_draws`: empates.
- `ranking`: posición en el ranking (puede ser nulo para no ranking).
- `image`: URL de la imagen del peleador.

4.1.2.2 user_app

Almacena la información de los usuarios registrados.

- `id_user_app` (PK): identificador único autoincremental.
- `username`: nombre de usuario único.
- `email`: correo electrónico único.
- `password`: contraseña segura almacenada en formato hash.

4.1.2.3 fav_fighter

Tabla intermedia que relaciona usuarios con sus peleadores favoritos.

- `id_user_app` (FK): referencia a `user_app`.
- `id_fighter` (FK): referencia a `fighter`.
- Clave primaria compuesta por ambos campos para evitar duplicados.

4.1.2.4 fav_fight

Registra peleas simuladas favoritas de usuarios, guardando estadísticas de porcentajes de victoria y fecha de simulación.

- `id_fight` (PK): identificador único autoincremental.
- `id_user_app` (FK): usuario dueño de la simulación.
- `id_fighter1` y `id_fighter2` (FK): peleadores participantes.
- `percentage_f1`, `percentage_f2`: porcentajes estimados de victoria.
- `date_fight`: fecha en que se guardó la simulación.

4.1.3 Claves Primarias, Externas y Relaciones

Las claves primarias garantizan unicidad en cada tabla. Las claves foráneas (`id_user_app`, `id_fighter`) aseguran la integridad referencial y establecen las relaciones entre tablas. Además, la opción `ON DELETE CASCADE` elimina automáticamente los registros relacionados para mantener la coherencia cuando se borra un usuario o peleador.

4.2 Estrategia de Transición: H2 → PostgreSQL

Durante el desarrollo inicial, se utilizó H2 como base de datos en memoria por su rapidez y facilidad de configuración. Para producción y pruebas finales se migró a PostgreSQL, garantizando escalabilidad, seguridad y robustez. Se mantuvo compatibilidad en el esquema para facilitar la transición y minimizar modificaciones en el código.

4.3 Seguridad e Integridad de Datos

Se aplicaron varias medidas para proteger la integridad y confidencialidad de los datos:

- Contraseñas almacenadas con hashing seguro para evitar accesos no autorizados.
- Restricciones `UNIQUE` para evitar duplicados en campos clave como username y email.
- Relaciones con `FOREIGN KEY` para mantener la consistencia entre tablas.
- Validaciones en la capa de aplicación para evitar datos inconsistentes o inválidos.

4.4 Testing de Persistencia y Validaciones

Se realizaron pruebas para garantizar la correcta persistencia de datos y el cumplimiento de las restricciones:

- Inserción y eliminación de usuarios y peleadores para validar cascada y restricciones.
- Validación de la lógica de favoritos y peleas simuladas para prevenir inconsistencias.
- Pruebas de integración en Spring Boot usando la base de datos H2 en memoria y PostgreSQL real para asegurar el correcto funcionamiento en ambos entornos.

5. Arquitectura del Proyecto y Desarrollo

5.1 Estructura del Proyecto Spring Boot

El proyecto está construido con **Spring Boot**, adoptando una estructura modular y limpia, que permite una separación clara de responsabilidades. A continuación, se detalla cada uno de los paquetes principales que componen la aplicación:

- **controllers**: Contiene los controladores responsables de gestionar las peticiones HTTP y dirigirlos a los servicios correspondientes. Cada controlador se vincula a una funcionalidad específica del sistema, como usuarios (`UserController`), peleadores (`FighterController`), favoritos (`FavoriteController`), simulaciones (`SimulatorController`) o ranking (`RankingController`). También se incluye un controlador para manejar errores (`ErrorController`) y uno para la integración con OpenAI (`OpenAIController`).
- **dtos (Data Transfer Objects)**: Incluye clases que encapsulan los datos que se transfieren entre el frontend y el backend, como `FighterDto`, `UserAppDto`, `FavFightDto`, etc. Ayudan a separar la lógica de negocio del modelo de persistencia y controlan los datos expuestos a las vistas.
- **repositories**: Este paquete contiene las consultas SQL de forma explícita, hachas con `jdbcTemplate`. Aquí se encuentran los repositorios de usuarios, peleadores, favoritos y simulaciones.
- **security**: Se encarga de la configuración de seguridad de la aplicación. Incluye `SecurityConfig` (donde se define el sistema de autenticación y autorización), `CustomOAuth2User` para gestionar usuarios autenticados vía Google, y `UserAppInfo`, clase auxiliar para manejar datos del usuario autenticado.
- **services**: Contiene la lógica de negocio. Cada clase de servicio (`FighterService`, `UserAppService`, etc.) se comunica con los repositorios y devuelve los datos necesarios a los controladores. También se encuentra aquí la integración con OpenAI y Google OAuth.

- **resources/static**: Contiene los recursos estáticos como CSS, imágenes, fuentes y JavaScript. Se estructura en subdirectorios para una organización clara:
 - **css**: hojas de estilo.
 - **fonts**: tipografías personalizadas.
 - **img** y **logos**: recursos gráficos utilizados en la interfaz.
 - **js**: scripts comunes a todas las páginas y librerías externas .
- **resources/templates**: Incluye las plantillas HTML gestionadas por Thymeleaf. Están organizadas por función:
 - **fragments**: componentes reutilizables como cabecera, pie de página, preloader, etc.
 - **error**: plantillas de error personalizadas (404, 500).
 - Otras vistas como **home**, **login**, **fighters**, **simulator**, **ranking**, etc.
- **resources/db**: Archivos **schema.sql** y **data.sql** que permiten la inicialización automática de la base de datos al ejecutar la aplicación durante el desarrollo en H2.
- **application.properties**: Archivo clave para la configuración de base de datos, seguridad, puertos, rutas de recursos, conexión con servicios externos, etc.

