UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Informática

Team UP. Una plataforma web social para formar equipos de jugadores online

Trabajo Fin de Grado

Grado en Ingeniería Informática

AUTOR/A: Castillo Llorens, Alejandro

Tutor/a: Valderas Aranda, Pedro José

CURSO ACADÉMICO: 2023/2024

Resum

El treball realitzat consisteix en el desenvolupament d'una aplicació en què els usuaris poden subscriure's als video jocs d'interès. Cada un d'aquests jocs actua com el seu propi fòrum, oferint la possibilitat de *publicar* i crear sales de xat amb altres usuaris.

L'aplicació té com a objectiu ajudar als jugadors a socialitzar i compartir la seva afició pels videojocs, posant especial cura en crear un espai el més sa possible per als usuaris. Per aconseguir-ho, es evita la presència d'actors comercials a la plataforma així com les mètriques tradicionals de xarxes socials com *likes* o seguidors.

El desenvolupament de l'aplicació s'ha anat adaptant a les necessitats i funcionalitats requerides, tant inicials com noves, que han sorgit al llarg del projecte. Aquesta memòria reflecteix les diferents fases que ha comportat el projecte: recopilació de requisits, anàlisi i disseny, planificació, desenvolupament, proves, desplegament i manteniment.

D'altra banda, les tecnologies emprades també han evolucionat durant les diferents fases del projecte, consolidant els principals mòduls amb *Spring Boot* per al servidor, *React* per al client i una base de dades *MySQL* per a la persistència.

Aquest treball ha permès aplicar els coneixements adquirits durant la carrera per desenvolupar una plataforma innovadora que facilita la interacció entre jugadors, amb l'objectiu de fomentar la connexió autèntica i genuïna entre els usuaris. La flexibilitat en el seu desenvolupament assegura que s'adapti a noves necessitats i reptes futurs.

Paraules clau: React, Spring Boot, Scrum, Team UP, xarxa social, front-end, back-end, Redux, Hibernate

Resumen

El trabajo llevado a cabo consiste en el desarrollo de una aplicación en la que los usuarios pueden suscribirse a sus videojuegos de interés. Cada uno de estos juegos actúa como su propio foro, brindándoles posibilidad de *postear* y crear salas de chat con otros usuarios.

La aplicación tiene como objetivo ayudar a los jugadores a socializar y compartir su afición por los videojuegos, poniendo especial cuidado en crear un espacio lo más sano posible para los usuarios. Para lograr esto, se evita la presencia de actores comerciales en la plataforma así como las métricas tradicionales de redes sociales como *likes* o seguidores.

El desarrollo de la aplicación se ha ido adaptando a las necesidades y funcionalidades requeridas, tanto iniciales como nuevas, que han surgido a lo largo del proyecto. Esta memoria refleja las diferentes fases que ha conllevado el proyecto: recolección de requisitos, análisis y diseño, planificación, desarrollo, pruebas, despliegue y mantenimiento.

Por otro lado, las tecnologías empleadas también han evolucionado durante las diferentes fases del proyecto, consolidando los principales módulos con *Spring Boot* para el servidor, *React* para el cliente y una base de datos *MySQL* para la persistencia.

Este trabajo ha permitido aplicar los conocimientos adquiridos durante la carrera para desarrollar una plataforma innovadora que facilita la interacción entre jugadores, con el objetivo de fomentar la conexión auténtica y genuina entre los usuarios. La flexibilidad en su desarrollo asegura que se adapte a nuevas necesidades y desafíos futuros.

Palabras clave: React, Spring Boot, Scrum, Team UP, red social, front-end, back-end, Redux, Hibernate

Abstract

The work carried out involves developing an application where users can subscribe to their video games of interest. Each game acts as its own forum, providing the ability to post and create chat rooms with other users.

The application's goal is to help gamers socialize and share their passion for video games, with a focus on creating as healthy an environment as possible for users. To achieve this, the presence of commercial actors on the platform and traditional social media metrics like likes or followers are avoided.

The application's development has been adapted to the required needs and functionalities, both initial and new ones that have arisen throughout the project. This report reflects the different phases involved in the project: requirements gathering, analysis and design, planning, development, testing, deployment, and maintenance.

Additionally, the technologies used have also evolved throughout the different phases of the project, consolidating the main modules with *Spring Boot* for the server, *React* for the client, and a *MySQL* database for persistence.

This work has allowed the application of knowledge acquired during the degree to develop an innovative platform that facilitates interaction between gamers, aiming to foster authentic and genuine connections among users. The flexibility in its development ensures it adapts to new needs and future challenges.

Key words: React, Spring Boot, Scrum, Team UP, social network, front-end, back-end, Redux, Hibernate

Índice general

Índice de figuras

Índice de tablas

CAPÍTULO 1
Introducción

Es innegable que el nacimiento y evolución de Internet ha sido uno de los eventos más influyentes en el desarrollo y progreso de nuestra sociedad, haciendo que el impacto de este fenómeno en diversos aspectos de nuestras vidas sea prácticamente incalculable.

En este contexto de constante evolución digital, el presente trabajo introduce un modelo de red social diseñada específicamente para conectar a personas en torno a una forma de entretenimiento que ha ganado un protagonismo creciente: los videojuegos. Este fenómeno no solo se ha establecido como una industria multimillonaria, sino que también ha creado comunidades globales apasionadas por los videojuegos. La red social propuesta, denominada *Team UP*, está concebida como una plataforma dividida por foros, en la que cada juego se corresponde a un foro. A través de esta estructura, los usuarios tienen la oportunidad de comentar y debatir sobre sus juegos favoritos, discutir estrategias y novedades, y formar grupos o equipos con los que compartir su pasión y experiencias en el mundo del *gaming*.

Además, *Team UP* busca fomentar la interacción y la construcción de comunidades virtuales, permitiendo a los usuarios conectarse con otros jugadores que comparten intereses similares, independientemente de su ubicación geográfica. Esto no solo facilita el intercambio de conocimientos y la creación de lazos sociales, sino que también enriquece la experiencia de los jugadores al proporcionar un espacio donde pueden sentirse comprendidos y apoyados en su afición. En definitiva, *Team UP* se presenta como una propuesta destinada a aprovechar el poder de Internet para unir a personas a través de su amor por los videojuegos, creando un entorno dinámico y participativo que refleja las tendencias actuales en entretenimiento digital y conexión social.

1.1 Objetivos

El objetivo de este trabajo de fin de grado es desarrollar una red social que gira en torno a los videojuegos, que permita a los usuarios además de compartir esta afición crear una comunidad así como comentar y debatir al respecto. Algunos de los objetivos principales en el desarrollo de la aplicación son los siguientes:

- **Crear una red social agradable y pensada para los usuarios**. Para ello, se pondrá especial cuidado en promover la igualdad. No habrá jerarquías basadas en *likes* o seguidores, fomentando así una comunidad donde cada voz tenga el mismo peso y relevancia.

- **Evitar la capitalización**. Para ello, se eliminará la presencia de micro-pagos o publicidad, proporcionando una plataforma donde los usuarios puedan interactuar sin presiones comerciales.
- **Transformar el pasatiempo de los videojuegos**. En muchas ocasiones esta afición suele ser solitaria. Se pretende impulsar que se convierta en una actividad social. Esta plataforma pretende abrir las puertas a que los jugadores compartan su afición y socialicen con otros en base a este *hobby*, ayudándoles a encontrar equipo, compañeros de juego o simplemente pertenecer a una comunidad.
- **Desarrollo de una aplicación con tecnologías actuales**. Para ello, se abordan todas las fases del desarrollo de un producto *software*: recolección de requisitos, análisis y diseño, planificación, desarrollo, despliegue y mantenimiento.

1.2 Estructura del documento

En este apartado se detalla la estructura de la memoria:

- **Capítulo I**: Un preámbulo de la memoria, se explican los objetivos y se lista la estructura del documento.
- **Capítulo II**: Una breve introducción a las redes sociales en la actualidad y una comparación de *Team UP* respecto a redes sociales con objetivos similares.
- **Capítulo III**: Explicación de la metodología *Scrum* y su aplicación durante el desarrollo del trabajo.
- **Capítulo IV**: Proceso de análisis de requisitos y los diferentes casos de uso en la aplicación.
- **Capítulo V**: Diagrama UML de las clases mas relevantes del servidor y lista de componentes del cliente. También se muestran los prototipos de la interfaz y el esquema de base de datos de la aplicación.
- **Capítulo VI**: Arquitectura general elegida para la aplicación junto con la arquitectura del cliente y del servidor. Principales herramientas y software utilizados para su desarrollo y ejemplos de código de los puntos más relevantes.
- **Capítulo VII**: Se muestra el uso y comportamiento de la aplicación. Se explican los escenarios descritos en el capítulo **IV**.
- **Capítulo VIII**: Validación de la interfaz y proceso de despliegue.
- **Capítulo IX**: Conclusiones obtenidas durante el desarrollo del proyecto y planteamiento de futuros cambios y adiciones al proyecto. También se relaciona el proyecto con los estudios cursados en el grado.