5.2 Patrón MVC Aplicado

La arquitectura del proyecto se basa en el patrón **MVC (Modelo - Vista - Controlador)**, una práctica común en aplicaciones Spring Boot por su claridad y mantenimiento.

- **Modelo**:
 - Representado por los **DTOs** que mapean los datos recibidos de los repositorios. Estas clases encapsulan los datos necesarios para ser procesados o visualizados sin exponer detalles sensibles o innecesarios.
- **Vista**:
 - Se implementa con **Thymeleaf**, un motor de plantillas que permite generar páginas HTML dinámicas desde el backend. Las vistas se componen con fragmentos reutilizables y plantillas específicas para cada vista.
 - Además, se integran recursos estáticos (CSS, JS, imágenes) y librerías externas como Chart.js, Google Fonts, SweetAlert 2, etc., para mejorar la experiencia visual y la interacción del usuario.
- **Controlador**:
 - Cada controlador recibe las peticiones del cliente, invoca a los servicios necesarios y decide qué vista renderizar o qué datos devolver.

Esta separación favorece una estructura escalable, testable y fácil de mantener. Permite modificar las vistas sin afectar la lógica del negocio, o cambiar la persistencia sin alterar los controladores o vistas.

5.3 Controladores (Controllers)

5.3.1 MainController

Se introdujo el **MainController**, un controlador básico encargado de gestionar las rutas principales de la aplicación web. Se desarrollaron dos métodos con mapeo GET: uno para la ruta raíz ("/") que carga la vista principal llamada "home", y otro para la ruta "/privacy", que muestra la página de privacidad. En ambos métodos se agregó un atributo al modelo llamado "page" con valor "home", que sirve para que quede marcado en el header en qué página estás.

5.3.2 ErrorController

Se desarrolló el **ErrorController**, responsable de manejar las páginas de error en la aplicación. Se implementó un método que intercepta las solicitudes dirigidas a la ruta "/error" y, según el código de estado HTTP recibido (como 404 o 500), redirige a la plantilla HTML correspondiente para mostrar mensajes de error personalizados. Esta gestión permite mejorar la experiencia del usuario ante errores comunes, mostrando páginas específicas para "No encontrado" y "Error interno del servidor", y una página genérica para otros casos.

5.3.3 RankingController

Se desarrolló el **RankingController**, cuyo propósito es gestionar la vista del ranking de peleadores. Se introdujo un método mapeado a la ruta "/ranking" que agrega al modelo dos atributos: uno para identificar la página activa ("page" con valor "ranking") y otro que contiene la lista de peleadores obtenida a través del servicio **FighterService** mediante el método `findByRankingBetween()`. Esto permite mostrar dinámicamente el ranking actualizado en la plantilla "ranking".

5.3.4 FighterController

Se introdujo el **FighterController**, responsable de gestionar las operaciones relacionadas con los peleadores dentro de la aplicación. Se desarrolló un método para listar peleadores con paginación, búsqueda y ordenamiento dinámico, integrando la información de los peleadores favoritos del usuario autenticado para una experiencia personalizada. Además, se implementaron dos endpoints REST para agregar y eliminar peleadores favoritos mediante solicitudes POST y DELETE, respectivamente, gestionando estas acciones de forma segura con la información del usuario autenticado. Esta estructura facilita tanto la interacción con la interfaz web como la comunicación asíncrona con el frontend.

5.3.5 SimulatorController

Se desarrolló el **SimulatorController**, encargado de manejar la funcionalidad del simulador de peleas. Se implementó un método GET para mostrar la vista del simulador, que permite seleccionar categoría y peleadores, cargar sus datos y gestionar la interacción con el usuario autenticado. Además, se añadió un endpoint POST para guardar los resultados de una pelea simulada, recibiendo un objeto DTO con los detalles y persistiendo esta información mediante el servicio correspondiente. Esta estructura facilita la simulación y almacenamiento de combates de manera dinámica y personalizada.

5.3.6 UserController

Se desarrolló el **UserController**, orientado a gestionar las acciones relacionadas con los usuarios dentro de la aplicación. Se introdujeron rutas para visualizar las páginas de login y registro, así como para registrar nuevos usuarios, incluyendo validaciones y manejo de errores.

Además, se implementó la visualización del perfil del usuario autenticado y una salida controlada mediante la ruta `/logout`. También se incorporaron dos endpoints POST que permiten, por un lado, la **eliminación** de cuentas y, por otro, la **actualización** de datos del usuario (nombre, correo, contraseña), utilizando el servicio correspondiente y validando credenciales mediante el campo de contraseña actual. Todo el flujo asegura una experiencia segura y centrada en el usuario, con una gestión eficaz de las credenciales y datos personales.

5.3.7 FavoriteController

Se desarrolló el **FavoriteController**, encargado de gestionar la visualización y manipulación de elementos favoritos del usuario, tanto peleadores como combates simulados.