CAPÍTULO 2
Estado del arte

Las redes sociales, aunque relativamente modernas, representan una adaptación contemporánea de un aspecto fundamental y ancestral de la humanidad: la sociabilidad y el asociacionismo. Desde tiempos inmemoriales, los seres humanos han buscado formar conexiones, comunidades y redes de apoyo, manifestando una inclinación natural hacia la interacción social y la cooperación. Las plataformas digitales actuales amplifican y facilitan estas dinámicas, permitiendo que las personas se conecten, compartan y colaboren a una escala sin precedentes, superando barreras geográficas y temporales. De este modo, las redes sociales modernas no son más que una evolución tecnológica de nuestra innata necesidad de relacionarnos y organizarnos en sociedad.

En cuanto a tipos de sociabilidad, podemos dividirlas en dos grupos:

- Sociabilidad formal (asociativa): Émile Durkheim en su obra *La división del trabajo social* (1893) [2], argumenta que las instituciones formales son cruciales para la cohesión social y el orden en sociedades complejas. Aunque no hay una cita directa de él sobre "sociabilidad formal" como tal, su análisis de las instituciones proporciona una base teórica para comprender cómo las interacciones dentro de estas estructuras contribuyen a la cohesión social.
- Sociabilidad informal (relacional): La sociabilidad informal se fundamenta en interacciones espontáneas y relaciones personales que no están reguladas por estructuras formales. En su obra *La presentación de la persona en la vida cotidiana* (1956) [3], Erving Goffman analizó cómo las personas gestionan sus interacciones sociales en contextos informales y de qué manera estas interacciones contribuyen a la construcción de la identidad y las relaciones personales.

De aquí en adelante, pondremos el foco en las redes sociales relacionales, donde encaja *Team UP*. A continuación, se presenta un breve análisis de otras redes sociales relevantes que comparten algunos objetivos con el modelo de red social propuesta en este proyecto:

- **X (Twitter)** [6] : Plataforma creada en 2006, es posiblemente una de las redes sociales con mas relevancia en la actualidad, contando con aproximadamente 500 millones de usuarios. Es un claro ejemplo de la presencia de la capitalización en las redes sociales, se puede ver reflejado en aspectos como:
  - Promociones. Donde particulares y empresas pagan para promocionar sus cuentas o publicaciones y llegar a más usuarios.
  - Publicidad y anuncios. En los tablones donde se exponen las publicaciones de tu interés, se muestra publicidad.

- Privilegios de pago. Una suscripción mensual que reduce el numero de anuncios y permite monetizar la cuenta. También aumenta el numero de caracteres permitidos por publicación.

También está presente el fenómeno de las cámaras de eco, ya que los usuarios ajustan el contenido que prefieren mediante *likes*, compartiendo publicaciones y siguiendo a perfiles de interés que generalmente comparten sus ideologías y puntos de vista. Con los ya mencionados *likes* y el número de seguidores, *X (Twitter)* es otro ejemplo de jerarquización por estatus en una red social, lo cual contribuye negativamente en la salud mental de algunos usuarios. En cuanto a funcionalidad, el rasgo más relevante que comparte con *Team UP* es el *feed* o tablón central en el que se muestran las publicaciones. En lugar de dividirse por temáticas, se organiza mostrando las publicaciones de los usuarios de interés. Está disponible tanto en móviles como en web, y está desarrollado en *JavaScript, Ruby, Scala y Java*.

Figura 2.1: Icono de *X*.

Fuente:https://www.freepik.es/fotos-vectores-gratis/twitter-x-logo-png

- **Discord** [4] : Lanzada en 2015, esta red social cuenta con aproximadamente 250 millones de usuarios activos. Destaca por permitir, además de mensajes de texto, la comunicación mediante llamadas de voz, videollamadas y contenido multimedia. Los usuarios pueden comunicarse entre ellos de forma directa o agrupándose en servidores, donde cada servidor es una colección de canales de texto y voz. Esta es la característica más cercana a *Team UP*, solo que los servidores pueden ser de cualquier temática. También está presente la capitalización, ya que ofrece dos planes de suscripción mensual para mejorar las prestaciones de la plataforma, ofreciendo mayores anchos de banda para transmitir contenido y más opciones de personalización del perfil. En sus inicios, se auto denominaba "La plataforma de los *gamers*", aunque esto cambió hace pocos años para dar cabida a cualquier tipo de comunidad. Está desarrollado con *JavaScript, Elixir, Rust, Python y C++*.

Figura 2.2: Icono de *Discrod*.

Fuente: https://support.discord.com/hc/es-419/community/posts/1500001083322-A-recommendation-for-the-new-Discord-logo

- **Reddit** [5] : Creado en 2005, cuenta actualmente con aproximadamente 73 millones de usuarios activos. Los foros se dividen en *subreddits*, áreas de interés creadas por los usuarios para organizar discusiones o crear comunidades. Una de las características más distintivas de esta plataforma es que los usuarios pueden votar a favor o en contra del contenido de otros usuarios, priorizando los contenidos con más votos a favor. También se puede ver la capitalización en la plataforma, ya que ofrece un plan de suscripción mensual para eliminar anuncios y aumentar el nivel de personalización de cada perfil. La funcionalidad más parecida a *Team UP* es la división en foros de los usuarios, solo que en este caso, en lugar de agruparse por videojuegos, los temas los crean los usuarios. Está desarrollado mayoritariamente en *python*.

Figura 2.3: Icono de *Reddit*.

Fuente: https://seeklogo.com/free-vector-logos/reddit

- **Fogges** [7] : Lanzada recientemente, Fogges es una red social que permite a los usuarios crear y unirse a grupos llamados "Fogges". Estos espacios están diseñados para facilitar la comunicación y la colaboración entre miembros, permitiendo la organización de eventos, la compartición de contenido multimedia y la discusión en tiempo real. Fogges está orientada a crear comunidades alrededor de intereses comunes, desde aficiones hasta proyectos profesionales. La plataforma ofrece herramientas para personalizar la experiencia del usuario, como la creación de perfiles personalizados y la posibilidad de integrar servicios de terceros. Actualmente, Fogges se enfoca en crear una experiencia social completa y segura para sus usuarios, fomentando la interacción positiva y el sentido de comunidad. Está desarrollado con tecnologías modernas como *JavaScript, Node.js, y MongoDB*.

Figura 2.4: Icono de *Fogges*.

Fuente: https://fogges.com/es

- **Twich** [8] : Lanzada en 2011, *Twitch* es una plataforma de transmisión en vivo que cuenta con aproximadamente 140 millones de usuarios activos mensuales. Es particularmente popular en la comunidad de videojuegos, aunque se ha expandido para incluir contenido de otros ámbitos como música, arte, cocina y charlas. Los usuarios pueden crear canales para transmitir contenido en vivo, interactuar con su audiencia a través de un chat en tiempo real y recibir donaciones o suscripciones que les permiten monetizar sus transmisiones. *Twitch* ofrece diferentes niveles de suscripción con beneficios adicionales para los usuarios y *streamers*, como *emotes* exclusivos y opciones de personalización. Está desarrollado con tecnologías como *Go, Python y C++* entre otras.

Figura 2.5: Icono de *Twich*.

Fuente: https://www.twitch.tv/

CAPÍTULO 3
Metodología

Para el desarrollo del proyecto se ha seguido la metodología [9]*Scrum*. Esta metodología ágil, ampliamente utilizada para la gestión y desarrollo de proyectos complejos, surgió a principios de la década de 1990, cuando Jeff Sutherland y Ken Schwaber presentaron este marco de trabajo durante la conferencia OOPSLA (Object-Oriented Programming, Systems, Languages and Applications). Desde entonces, *Scrum* se ha consolidado como una de las metodologías ágiles más populares, especialmente en el desarrollo de software, pero su aplicación se ha extendido a diversas industrias y tipos de proyectos.

Scrum se fundamenta en la idea de dividir proyectos complejos en ciclos de trabajo cortos y manejables llamados *sprints*, que suelen durar entre dos y cuatro semanas 3.1. Los principales fundamentos que guían la implementación de esta metodología son:

- Transparencia. Característica esencial que permite a los miembros del equipo conocer las tareas del proyecto y sus estados, permitiendo una mejor compresión del trabajo y del progreso tanto para miembros del equipo como para los *stakeholders*.
- Inspección. Proceso mediante el cual se revisan regularmente las distintas tareas del proyecto y el progreso hacia los objetivos establecidos. Este análisis constante permite detectar problemas actuales o que puedan surgir y también comprobar si se está siguiendo el plan inicialmente establecido.
- Adaptación. Basándose en la información obtenida durante las inspecciones, el equipo debe estar dispuesto a ajustar su enfoque para optimizar el producto. Esta flexibilidad es crucial en entornos dinámicos donde los requisitos y las circunstancias pueden cambiar rápidamente.