Se introdujo un método GET vinculado a la ruta `/favorites` que carga la vista correspondiente, integrando en el modelo las listas de peleadores y combates favoritos del usuario autenticado, junto con todos los peleadores disponibles, facilitando así una navegación completa dentro del apartado de usuario.

Además, se implementaron dos endpoints para eliminar favoritos: uno para combates y otro para peleadores. Estos endpoints permiten eliminar elementos de forma asíncrona a partir de los identificadores recibidos, garantizando una experiencia interactiva y dinámica en la gestión de favoritos.

5.3.8 OpenAIController

Se desarrolló el **OpenAIController**, cuya finalidad es gestionar las interacciones entre la aplicación y el servicio de inteligencia artificial de OpenAI.

Se introdujo un endpoint POST en la ruta `/api/openai/chat` que recibe un `prompt` en formato JSON, lo envía al servicio `OpenAIClientService`, y devuelve la respuesta generada por el modelo de lenguaje. Se incorporó un manejo de errores que captura

cualquier excepción durante el proceso y responde con un mensaje adecuado y un estado HTTP 500, informando al cliente sobre la necesidad de recargar la página.

Esta funcionalidad amplía las capacidades de la aplicación al permitir la generación de texto dinámico e inteligente basado en entradas del usuario.

5.4 Data Transfer Objects (DTOs)

5.4.1 FighterDto

Se definió el **FighterDto**, un objeto de transferencia de datos que encapsula la información relevante de un peleador.

Se implementaron atributos detallados como nombre, apodo, nacionalidad, edad, y medidas físicas (peso, altura, alcance), además de estadísticas completas de su historial de peleas (total de combates, victorias y derrotas por diferentes modalidades y empates) y su ranking actual.

Se utilizó Lombok para simplificar la creación de getters, setters, constructores y el patrón builder, facilitando la manipulación y transferencia de datos entre capas sin exponer directamente la entidad de persistencia, contribuyendo así a mantener una arquitectura limpia y desacoplada.

5.4.2 UserAppDto

Se definió el **UserAppDto**, un Data Transfer Object que implementa las interfaces **UserDetails** y **UserAppInfo** para integrar la información del usuario en el sistema de seguridad de Spring Security.

Se introdujeron atributos clave como el identificador del usuario, nombre de usuario, correo electrónico, contraseña y los roles o autoridades que posee el usuario.

Además, se implementaron los métodos requeridos por **UserDetails** para indicar el estado de la cuenta (no expirada, no bloqueada, credenciales válidas y habilitada), asegurando así una correcta compatibilidad con el framework de seguridad.

Esta clase facilita el transporte seguro y estructurado de los datos del usuario autenticado entre las capas de la aplicación.

El motivo de llamarse UserApp es debido a que User es una palabra reservada en SQL y Java y daba problemas durante el desarrollo de la aplicación.

5.4.3 FavFighterDto

Se definió el **FavFighterDto**, un objeto de transferencia simple que representa la relación de un peleador favorito asociado a un usuario.

Contiene únicamente los identificadores del usuario (`id_user_app`) y del peleador (`id_fighter`), facilitando la comunicación entre capas para operaciones relacionadas con favoritos sin exponer entidades completas.

Se aplicaron anotaciones de Lombok para generar automáticamente métodos comunes y construir instancias de forma fluida, manteniendo la estructura clara y concisa.

5.4.4 FavFightDto

Se desarrolló el **FavFightDto** como un objeto de transferencia de datos destinado a encapsular información sobre peleas favoritas guardadas por los usuarios.

Este DTO incluye los identificadores del combate (`id_fight`), del usuario (`id_user`) y de ambos peleadores involucrados (`id_fighter1`, `id_fighter2`), junto con los porcentajes asignados a cada peleador (`percentage_f1`, `percentage_f2`) y la fecha del combate (`date_fight`).

Se emplearon las anotaciones de Lombok para simplificar la escritura del código, generando automáticamente constructores, getters y setters. Esto facilita la transferencia segura y estructurada de datos entre las capas de la aplicación, sin acoplar directamente con las entidades persistentes.

5.5 Repositorios (Repositories)

5.5.1 FighterRepository

Se implementó el **FighterRepository** como el componente encargado de la persistencia y consulta de datos relacionados con los peleadores, utilizando `JdbcClient` para interactuar directamente con la base de datos mediante SQL.

Se desarrollaron varios métodos para:

- Consultar peleadores según un rango de ranking (`findByRankingBetween`).
- Obtener peleadores por su ID (`findAllByIdFighter`).
- Filtrar peleadores por categoría (`findAllByCategory`).
- Buscar peleadores paginados filtrando por nombre o apodo (`findByNameOrNicknamePaged`), aplicando paginación manual.

- Listar todos los peleadores (`findAll`) y listar todos con paginación y orden dinámico (`findAll(Pageable)`).

En resumen, se creó un repositorio que facilita consultas flexibles y eficientes, apoyado en paginación y ordenamiento, optimizando así la gestión y recuperación de datos para la aplicación.

5.5.2 UserAppRepository

UserAppRepository está estructurado para manejar las operaciones básicas CRUD sobre la entidad `UserAppDto`, usando `JdbcClient` para consultas SQL.

Aquí un resumen de lo que hace:

- **findByUsername** y **findByEmail**: buscan un usuario por su nombre de usuario o email y devuelven un `Optional<UserAppDto>`.
- **deleteUser**: elimina un usuario por su id.
- **save**: inserta un nuevo usuario en la tabla `user_app`.
- **updateUser**: actualiza dinámicamente el username, email y/o password de un usuario según los parámetros no nulos y no vacíos que se reciban.

Un punto a destacar en `updateUser` es la construcción dinámica del SQL para actualizar solo los campos que vienen con valores válidos, evitando actualizar con campos vacíos o nulos.

5.5.3 FavFighterRepository

FavFighterRepository está diseñado para manejar las operaciones básicas relacionadas con los peleadores favoritos de un usuario, utilizando `JdbcClient` para ejecutar consultas SQL.

Aquí un resumen de lo que hace:

- **save**: inserta una relación entre un usuario y un peleador favorito en la tabla `fav_fighter`.
- **deleteById**: elimina todas las entradas de `fav_fighter` que tengan el `id_fighter` especificado.
- **findByIdUserApp**: devuelve una lista de objetos `FavFighterDto` con los peleadores favoritos de un usuario.
- **findIdFighterByIdUserApp**: obtiene solo los IDs de los peleadores favoritos asociados a un usuario.

5.5.4 FavFightRepository

FavFightRepository está enfocado en manejar las operaciones básicas para la entidad FavFightDto, usando JdbcClient para ejecutar las consultas SQL.

Aquí un resumen de lo que hace:

- save: inserta un nuevo registro de pelea favorita en la tabla fav_fight con los datos proporcionados.
- deleteById: elimina una pelea favorita específica por su id_fight.
- findByIdUserApp: obtiene una lista de todas las peleas favoritas asociadas a un usuario dado por su id_user_app.

Un punto a destacar es que las operaciones están centradas en la relación usuario-pelea, permitiendo guardar, borrar y consultar las peleas favoritas vinculadas a un usuario específico.

5.6 Servicios (Services) y Lógica de Negocio

5.6.1 FighterService

FighterService encapsula la lógica para obtener datos de peleadores a través del FighterRepository, gestionando filtros, búsquedas y paginación.

Aquí un resumen de lo que hace:

- findByIdFighter: busca un peleador por su id.
- findFightersByCategory: obtiene todos los peleadores que pertenecen a una categoría específica.
- findByRankingBetween: devuelve peleadores cuyo ranking está entre 0 y 15.
- findAll: devuelve la lista completa de peleadores.
- findFilteredAndSorted: combina búsqueda por nombre o apodo, junto con ordenamiento dinámico (peso, victorias totales, victorias por KO o sumisión) y paginación, devolviendo una página de resultados según los criterios recibidos.

Un punto a destacar es la implementación flexible de filtros y ordenamientos, utilizando la clase Pageable y Sort para facilitar consultas dinámicas y paginadas desde el repositorio.

5.6.2 UserAppService

UserAppService gestiona la lógica de negocio para la entidad UserAppDto, implementando UserDetailsService para integración con Spring Security y usando UserAppRepository para persistencia.

Aquí un resumen de lo que hace:

- registerUser: registra un nuevo usuario validando que username y email no existan, además de validar formato de email y fortaleza de la contraseña. La contraseña se codifica antes de guardarla.
- deleteUser: elimina un usuario validando que la contraseña actual coincida con la almacenada.
- updateUser: actualiza el username, email y/o contraseña del usuario. Verifica la contraseña actual, valida los nuevos datos, y codifica la nueva contraseña si se cambia.
- loadUserByUsername: método requerido por Spring Security para cargar usuario por username.
- validatePassword: valida que la contraseña cumpla con longitud, letras mayúsculas y minúsculas, números y caracteres especiales.
- validateEmail: valida formato básico del email y restringe dominios a gmail, hotmail, outlook o yahoo.

Un punto a destacar es la gestión rigurosa de validaciones y seguridad, garantizando que solo se guarden datos correctos y que la contraseña se maneje de forma segura.

5.6.3 FavFighterService

El FavFighterService es una capa de servicio que gestiona la lógica para manejar los favoritos de peleadores, delegando todas las operaciones al FavFighterRepository.

Aquí un resumen de lo que hace:

- save: guarda un favorito relacionando un usuario con un peleador.
- deleteById: elimina un favorito usando el id del peleador.
- findByIdUserApp: devuelve la lista completa de favoritos para un usuario dado.
- findIdFighterByIdUserApp: devuelve solo los IDs de peleadores favoritos de un usuario.

Este servicio actúa como intermediario, facilitando el acceso a los datos de favoritos sin agregar lógica adicional compleja.

5.6.4 FavFightService

El FavFightService es una capa de servicio que maneja las operaciones relacionadas con las peleas favoritas, delegando las consultas y modificaciones al FavFightRepository.

Aquí un resumen de lo que hace:

- save: guarda una pelea favorita con los datos completos del objeto FavFightDto.
- deleteById: elimina una pelea favorita usando su id.
- findByIdUserApp: obtiene la lista de peleas favoritas asociadas a un usuario específico.

Este servicio actúa como intermediario simple, proporcionando una interfaz limpia para gestionar las peleas favoritas sin lógica adicional compleja.

5.6.5 CustomOAuth2UserService

El CustomOAuth2UserService extiende DefaultOAuth2UserService y se encarga de gestionar la autenticación OAuth2 personalizada integrando usuarios externos con la base de datos interna.