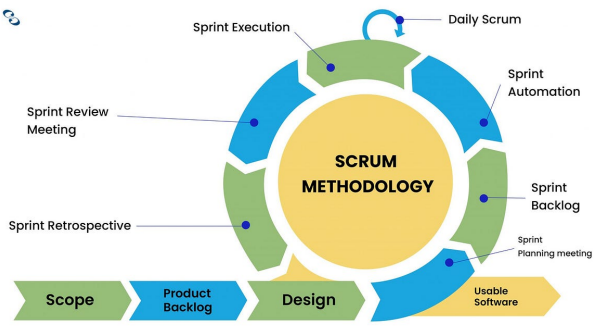

Figura 3.1: Esquema de *Scrum*.

Fuente:https://blog.stackademic.com/excelling-in-software-development-with-scrum-methodology-part-2-e2d0b29437ce

A continuación se resume el trabajo realizado en cada *Sprint*:

- *Sprint* 0. Preparación del entorno, esto implica descargar las herramientas software necesarias y configurarlas para tener el entorno de desarrollo preparado para el resto del proyecto.
- *Sprint* 1. Se realizan los prototipos y esquemas de los diferentes módulos. Por un lado se esquematiza el modelo y relaciones de la base de datos, se identifican los componentes necesarios en la capa vista y se estructura el *back-end* para acoger la arquitectura elegida de la aplicación.
- *Sprint* 2. Integración de las bases de la aplicación en los diferentes módulos definidas en el sprint anterior, esto permite acceder a la *url* de la aplicación y comprobar el correcto funcionamiento, conectando exitosamente todos los módulos entre si.
- *Sprint* 3. Se implementa la seguridad en la aplicación.
- *Sprint* 4. Desarrollo del componente principal de la capa vista e implementación de bibliotecas externas en la capa *front-end* de entre las cuales destaca *Redux* por su importancia y peso en el desarrollo.
- *Sprint* 5. Se integran los diferentes juegos con su respectivo foro en la aplicación.
- *Sprint* 6. Se añade la funcionalidad para poder mandar mensajes a los diferentes foros.
- *Sprint* 7. Se incluyen los componentes y su respectiva funcionalidad para permitir a los usuarios suscribirse y darse de baja en los diferentes juegos.
- *Sprint* 8. Se implementa la funcionalidad de los chats.
- *Sprint* 9. Validaciones y despliegue.

En conclusión, la implementación de la metodología *Scrum* ha sido fundamental para el éxito del proyecto. La estructura basada en *sprints* ha permitido una organización clara y un progreso constante, facilitando la inspección regular y la adaptación a los cambios. Gracias a la transparencia y revisión continua, se ha logrado cumplir con los objetivos y entregar un producto de calidad. Esta experiencia reafirma el valor de *Scrum* como un marco eficiente y adaptable para la gestión de proyectos complejos.

CAPÍTULO 4
Análisis de Requisitos

El análisis de requisitos es una etapa esencial en el desarrollo de cualquier aplicación, ya que establece las bases para la construcción de un sistema que cumpla con las expectativas y necesidades de los usuarios finales. Este capítulo se enfoca en la identificación y definición de los requisitos específicos para la aplicación web desarrollada en este proyecto. A través de cuestionarios y entrevistas, se han recopilado datos importantes que guiarán el diseño y desarrollo de la aplicación. En este capitulo, se detallarán los métodos empleados para obtener una comprensión clara de las expectativas del público objetivo, así como los resultados obtenidos y cómo estos han influido en la concepción del proyecto. El objetivo es asegurar que la aplicación no solo sea funcional, sino que también resuene con los intereses y preferencias de sus futuros usuarios, garantizando así una solución efectiva y satisfactoria.

4.1 Captura de requisitos

En esta sección se tratará la captura de requisitos, cuyo objetivo es obtener un conjunto preciso y detallado de funcionalidades y restricciones que el sistema debe cumplir para alcanzar los objetivos establecidos. Este proceso se ha llevado a cabo en las siguientes fases:

- **Identificación del Cliente o Usuario final**. La aplicación web desarrollada en este proyecto no ha sido solicitada por un cliente específico; en cambio, se ha diseñado para un público general con un perfil particular. Los usuarios finales son personas que tienen como afición los videojuegos y buscan una plataforma que les permita interactuar con otros usuarios con sus mismos intereses, ya sea para formar equipos, comentar sobre algún juego o conocer a otros usuarios. El objetivo es crear una aplicación que satisfaga las necesidades de estos perfiles de usuario, proporcionando una experiencia que sea tanto funcional como atractiva.

- **Proceso de Recogida de Información**. El proceso de recogida de información es una etapa crucial en el desarrollo de la aplicación, ya que establece la base para definir sus requisitos. En este apartado se describe la metodología utilizada para recopilar dicha información, que se centró en la realización de entrevistas y cuestionarios a un grupo seleccionado de personas con interés en el proyecto. Estas interacciones permitieron identificar las necesidades, expectativas y posibles limitaciones, asegurando que el desarrollo de la aplicación esté alineado con los objetivos y preferencias de los usuarios finales.

  Las conclusiones extraídas de las preguntas del cuestionario han sido:

P: ¿Cuál debería ser el objetivo principal de la aplicación?

C: Los interesados coinciden en que el propósito principal debería ser proporcionar una plataforma en la que poder relacionarse. Se puede hacer una división entre los usuarios que acostumbran a jugar videojuegos *multijugador*, que se inclinan más hacia enfocar la aplicación en la creación de equipos y salas de chat. Por otro lado los usuarios que suelen jugar videojuegos individuales, prefieren enfocar la aplicación en la creación de foros en los que debatir y compartir sobre los diferentes juegos.

P: ¿Qué plataforma sueles utilizar?

C: El grupo seleccionado tiende a utilizar diversas plataformas para jugar. Filtrando por cual es la plataforma principal, un 70 % juegan en ordenador, 20 % en dispositivos móviles y un 10 % en consolas.

P: ¿Qué características debería tener la interfaz?

C: Las respuestas obtenidas sugieren que la interfaz debe ser intuitiva, con un diseño simple y ordenado que permita una navegación fácil. Los usuarios valoran una disposición clara de las funciones principales, con menús y botones que sean fáciles de localizar y utilizar.

P: ¿Qué grado de importancia le atribuyes a la privacidad?

C: Los usuarios coinciden en que la privacidad es de máxima importancia. Consideran esencial que la aplicación garantice la protección de sus datos personales y confidenciales, destacando la necesidad de contar con medidas de seguridad robustas.

P: ¿Tolerarías publicidad?

C: Los usuarios expresaron que, aunque podrían tolerar la presencia de publicidad si es discreta y no intrusiva, prefieren que la aplicación esté libre de anuncios. La mayoría opina que una experiencia sin publicidad mejora significativamente la usabilidad y la satisfacción general.

4.2 Requisitos iniciales

El apartado de requisitos iniciales presenta una lista de las necesidades y expectativas identificadas para el desarrollo de la aplicación. Estos requisitos, recopilados en las etapas iniciales del proyecto, sirven como base para guiar el diseño y la implementación del sistema. A través de un análisis preliminar de las necesidades de los usuarios y las metas del proyecto, se definen los aspectos fundamentales que deben ser considerados para asegurar que la aplicación cumpla con sus objetivos funcionales y operativos desde el inicio. Los requisitos fijados son los siguientes:

- **Registrar un nuevo usuario**. Permitir que los nuevos usuarios creen una cuenta proporcionando su nombre de usuario, contraseña y correo electrónico. El nombre de usuario y correo proporcionados no pueden coincidir con otros ya existentes.
- **Acceso de usuario**. Permitir a los usuarios registrados iniciar sesión en la aplicación usando sus credenciales. Nombre de usuario y contraseña en este caso.
- **Cerrar sesión**. Ofrecer la opción de salir de la cuenta de manera segura cuando el usuario lo desee.

- **Seleccionar juegos**. Permitir a los usuarios elegir y añadir juegos a su lista o biblioteca personal dentro de la aplicación. Esto se podrá hacer desde una ventana en la que se mostrarán los juegos disponibles, pudiendo filtrar por nombre.
- **Eliminar juegos**. Facilitar que los usuarios eliminen juegos de su lista o biblioteca personal cuando ya no los deseen.
- **Crear publicaciones**. Habilitar a los usuarios la posibilidad para crear y compartir publicaciones. Estas publicaciones podrán ser de tres tipos: *post*, *post* con invitación a otro usuario para unirse a un chat privado o *post* con invitación a otros usuarios para unirse a un grupo con un numero de miembros fijado.
- **Unirse a un chat**. Permitir a los usuarios unirse a un chat privado o a un grupo mediante un Post con invitación creado por otro usuario. En el *post* aparecerá un botón para unirse al chat o grupo en cuestión.
- **Enviar mensajes dentro de un chat**. Los usuarios podrán enviar y recibir mensajes de texto dentro de cada chat específico.
- **Salir de un chat**. Opción de abandonar un chat cuando no se desee participar en este.
- **Consultar participantes de un chat**. Permitir a los usuarios ver la lista de participantes en un chat específico para identificar a los miembros activos en la conversación.