Aquí un resumen de lo que hace:

- loadUser: obtiene la información del usuario desde el proveedor OAuth2.
- Extrae el email y nombre del usuario autenticado.
- Busca el usuario en la base de datos por email; si no existe, registra uno nuevo con una contraseña dummy.
- Devuelve un objeto CustomOAuth2User que combina los datos del usuario local con los atributos OAuth2.

Este servicio permite la integración transparente de usuarios OAuth2 con la gestión interna de usuarios, creando usuarios automáticamente si no existen aún.

5.6.6 OpenAIClientService

El OpenAIClientService es una capa de servicio que encapsula la comunicación con la API de OpenAI para obtener respuestas de chat basadas en un prompt.

Aquí un resumen de lo que hace:

- En el constructor, inicializa un WebClient apuntando a la base URL de OpenAI y recibe la API key desde configuración.
- getChatCompletion: envía una petición POST a la API de chat completions con el prompt dado, usando el modelo "gpt-3.5-turbo".
- Procesa la respuesta bloqueando hasta obtener el resultado y extrae el texto de la respuesta dentro de la estructura JSON.
- Devuelve el contenido textual generado o un mensaje de error si no hay respuesta válida.

Este servicio ofrece el consumo de la API de OpenAI, el motivo de usar el modelo gpt-3.5 turbo es debido a que es un modelo más barato respecto a modelos superiores y que el prompt enviado y la respuesta recibida no necesitan una carga de procesamiento alta.

5.7 Plantillas Thymeleaf y Recursos Estáticos

5.7.1 Estructura HTML y Fragments

Se ha usado HTML básico, sin incluir meta etiquetas, solo elementos comunes como div, p, h1-h6, span, etc. Para organizar la estructura y evitar duplicación, utilizo fragments de Thymeleaf: header, footer, preloader y head.

El fragmento **head** contiene todos los imports necesarios y las rutas, lo que centraliza los recursos como CSS y scripts.

Con Thymeleaf, estos fragments se incluyen fácilmente en las páginas, permitiendo mantener una estructura limpia y modular.

5.7.2 Tailwind: Responsividad

Para el diseño y la responsividad se utilizó Tailwind CSS, que me permite aplicar clases utilitarias directamente en el HTML.

Gracias a Tailwind, el diseño se adapta a diferentes tamaños de pantalla (móviles, tablets, desktops) sin necesidad de escribir mucho CSS personalizado.

Esto facilita crear un diseño flexible y moderno con menos esfuerzo, aprovechando sus utilidades para espaciado, tamaños, colores, etc.

5.7.3 JavaScript y librerías Chart.js y SweetAlert 2

Existe un archivo llamado **comun.js** donde se centralizaron funciones JavaScript comunes para la aplicación.

Entre esas funciones están:

- El control del **preloader**, que muestra una animación mientras carga la página.
- El **toggle del menú** para la versión móvil, que permite abrir y cerrar el menú lateral o hamburguesa.
- Mostrar **frases aleatorias en el footer**, para dar dinamismo y personalización.

Además, se usó la librería **Chart.js** para crear gráficos dinámicos y visualmente atractivos, y **SweetAlert 2** para mostrar alertas y diálogos personalizados, mejorando la interacción con el usuario.

Inicialmente se utilizó **JFreeChart**, una biblioteca de Java para la generación de gráficos. A partir de los datos disponibles, esta herramienta creaba un gráfico en formato de imagen que luego se mostraba en la plantilla. Sin embargo, presentaba dos problemas importantes: el tiempo de generación era excesivo y la calidad de la imagen resultante era deficiente. Por esta razón, se decidió reemplazarla por una biblioteca desarrollada en **JavaScript**, que ofrecía mejores resultados en términos de rendimiento y calidad visual.

5.7.4 CSS y fuentes

En el proyecto se usó un CSS personalizado donde se definieron las fuentes y estilos específicos.

Para las fuentes, se incluyó una fuente externa llamada **Sonic Extra Bold** mediante `@font-face`, cargando el archivo `sonic-extra-bold-bt.ttf` desde la carpeta de fonts. Esta fuente se usa con la clase `.sonic-text` para textos destacados con un estilo extra bold.

También se utilizó la fuente **Koulen** importada desde Google Fonts con el link correspondiente, que se aplica globalmente al body y con la clase `.koulen-font`, dándole un estilo limpio y coherente a todo el sitio.

En cuanto a estilos CSS, manejo clases para imágenes de fondo como `.img_random`, que controla el comportamiento del fondo para que no se repita y siempre se muestre centrado y cubriendo el área.

Además, se incluyó un efecto visual llamado **card flipper** para tarjetas que giran al pasar el mouse, usando las clases `.flip-card`, `.flip-card-inner`, `.flip-card-front` y `.flip-card-back`. Este efecto usa transformaciones 3D con `perspective` y `rotateY` para dar una animación suave de volteo, mostrando contenido diferente en cada cara de la tarjeta.

5.7.5 Imágenes y logos

En la aplicación se usan diversas imágenes para mejorar la experiencia visual y aportar dinamismo.

Dispongo de 14 imágenes diferentes que se muestran de forma aleatoria en las páginas de **/home**, **/login** y **/register**, aportando variedad y frescura cada vez que el usuario accede a estas vistas.

Además, utilizo 3 gifs específicos para situaciones concretas:

- Un gif para la página **404** que aparece cuando se navega a una ruta no encontrada.
- Un gif en la página del **simulador**, aportando un toque visual llamativo antes de la elección de peleadores en la simulación.
- Un **preloader gif** que se muestra mientras la IA procesa y devuelve la respuesta del prompt, mejorando la percepción de carga y respuesta del sistema.

Respecto a los logos, se usan principalmente para:

- El **icono de la pestaña** del navegador (favicon), para dar identidad visual al sitio en las pestañas del navegador.
- Logos en formato **SVG** que se cargan como respaldo en caso de que falle la carga o disponibilidad de las fuentes externas, garantizando que siempre haya un logo visible con buena calidad y escalabilidad.

Este manejo de imágenes y logos contribuye a una interfaz más atractiva, interactiva y profesional. Durante el despliegue surgieron problemas de rendimiento debido a la lenta carga de las imágenes debido al peso y resolución de las mismas por ello se decidió cambiar el formato de jpg a webP ya que mejoró el rendimiento de forma radical y la mínima pérdida de calidad de imagen fue asumible

5.8 Seguridad

5.8.1 SecurityConfig (SpringSecurity)

SecurityConfig está configurado para gestionar la seguridad de la aplicación usando Spring Security, con soporte para autenticación tradicional y OAuth2.

Aquí un resumen de lo que hace:

- **h2ConsoleSecurityFilterChain**: permite acceso libre a la consola H2 en **/h2-console/****, desactivando CSRF y restricciones de iframe para que funcione correctamente, esto se usó durante el desarrollo.

- `securityFilterChain`: define las reglas generales de seguridad, permitiendo acceso público a rutas como `/`, `/login`, `/register`, recursos estáticos y endpoints OAuth2, y requiere autenticación para el resto.
- Configura el login con formulario personalizado en `/login`, definiendo URLs para éxito, fallo y logout.
- Integra OAuth2 login con un servicio personalizado `CustomOAuth2UserService` para manejar usuarios autenticados vía proveedores externos.
- Usa `UserDetailsService` para gestionar usuarios con seguridad tradicional.
- Define un bean `PasswordEncoder` con `BCryptPasswordEncoder` para cifrado seguro de contraseñas.

Este archivo centraliza y organiza la configuración de seguridad, ofreciendo flexibilidad para múltiples métodos de autenticación y control fino de acceso a recursos.

5.8.2 CustomOAuth2UserService

La clase `CustomOAuth2User` implementa `OAuth2User` y una interfaz personalizada `UserAppInfo` para representar a un usuario autenticado vía OAuth2, integrando los datos propios de la aplicación con la información proporcionada por el proveedor OAuth.

Aquí un resumen de lo que hace:

- Contiene un objeto `UserAppDto` que representa al usuario en la aplicación y un mapa con los atributos devueltos por el proveedor OAuth2.
- Implementa `getAttributes()` para devolver los datos originales del proveedor OAuth.
- Expone métodos para obtener el nombre de usuario (`getUsername()`), email (`getEmail()`) e ID (`getId()`) del usuario desde el DTO interno.
- Implementa `getAuthorities()` para devolver los roles o permisos asignados al usuario, este método está sobrescrito (`@Override`) pero no se usa ya que la aplicación no usa roles de usuario.
- Sobrescribe `getName()` para devolver el username, como requiere la interfaz `OAuth2User`.

Esta clase actúa como un puente entre el modelo de usuario interno y la autenticación externa vía OAuth2, facilitando la integración segura y consistente de ambos sistemas.

5.8.3 UserAppInfo

La interfaz `UserAppInfo` define un contrato para exponer la información básica necesaria de un usuario dentro del sistema de seguridad de la aplicación.

Aquí un resumen de lo que define:

- `getUsername()`: devuelve el nombre de usuario como cadena.
- : devuelve el correo electrónico del usuario.
- `getId()`: devuelve el identificador único del usuario (Integer).
- `getAuthorities()`: devuelve una colección de roles o permisos (`GrantedAuthority`) asignados al usuario.

Esta interfaz sirve para abstraer y unificar el acceso a los datos esenciales de usuario que pueden provenir de diferentes fuentes o modelos (por ejemplo, usuarios internos, usuarios autenticados vía OAuth2, etc.), facilitando la integración con Spring Security.

5.9 Integración con OpenAI

5.9.1 Prompt Design

El diseño del *prompt* es fundamental para obtener respuestas relevantes y precisas desde la API de OpenAI. En este proyecto, los *prompts* se construyen dinámicamente en el frontend con datos concretos, como los nombres y apellidos de los peleadores en una simulación de pelea. Por ejemplo, el prompt para simular una pelea entre dos peleadores se estructura con instrucciones claras y específicas, indicando el formato de la respuesta (descripción breve de cada ronda y el nombre del ganador) y el idioma en que se espera la respuesta (inglés). Este enfoque facilita que el modelo entienda el contexto y entregue resultados coherentes y útiles para la aplicación.

5.9.2 Llamadas a la API y Control de Errores

Las llamadas a la API de OpenAI se realizan desde el backend a través del servicio `OpenAIClientService`, que utiliza `WebClient` para enviar solicitudes POST con el prompt al endpoint `/chat/completions`. El servicio incluye la gestión básica de la respuesta, extrayendo el texto generado y manejando posibles casos donde la respuesta no sea válida o esté vacía.