4.3 Casos de uso y escenarios

El apartado de casos de uso se centra en describir cómo los usuarios interactuarán con la aplicación en distintas situaciones. A través de estos casos de uso, se exploran los escenarios en los que los usuarios realizarán tareas específicas, detallando los pasos que seguirán y la manera en que el sistema deberá responder. Este análisis es esencial para verificar que la aplicación cumpla con los requisitos funcionales y cubra de manera efectiva las necesidades y expectativas de los usuarios en las diversas funcionalidades.

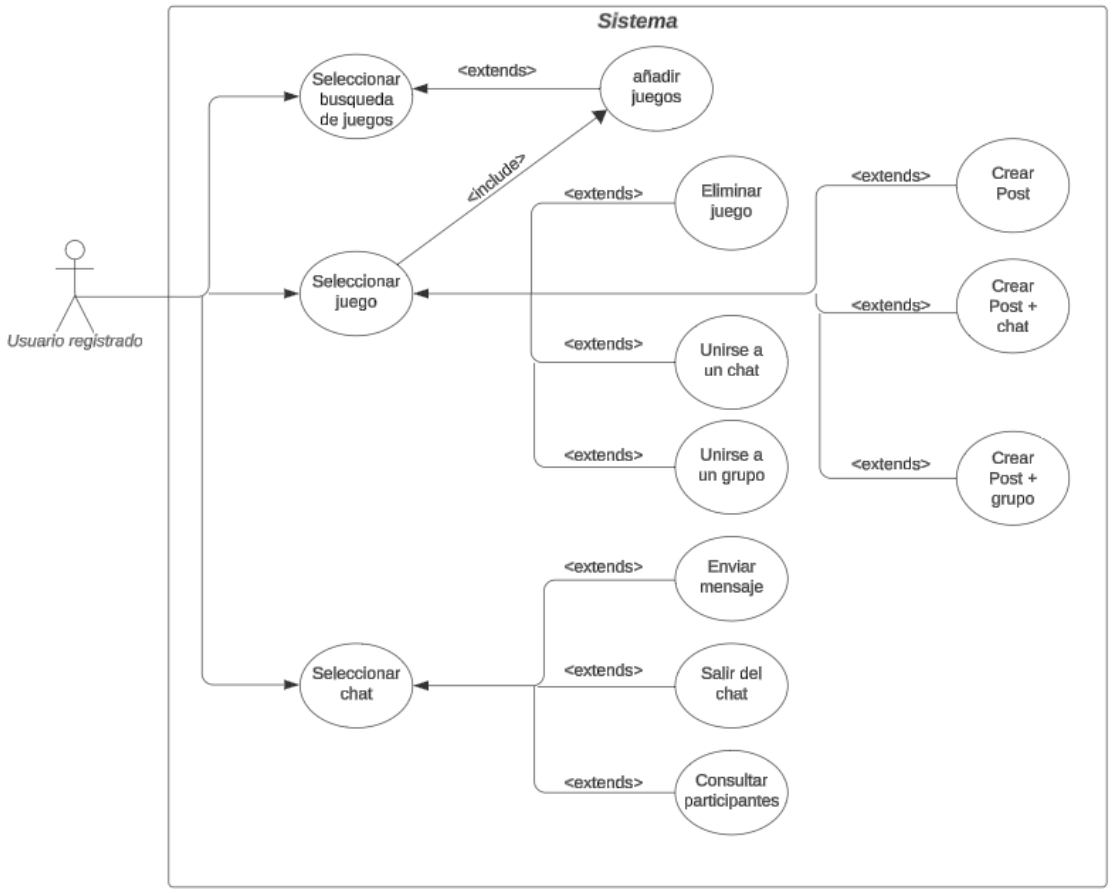

Figura 4.1: Diagrama de casos de uso.

Descripción	**Seleccionar búsqueda de juegos**
Entradas	▪ El sistema muestra los juegos del usuario, entre los cuales siempre estará la opción de "Búsqueda". A traves de esta opción se muestran los juegos disponbiles.
Procesos	▪ Se solicita al servidor la lista de juegos disponibles. Se eliminan de esta lista los juegos que el usuario ya posea en su biblioteca.
Salidas	▪ Se muestra una ventana con los juegos disponibles. ▪ En caso de error, se muestra un *toast* de error indicando que ha sucedido un problema no controlado.

Tabla 4.1: Escenario de Seleccionar búsqueda de juegos.

Descripción	**Añadir juego**
Entradas	▪ Desde el buscador de juegos se muestran los juegos disponibles. El usuario selecciona los que quiere añadir.
Procesos	▪ Se añaden los juegos seleccionados a la biblioteca del usuario
Salidas	▪ Se actualiza la la lista de juegos del usuario.

Tabla 4.2: Escenario de Añadir juego.

Descripción	**Seleccionar juego**
Entradas	▪ El sistema muestra los juegos de la biblioteca del usuario. El usuario puede seleccionar alguno de ellos para navegar a su foro.
Procesos	▪ Se actualiza el estado del cliente con el juego seleccionado. ▪ Se solicita al servidor las publicaciones del foro correspondiente.
Salidas	▪ Se actualiza la interfaz. ▪ Caso especial en el que el foro aún no tenga ningún mensaje. Se muestra un *toast* para avisar al usuario. ▪ En caso de error, se muestra un *toast* de error indicando que ha sucedido un problema no controlado.

Tabla 4.3: Escenario de Seleccionar búsqueda de juegos.

Descripción	**Eliminar juego**
Entradas	▪ Con un juego seleccionado y su correspondiente foro activo, el usuario pulsa el botón para eliminar el juego de su lista.
Procesos	▪ Se enviar al servidor la petición para eliminar el juego de la biblioteca del usuario. Se actualiza el estado en el cliente.
Salidas	▪ Se actualiza la información de la biblioteca de juegos del usuario para no mostrar el juego que se acaba de eliminar ▪ En caso de error se muestra un *toast* indicando que se ha producido un fallo.

Tabla 4.4: Escenario de Eliminar juego.

Descripción	**Crear Post**
Entradas	▪ Con un juego seleccionado y su correspondiente foro activo, el usuario puede crear un post para el foro en cuestión.. ▪ Se mostrará una ventana para escribir el post. El usuario tendrá la opción para añadir invitación a un chat o grupo si lo desea.
Procesos	▪ Se envía al servidor la petición para crear una nueva publicación en el foro correspondiente y con los datos introducidos por el usuario.
Salidas	▪ Se actualiza la información del foro con la nueva publicación. Se muestra un *toast* de éxito. ▪ En caso de error se muestra un *toast* indicando que se ha producido un fallo.

Tabla 4.5: Escenario de Crear Post.

Descripción	**Crear Post + chat**
Entradas	▪ Siguiendo los mismos pasos que en el escenario **Crear Post** 4.5, pero especificando que el post tendrá una invitación a chat.
Procesos	▪ Mismo comportamiento que que el escenario **Crear Post** 4.5.
Salidas	▪ Mismo comportamiento que el escenario **Crear Post** 4.5.

Tabla 4.6: Escenario de Crear Post + chat.

Descripción	**Crear Post + grupo**
Entradas	▪ Siguiendo los mismos pasos que en el escenario **Crear Post** 4.5, pero especificando que el post tendrá una invitación a grupo.
Procesos	▪ Mismo comportamiento que que el escenario **Crear Post** 4.5.
Salidas	▪ Mismo comportamiento que el escenario **Crear Post** 4.5.

Tabla 4.7: Escenario de Crear Post + grupo.

Descripción	**Unirse a chat**
Entradas	▪ Con un foro activo (4.3), localizar una publicación que tenga invitación a chat y unirse.
Procesos	▪ Se comprueba que el usuario que se intenta unir no sea el mismo que el creador del *Post*. ▪ Si el chat no se ha creado, se crea con los dos participantes (creador y usuario que decide unirse). ▪ Si el chat ya existe se comprueba si hay huecos libres.
Salidas	▪ Si tiene éxito se muestra el nuevo chat en la lista de chats del usuario ▪ Si no tiene éxito se muestra un toast de error especificando el motivo (Ya completo, error del servidor, etc ...)

Tabla 4.8: Escenario de Unirse a chat.

Descripción	**Unirse a grupo**
Entradas	▪ Mismo comportamiento que el escenario **Unirse a chat** 4.8 pero uniéndose a un post que tenga invitación a un grupo.
Procesos	▪ Mismo comportamiento que el escenario **Unirse a chat** 4.8.
Salidas	▪ Mismo comportamiento que el escenario **Unirse a chat** 4.8.

Tabla 4.9: Escenario de Unirse a grupo.

Descripción	**Seleccionar chat**
Entradas	▪ El sistema mostrara una lista con los chats disponibles, si los hay. EL usuario podrá seleccionar un chat para entrar en este.
Procesos	▪ Se cambia la vista principal para mostrar la sala del chat correspondiente. ▪ Se solicita al servidor los últimos 50 mensajes del chat.
Salidas	▪ Se muestra la sala del chat en cuestión con los mensajes (últimos 50) de los participantes.

Tabla 4.10: Escenario de Seleccionar chat.

Descripción	**Enviar mensaje**
Entradas	▪ Partiendo del escenario **Seleccionar chat** 4.10, una vez dentro de la sala de chat, el sistema permite enviar un mensaje para el chat en cuestión.
Procesos	▪ Se lanza la petición al servidor para crear el mensaje en el chat correspondiente y que llegue al resto de participantes.
Salidas	▪ Se actualizan los mensajes del chat con el nuevo mensaje. Se muestra un *toast* de éxito. ▪ Se muestra un *toast* de error en caso de que se produzca uno.

Tabla 4.11: Escenario de Enviar mensaje.

Descripción	**Salir del chat**
Entradas	▪ Partiendo del escenario **Seleccionar chat** 4.10, una vez dentro de la sala de chat el sistema permitirá la opción de salir del chat y eliminarlo de la lista del usuario.
Procesos	▪ El usuario deja de formar parte del chat y se elimina de su lista de chats.
Salidas	▪ Se actualiza la vista del usuario para eliminar el chat en cuestión y se muestra un *toast* de éxito. ▪ Se muestra un *toast* de error en caso de que se produzca uno.

Tabla 4.12: Escenario de Salir del chat.

Descripción	**Consultar participantes**
Entradas	▪ Partiendo del escenario **Seleccionar chat** 4.10, una vez dentro de la sala de chat el sistema permitirá la opción de consultar los participantes del chat.
Procesos	▪ Se lanza la petición al servidor para consultar la lista de participantes del chat en cuestión.
Salidas	▪ Se muestra un toast de éxito con la lista de participantes del chat. ▪ Se muestra un *toast* de error en caso de que se produzca uno.

Tabla 4.13: Escenario de Consultar participantes.

CAPÍTULO 5
Análisis Conceptual y Diseño

En este capitulo se presenta el diagrama de clases que componen la aplicación, también se muestran los prototipos para las vistas del cliente y se explica el modelo y arquitectura de la base de datos.

5.1 Diagrama de clases del sistema

En esta sección se presenta el diagrama de clases del sistema, concretamente del servidor para las capas de servicio y *DAO's* ya que son las que concentran una mayor lógica y entramado de dependencias. Utilizando el lenguaje de modelado unificado (UML), el diagrama de clases muestra las distintas clases que componen el sistema, sus atributos y métodos, así como las relaciones entre ellas. Este modelo es esencial para comprender la organización del sistema, la jerarquía de clases, y cómo interactúan entre sí.

El diagrama de clases proporciona una visión clara de la lógica de negocio implementada en el servidor, destacando los componentes clave y las dependencias. Además, este diagrama es fundamental para la fase de desarrollo, ya que sirve como guía para los desarrolladores y facilita la comunicación entre los miembros del equipo.

A continuación se muestran la figura 5.1 donde se hace uso de la herramienta *LucitChart* [29] para crear un diagrama que representa las clases que contienen la información en el servidor.

Para el cliente se muestra una lista con los compopnentes necesarios para crear la interfaz. Esta lista de componentes se crea a partir de los primeros prototipos y diseños que se muestran en la sección **Bocetos de las interfaces** 5.3. Los componentes que conforman la interfaz de la aplicación son:

- **App**. Raíz del proyecto, contiene la lógica de redirecciones.
- **Login**. Página para el formulario de acceso.
- **Registro**. Página para el formulario de registro.
- **Home**. Última de las tres páginas y componente principal de la interfaz, contiene el resto de componentes o se deben acceder desde este.
- **Cabecera superior**. Contiene la imagen de perfil del usuario, el nombre de la aplicación y el botón para cerrar sesión
- **Cabecera inferior**. Contiene la lista de juegos del usuario, además de dos iconos que cumplen una función estética e informativa.

- **Tablón de Post**. Contiene las publicaciones del foro seleccionado.
- **Tablón de chats**. Contiene los chats del usuario.
- **Post**. Componente para representar cada publicación.
- **Chat**. Componente para representar cada chat.
- **Juego**. Componente para representar cada juego.
- **Buscador de juegos**. Ventana emergente que muestra la lista de juegos disponibles, permitiendo al usuario añadirlos a su lista.
- **Redactar *Post***. Ventana emergente que muestra en pantalla los elementos necesarios para crear un *Post*.
- **Sala de chat**. Sustituye el espacio designado al foro cuando hay un chat seleccionado en lugar de un juego. Contiene los mensajes del chat y permite crearlos.
- **Mensaje de chat**. Componente para representar cada mensaje del chat.

Estos componentes conforman la interfaz de la aplicación, se mostrará su representación y usabilidad en el apartado **Producto desarrollado** 7.

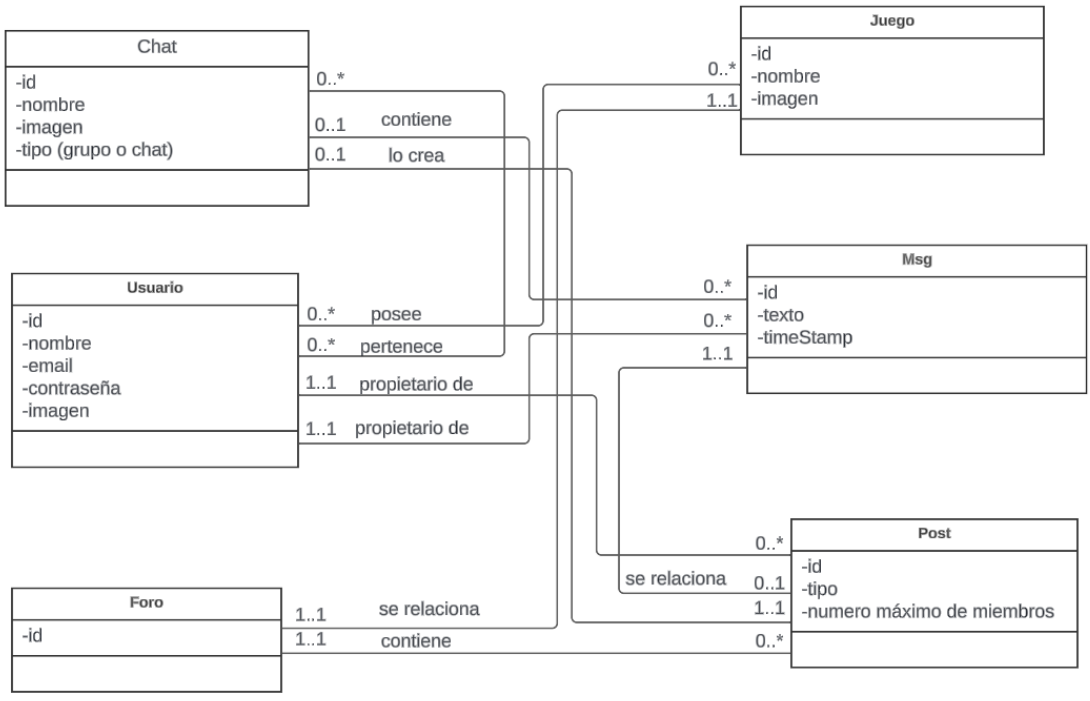

Figura 5.1: Diagrama UML de las entidades en java.

5.2 Modelo de base de datos

En este apartado se presenta el proceso de diseño del modelo de base de datos utilizado en la aplicación, destacando cómo se abordaron las fases de boceto y el esquema final. Desde el inicio, se optó por un enfoque relacional, utilizando *SQL* como lenguaje de gestión y apoyándose en *JPA (Java Persistence API) y Hibernate* para la capa de persistencia.

JPA (Java Persistence API) [35] es una especificación de Java que define cómo interactuar con bases de datos relacionales utilizando objetos Java, facilitando la persistencia de datos de una manera orientada a objetos. ***Hibernate*** [14] es una implementación de *JPA*, es decir, una herramienta concreta que cumple con la especificación de *JPA* y ofrece características adicionales. *Hibernate* simplifica las operaciones de base de datos al permitir que los desarrolladores trabajen directamente con objetos Java, en lugar de escribir consultas *SQL* manuales, gestionando automáticamente las relaciones entre las tablas, la gestión de transacciones y la conversión entre tipos de datos *SQL* y tipos de datos de Java. En resumen, mientras *JPA* establece el estándar, *Hibernate* es una de las implementaciones más populares y avanzadas de ese estándar.