En el controlador `OpenAIController`, se expone un endpoint REST `/api/openai/chat` que recibe el prompt desde el frontend, llama al servicio y devuelve la respuesta al cliente. Se implementa un control de errores para capturar excepciones durante la llamada a la API o procesamiento, devolviendo un mensaje de error claro y un código HTTP 500 cuando algo falla.

En el frontend, la función `askOpenAI` realiza la petición al backend, muestra un loader mientras espera la respuesta y maneja errores de red o comunicación, mostrando un mensaje adecuado si ocurre algún problema. Esto asegura una experiencia de usuario fluida y robusta ante fallos temporales en la API o la conexión.

5.10 Flujo de Datos

5.10.1 De Frontend a Backend

El frontend envía solicitudes HTTP al backend. Por ejemplo, cuando se genera un prompt para la simulación de una pelea, el frontend construye dinámicamente el contenido y lo envía mediante un POST al endpoint `/api/openai/chat`. Esta comunicación utiliza JSON para transmitir la información, facilitando el manejo y la interoperabilidad. En su mayoría se usan peticiones GET para servir las vistas y POST para ejecutar acciones como guardar, modificar o eliminar.

El backend recibe estas peticiones en controladores, que actúan como puntos de entrada para procesar la información. El controlador extrae los datos necesarios del cuerpo de la petición, los valida y gestiona la lógica necesaria, para luego usar los servicios correspondientes.

5.10.2 De Backend a la Base de Datos

La persistencia de datos se realiza a través de repositorios que interactúan con la base de datos mediante JDBC usando las consultas explícitamente para tener mayor control y poder separar el proceso del desarrollo de la base de datos y de la aplicación Spring Boot. En un inicio se usó JPA pero se prefirió hacerlo con JDBC por la posibilidad de cambios en la base de datos durante el desarrollo y poder separarla de la propia aplicación. Inicialmente, el proyecto usa una base de datos en memoria H2 para facilitar el desarrollo local y pruebas rápidas. Los scripts y configuraciones de H2 (`schema.sql`, `data.sql` o similares) definen la estructura y datos iniciales.

Sin embargo, para el despliegue en producción, la aplicación está diseñada para migrar a PostgreSQL alojado en Heroku, una base de datos más robusta y adecuada para entornos en la nube. Esta migración implica que los scripts y configuraciones específicas de H2 ya no serán necesarios, ya que PostgreSQL gestionará la persistencia de datos.

Esta separación y abstracción permiten que el backend no dependa directamente del tipo de base de datos, facilitando la escalabilidad y mantenimiento a largo plazo sin cambios significativos en el código de servicio o controlador.

5.11 Testing y Validación del Código y Base de datos

5.11.1 Testing Manual

Durante el desarrollo se ha realizado testing manual para validar las funcionalidades principales y detectar posibles errores o comportamientos no deseados. Esto incluye:

- Pruebas de los endpoints REST, enviando diferentes tipos de solicitudes desde el frontend o mediante herramientas como Postman para verificar las respuestas esperadas.
- Validación de la interacción con la base de datos, asegurando que las operaciones CRUD funcionen correctamente tanto en la base de datos H2 de desarrollo como en PostgreSQL en producción.
- Pruebas de la interfaz de usuario para comprobar que la navegación, carga de datos y funcionalidades JavaScript (por ejemplo, la llamada a la API OpenAI y la gestión del preloader) funcionen de forma fluida y sin errores.
- Verificación visual y funcional de elementos dinámicos como gráficos con Chart.js y alertas con SweetAlert2.
- Responsividad, probándolo en diferentes tamaños de pantallas como móviles, tablets y escritorio.

Este enfoque manual permite encontrar y corregir rápidamente errores antes del despliegue.

6. Despliegue y Producción

Esta sección describe las diferentes estrategias de despliegue utilizadas para ejecutar la aplicación en entornos reales o simulados. Se exploraron dos métodos: despliegue en Heroku y servidor personal de desarrollo. Cada uno con sus propias configuraciones y consideraciones.

6.1 Despliegue en Heroku

Heroku es una plataforma como servicio (PaaS) que permite desplegar aplicaciones Java de forma sencilla. En este proyecto se utilizó para facilitar el acceso a la aplicación desde cualquier navegador, sin necesidad de contar con infraestructura local.

Durante el proceso de despliegue surgieron varios problemas que se fueron resolviendo progresivamente. Uno de los primeros inconvenientes fue que el archivo `pom.xml` no se encontraba en la raíz del proyecto, lo cual impedía que Heroku reconociera y compilara correctamente la aplicación. Además, fue necesario configurar explícitamente el entorno para que Heroku supiera que se trataba de una aplicación Java.

En cuanto a la base de datos, la instancia de PostgreSQL proporcionada por Heroku no fue suficiente al utilizar únicamente la interfaz gráfica. Se requirió instalar el cliente de línea de comandos para poder crear las tablas y realizar los inserts manualmente.

También hubo dificultades al trabajar en local con las variables de entorno, ya que la base de datos de Heroku requiere una conexión SSL. Mientras que en Heroku el certificado SSL se genera automáticamente, en local no fue posible establecer la conexión. Para solucionar esto, se ajustó la configuración de la aplicación para que no exigiera el uso del certificado SSL en el entorno de desarrollo local.

6.1.1 Configuración del `application.properties`

Para permitir que Heroku gestione las credenciales de conexión a la base de datos, se configuró el archivo `application.properties` para que lea variables de entorno proporcionadas automáticamente por la plataforma. Así se evita el almacenamiento directo de información sensible dentro del código fuente.