Este enfoque permitió mapear eficientemente las entidades de la aplicación a la base de datos y gestionar de manera automatizada las relaciones entre ellas. Además, la base de datos se despliega en un contenedor, lo que garantiza un entorno aislado y consistente, simplificando la configuración y asegurando una integración y despliegue continuo más eficiente. Estos elementos clave permitieron no solo diseñar un modelo robusto y eficiente, sino también facilitar su implementación y escalabilidad. En el apartado **Ejemplos de código** 6.5 se muestra el código mas relevante para implementar este modelo de base de datos.

A continuación se muestran una lista de las relaciones entre las tablas y la figura 5.2 que ofrece un esquema gráfico del modelo final:

- **usuarios**. Tabla para almacenar la información de cada usuario. Tiene dos relaciones de muchos a muchos, en primer lugar con la tabla **chats**, para gestionar esta relación se crea la tabla intermedia **usuario-chat**. La relación es de este tipo porque un usuario puede tener tantos chats como desee y un chat puede tener de uno hasta ocho usuarios. La segunda relación muchos a muchos es con la tabla **juegos**, gestionada a través de la tabla intermedia **usuario-juego**, esto es porque cada usuario puede tener en su biblioteca tantos juegos como hayan disponibles al igual que un juego puede tener una cantidad indefinida de usuarios "suscritos" a él.

  Otras dos relaciones del tipo uno a muchos con las tablas **msgs** y **posts**. Esto es porque un usuario puede ser propietario de una cantidad indefinida tanto de mensajes como de *posts*.

- **chats**. Tabla para almacenar la información relativa a cada chat o grupo. Tiene una relación de muchos a muchos con la tabla **usuarios** comentada en el punto anterior, otro relación de uno a uno con la tabla **posts** para asociar a cada chat o grupo con el post que permite unirse a este. Por último una relación de uno a muchos con la tabla **msgs** ya que cada chat puede tener un numero ilimitado de mensajes.

- **msgs**. Tabla para almacenar la información referente a cada mensaje. Tiene dos relaciones uno a uno con la tabla **chats** y **posts**, esto es porque cada mensaje tiene que estar asociado a un chat o a un post. También tiene una relación uno a muchos con **usuarios** para relacionarlo con el usuario creador.

- **foros**. Tabla para asociar los *posts* al foro correspondiente de cada juego. La asociación *post-juego* se podría hacer directamente sin pasar por esta tabla, pero se mantiene cara a futuros desarrollos.
- **juegos**. Tabla para almacenar la información de cada elemento juego. Tiene una relación de uno a uno para asociarse con su foro correspondiente y una tabla intermedia ya mencionada en el primer punto para gestionar la relación de muchos a muchos con la tabla usuarios.
- **posts**. Esta tabla almacena todos los *posts* de los usuarios. Las relaciones que afectan a esta tabla están definidas en los puntos anteriores.

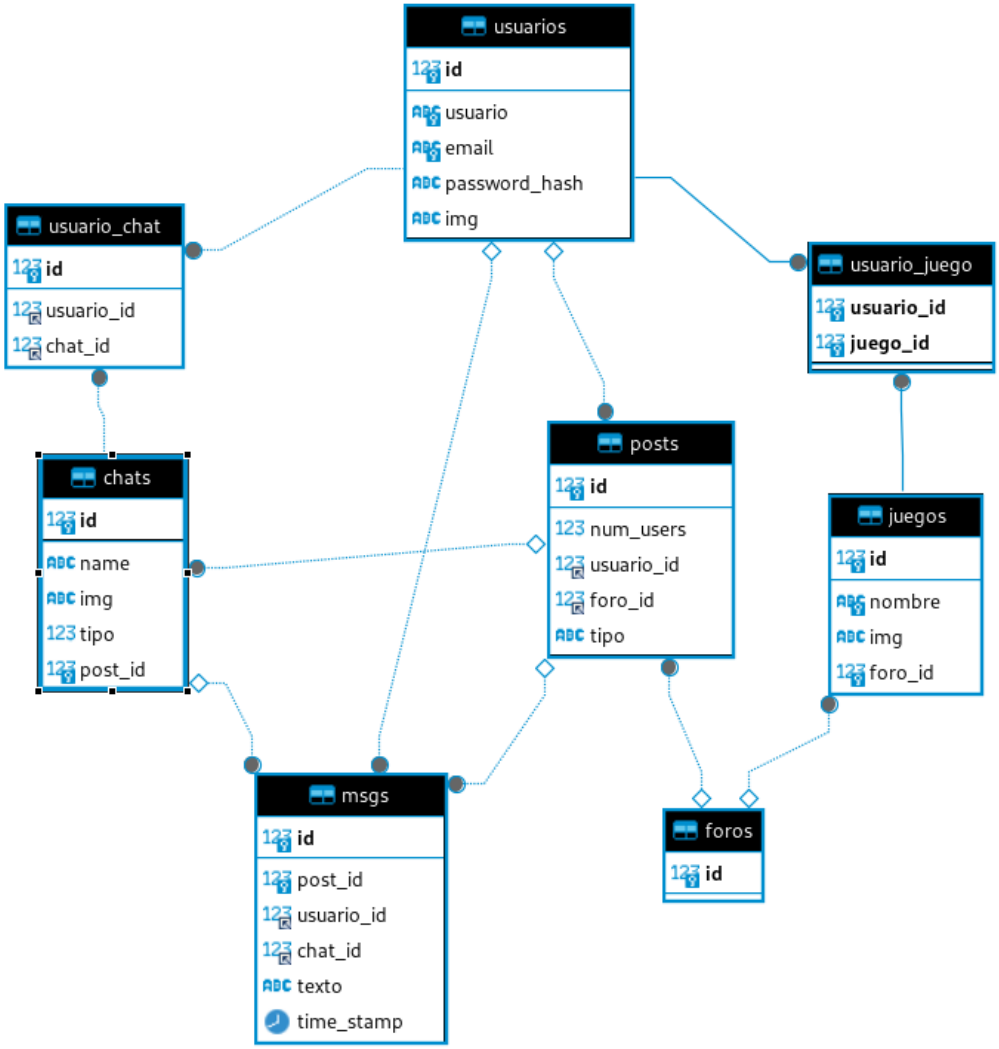

Figura 5.2: Modelo de la base de datos.

5.3 Bocetos de las interfaces

El diseño de la interfaz de la aplicación se llevó a cabo en dos fases clave. En la primera, se realizaron bocetos a mano en papel para conceptualizar la estructura básica, luego, en la segunda fase, estos bocetos se digitalizaron para perfeccionar los detalles visuales y funcionales, asegurando una experiencia de usuario coherente y atractiva.

- **Primeros bocetos sobre papel**. La fase inicial del proceso de diseño comenzó con la creación de los primeros bocetos sobre papel. Estos bocetos 5.3 se utilizaron como una herramienta fundamental para visualizar las ideas preliminares y explorar diferentes opciones de diseño de manera rápida y flexible. A través de este enfoque manual, se pudieron identificar las estructuras básicas de las interfaces y establecer un punto de partida para el desarrollo de prototipos más detallados, permitiendo así una primera iteración en la conceptualización del producto final.

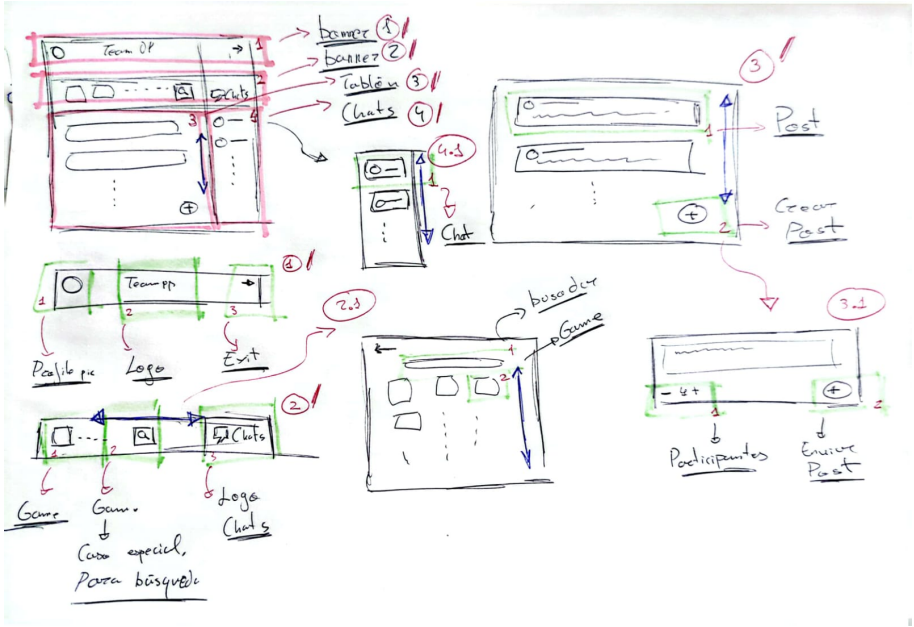

Figura 5.3: Boceto a papel 1.

- **Prototipo digital en *Figma***. En la fase de diseño digital en *Figma*, se trasladan los conceptos iniciales plasmados en los bocetos de papel a un entorno digital. Utilizando *Figma*, se crean prototipos de alta fidelidad que permiten una visualización más precisa y detallada de las interfaces, como se muestra en las figuras 5.4 y 5.5. Esta herramienta facilita la iteración rápida, lo que resulta clave para refinar el diseño, definir la paleta de colores, la tipografía y la disposición de los elementos en pantalla, asegurando que el diseño sea coherente y estéticamente atractivo.

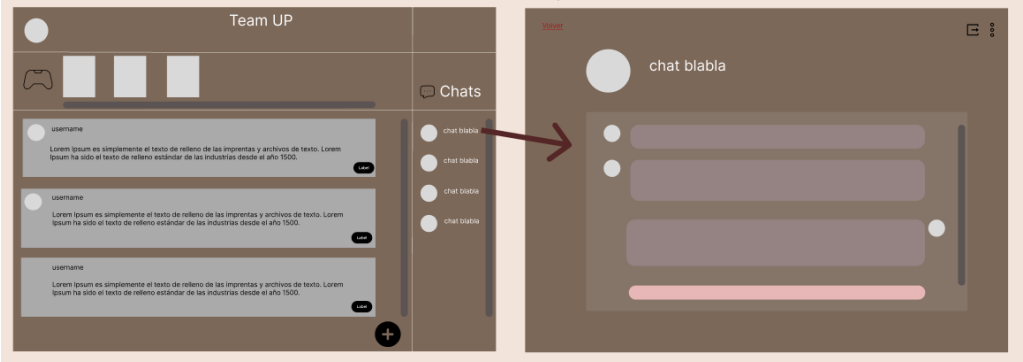

Figura 5.4: Boceto con *Figma* 1.

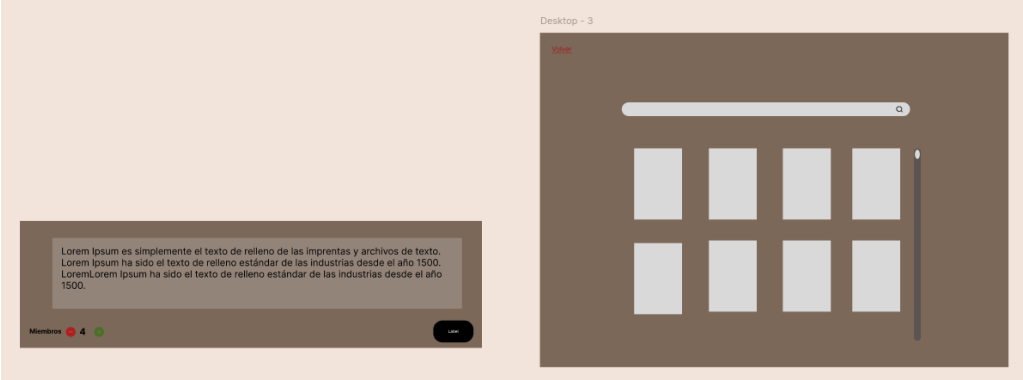

Figura 5.5: Boceto con *Figma* 2.

El prototipado con *Figma* ha sido fundamental para sintetizar y definir los componentes principales que integrarán la versión final de la aplicación. A través de este proceso iterativo, fue posible experimentar con diferentes diseños y flujos de usuario, obteniendo retroalimentación temprana que ayudó a refinar tanto la funcionalidad como la estética de la interfaz.

CAPÍTULO 6
Desarrollo de la solución

En este capitulo se aborda la implementación de los módulos servidor y cliente así como la arquitectura general que conecta toda la aplicación. También se hace un repaso de las tecnologías con mayor relevancia durante el desarrollo del proyecto y se muestran y explican algunos de los bloques de código con mayor relevancia en la aplicación.

6.1 Arquitectura general de la aplicación

En esta sección se explica la conexión de las diferentes partes que componen la aplicación y como se conectan entre ellas. La aplicación se divide en tres módulos o capas principales principales: el cliente o *front-end*, el servidor o *back-end* y por último la base de datos.

A continuación, se pasa a explicar como se conectan y comunican los diferentes módulos entre sí.

Para la **comunicación entre el cliente *(front-end)* y el servidor *(back-end)***, se ha utilizado la librería *Axios* instalada mediante *NPM* [27]. *Axios* es una popular librería de *JavaScript* que facilita la realización de solicitudes *HTTP* desde el navegador hacia el servidor. Esta herramienta permite enviar y recibir datos en formato *JSON* [28], gestionar errores de manera sencilla y manejar respuestas síncronas, lo que la hace ideal para interactuar con *APIs RESTful*.

En este caso, la comunicación sigue el estándar *REST (Representational State Transfer)* [26], que es un estilo arquitectónico que utiliza métodos *HTTP* como *GET, POST, PUT y DELETE* para operar sobre los recursos en el servidor. A través de *Axios*, las solicitudes se envían utilizando *HTTP*, y los datos se intercambian en formato *JSON*, lo que asegura una comunicación eficiente y clara entre el cliente y el servidor.

Por otro lado, **la base de datos** *MySQL* se aloja dentro de un contenedor, lo que ofrece un entorno controlado y reproducible, asegurando que la base de datos esté aislada y fácilmente gestionable. Para establecer la conexión entre el servidor *Spring Boot* y esta base de datos, se configura un *bean* de tipo *DataSource*. Este *bean* define los parámetros de conexión, como el driver *JDBC* de *MySQL*, la *url* del servidor de base de datos, y las credenciales de acceso. El código de estas partes se muestra en la sección **Ejemplos de código** 6.5.

En último lugar la **implementación de la seguridad** en la aplicación. Por un lado las contraseñas de los usuarios se guardan utilizando un algoritmo de *hasing* proporcionado por la biblioteca *Bcrypt*, que incluye una *salt* aleatoria para robustecer la seguridad del encriptado. De esta manera se garantiza que en el hipotético caso de que un atacante

consiguiera acceso a la base de datos, no pudiese obtener acceso a la cuenta de ningún usuario.

El segundo aspecto fundamental en cuanto a la seguridad es la comprobación de autorización en todas las llamadas que se realizan al servidor. Para ello se hace uso de un *JWT (JSON web token)* [31]. Este proceso comienza por el lado del servidor, importamos la librería *auth0* desde el *POM (Project Object Model)* de *Maven*. El segundo paso es proteger las rutas de la *API*, para ello se debe configurar *Spring Security*. Se crea una clase de configuración que extiende de *WebSecurityConfigurerAdapter*. En esta clase, se configuran las reglas de seguridad, como permitir el acceso sin autenticar a ciertas rutas (por ejemplo, la ruta de login o registro) y requerir autenticación para el resto de rutas.

En esta configuración también se define que la aplicación no mantendrá sesiones de usuario, ya que *JWT* es *"stateless"* (sin estado). Además, se añade un filtro que se encargará de interceptar las peticiones y validar el token *JWT*.

Posteriormente, cuando un usuario inicia sesión, el *Backend* valida las credenciales y, si son correctas, genera un token *JWT*. Este token contiene información codificada sobre el usuario, su nombre en este caso y una fecha de expiración que en la aplicación se configura con una duración de una hora. El token se firma con una clave secreta para garantizar que no pueda ser modificado. Luego, se envía este token de vuelta al cliente como parte de la respuesta de autenticación.

Desde el cliente, al realizar un registro o *login* exitoso, el servidor responde con un token *JWT*, este token se guarda en el almacenamiento local del navegador *localStorage*. Este token se usará para autenticar futuras peticiones. En cada petición subsiguiente a la *API* que requiera autenticación, *Axios* incluye el token *JWT* en los encabezados de la petición *HTTP*. Esto se hace en el encabezado *Authorization* seguido del prefijo *Bearer* y el token.

Si el token es válido, el *Backend* permitirá el acceso a los recursos protegidos; de lo contrario o si ha expirado, devolverá un error 401 *(Unauthorized)* y redigirá al usuario a la pantalla de *Login*. En la figura 6.1 se muestra un esquema de la arquitectura global de la aplicación.

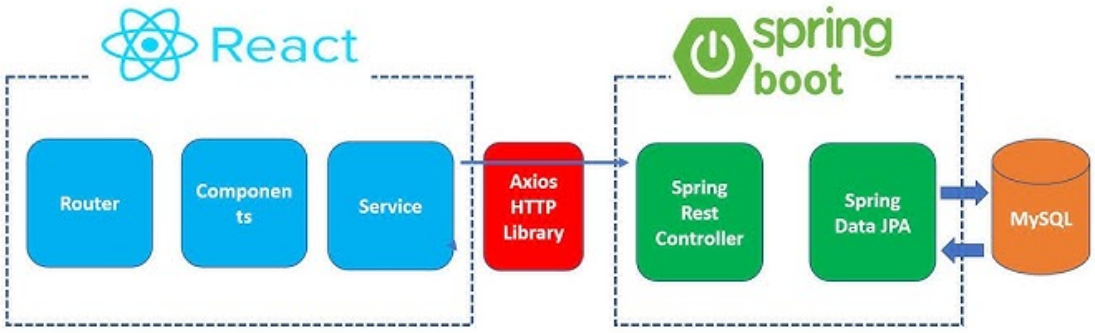

Figura 6.1: Esquema de arquitectura general.