Este enfoque permite que Heroku inyecte automáticamente los valores de conexión de su base de datos PostgreSQL asignada al proyecto, además de mantener ocultas las API Keys de GoogleAuth y OpenAI.

6.1.2 Variables de Entorno y Seguridad

Se utilizaron variables de entorno definidas en el panel de control de Heroku para gestionar datos como:

- La URL, usuario y contraseña de la base de datos PostgreSQL.
- Credenciales de servicios externos como Google OAuth.
- Cualquier otra clave privada o configuración sensible.

Estas variables están cifradas y accesibles únicamente durante la ejecución, lo que mejora la seguridad de la aplicación.

6.2 Despliegue en Servidor Personal (Desarrollo)

Además, se realizó un despliegue en un servidor personal para facilitar pruebas de desarrollo en un entorno más controlado. Se abrió el puerto 80 en el router y, mediante FreeDNS, se configuró un nombre de dominio que apunta a la IP pública. Se ajustó el equipo local para que permaneciera encendido y ejecutando el servidor Tomcat, lo que permitió acceder al proyecto desde redes externas.

7. Conclusiones Finales

El desarrollo de este proyecto web completo con Spring Boot, seguridad, frontend dinámico y despliegue en distintos entornos supuso un reto que permitió poner en práctica una amplia variedad de tecnologías. A continuación, se resumen las lecciones extraídas del proceso y las mejoras que se proyectan a futuro.

7.1 Lecciones Aprendidas durante el Proyecto

El desarrollo de este proyecto permitió consolidar una serie de habilidades clave:

- **Comprensión de Spring Boot** y su ecosistema, especialmente en temas de seguridad, persistencia y despliegue.
- **Manejo de plantillas Thymeleaf** en combinación con JavaScript para enviar peticiones al Backend.
- **Buenas prácticas en diseño responsive**, modularidad de componentes y experiencia de usuario.
- **Gestión de errores y estados de carga** en interfaces dinámicas.
- **Valor de una base de datos bien diseñada**, especialmente al relacionar múltiples entidades (usuarios, peleadores, combates).
- **Uso estratégico de herramientas externas** (Chart.js, SweetAlert, Tailwind, OAuth, OpenAI) de manera controlada.

7.2 Mejoras Futuras e Ideas de Escalabilidad

Para seguir evolucionando este proyecto, se plantean varias ideas:

- **Implementar API REST completa** para consumo desde frontend moderno (React, Angular o apps móviles).
- **Mantenimiento de la base de datos** para tener datos actualizados al día.
- **Añadir más parámetros estadísticos** a la base de datos para tener unas estimaciones más realistas.
- **Panel de administración más robusto**, con gráficas y métricas de uso.
- **Soporte multilingüe** y adaptación para distintos países.
- **Despliegue en Docker o servicios escalables en la nube**, con balanceo de carga y monitoreo.

8. Bibliografía y Recursos

A lo largo del desarrollo del proyecto, se recurrió a una variedad de fuentes oficiales, tutoriales prácticos y bibliotecas externas que permitieron construir una aplicación sólida, visualmente atractiva y funcional. Esta sección recoge dichas referencias de forma estructurada.

8.1 Documentación Oficial

- **Spring Boot:** Utilizada para la configuración de seguridad, persistencia, controladores y despliegue.
- **Thymeleaf:** Guía de referencia para plantillas dinámicas, renderizado condicional y estructuras HTML extendidas.
- **Tailwind CSS:** Referencia central para estilos utilitarios, diseño responsive y estructura visual de componentes.
- **Chart.js:** Usada para construir gráficas dinámicas en el simulador de peleas.
- **PostgreSQL:** Documentación oficial para el diseño, relaciones y gestión de la base de datos relacional.
- **Heroku Dev Center**
<https://devcenter.heroku.com/>
Guía principal para el despliegue de la aplicación en la nube.
- **Google Cloud – OAuth 2.0**
<https://developers.google.com/identity>
Documentación empleada para la integración de inicio de sesión mediante Google.

8.2 Recursos y Tutoriales Utilizados

- **Baeldung:** Portal muy útil para resolver dudas sobre Spring Security, autenticación personalizada y uso de DTOs.
- **Stack Overflow:** Utilizado en la solución de errores concretos con Thymeleaf, Spring Boot, y comportamiento de JavaScript.
- **YouTube:** Diversos canales especializados en desarrollo backend y full-stack, usados para repasar conceptos de seguridad, despliegue y buenas prácticas.
- **IA:** Se utilizaron herramientas de inteligencia artificial, como GPT y Gemini, para apoyar el desarrollo del proyecto en distintas fases: planificación, implementación y aprendizaje de las tecnologías involucradas.

8.3 Créditos a APIs y Librerías Externas

- **SweetAlert 2:** Para mostrar alertas elegantes en acciones como guardar, eliminar o mostrar errores.
- **Chart.js:** Para visualización de estadísticas de peleas, porcentajes de simulación y comparativas entre peleadores.
- **Google Fonts:** Se utilizaron tipografías modernas como *Inter* y *Roboto* para mejorar la legibilidad y estética general.
- **Icons8:** Fuente principal de iconos visuales utilizados para acciones, botones y representación gráfica intuitiva.
- **OpenAI:** Considerado para funcionalidades futuras de simulación inteligente y generación de texto automático.