Fuente:https://www.youtube.com/watch?v=iK__liBIXWk

6.2 Arquitectura y diseño del cliente

En esta sección se aborda el análisis conceptual y el proceso de diseño del cliente o *front-end*. Este módulo se ha desarrollado utilizando ***React*** [11] , una biblioteca de *JavaScript* que permite un enfoque basado en componentes. Este enfoque facilita la creación de interfaces de usuario modulares y reutilizables, asegurando una mayor flexibilidad y mantenibilidad en el diseño de la capa de vista. Cada componente se ha diseñado para manejar una parte específica de la interfaz, creando así una experiencia de usuario fluida y eficiente.

Un aspecto clave en el diseño de la interfaz de la aplicación ha sido su enfoque como ***Single Page Application (SPA)*** [12]. Este modelo, esquematizado en la figura 6.2, ofrece una experiencia de usuario más fluida y rápida, ya que, en lugar de recargar la página completa al navegar entre secciones, solo se actualizan las partes necesarias del contenido. Las *SPA* se usan para gestionar el estado y la lógica de la interfaz, permitiendo una navegación similar a la de una aplicación nativa. Además, al minimizar las solicitudes al servidor, mejoran el rendimiento y ofrecen una mayor capacidad de respuesta.

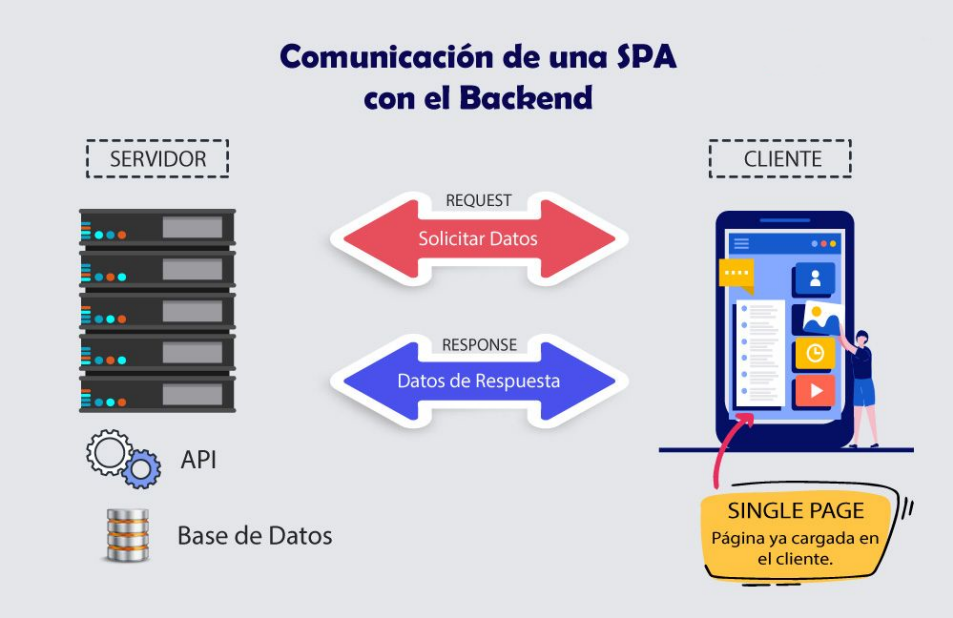

Figura 6.2: Funcionamiento de una *SPA*.

Fuente:https://blog.codmind.com/como-funciona-react/

Otro elemento principal a la hora de construir este módulo ha sido el uso de la biblioteca ***Redux*** [13]. *Redux* es una biblioteca utilizada para la gestión del estado en aplicaciones *JavaScript* que proporciona un almacén centralizado para mantener toda la información de la aplicación. Con este enfoque, se facilita la sincronización y consistencia del estado global, lo cual resulta esencial en aplicaciones complejas que requieren una gestión del estado robusta y escalable. Al seguir el patrón de arquitectura *Flux* 6.3, se asegura que todos los componentes de la aplicación accedan al estado global de manera predecible y ordenada.

A su vez, también se utiliza para manejar de manera eficiente el estado global, incluyendo la información de cada usuario, como sus juegos, publicaciones del foro activo y el historial de mensajes del chat activo.

El uso de *Redux* también ha facilitado la implementación de características avanzadas, como la persistencia del estado a través de sesiones. Mediante la integración de *middle-*

ware como *redux-persist*, los datos importantes del usuario pueden ser guardados y restaurados incluso después de que la aplicación se haya recargado. Esto no solo mejora la experiencia del usuario al mantener su progreso y preferencias, sino que también optimiza el rendimiento de la aplicación al reducir la necesidad de recargar información desde el servidor cada vez que se inicia.

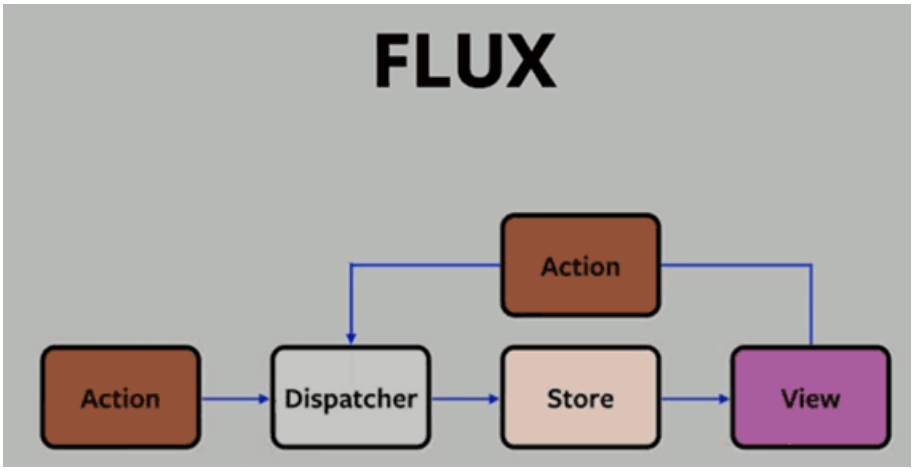

Figura 6.3: Funcionamiento de *redux*.

Fuente:https://desarrolloweb.com/articulos/que-es-redux.html

6.3 Arquitectura y diseño del servidor

En esta sección se detalla el funcionamiento del módulo *back-end* o servidor que se ha implementado para la aplicación, así como su funcionamiento y conexión entre el resto de módulos. El servidor se ha desarrollado en *Java* [22] utilizando el marco de trabajo que proporciona ***Spring Boot*** [15] una extensión de *Spring Framework*, algunas de las características principales de este *framework* y su importancia en el proyecto son:

- **Generación de aplicaciones independientes**. *Spring Boot* permite la creación de aplicaciones que pueden ejecutarse de manera autónoma, sin necesidad de servidores externos. Esto se logra gracias a los contenedores embebidos como *Tomcat* [24], que está integrado directamente en la aplicación, permitiendo su ejecución inmediata con un simple comando. Esta característica ha sido esencial durante el proceso de desarrollo, pudiendo ejecutar la aplicación de manera rápida y sencilla en esta fase para validar los constantes cambios y adiciones que se realizan en esta.

- **Configuraciones automáticas**. Uno de los mayores beneficios de *Spring Boot* es su capacidad de realizar configuraciones automáticas. En lugar de configurar manualmente cada aspecto del entorno de desarrollo, *Spring Boot* detecta automáticamente las dependencias y configura el entorno según las mejores prácticas. Además se ha utilizado *Maven*, un gestor de dependencias y automatización de proyectos, esto ha permitido agilizar el proceso de desarrollo ya que *Maven* [25] maneja las dependencias y *Spring Boot* aplica las configuraciones optimizadas.

- **Soporte para *RESTful APIs***. *Spring Boot* ofrece soporte nativo para el desarrollo de servicios *RESTful*, facilitando la creación y gestión de *APIs* web que pueden

ser consumidas por diversas aplicaciones y servicios. Esto incluye herramientas para el enrutamiento de solicitudes, manejo de datos y validación de entrada. Esta característica se ha utilizado para disponer las diferentes direcciones a las que el modulo *front-rnd* manda peticiones para obtener información.

- **Integración con *Spring Security***. El *framework* se integra a la perfección con *Spring Security*, proporcionando características de autenticación y autorización. Esto permite proteger las aplicaciones mediante mecanismos avanzados de seguridad, como la autenticación al acceder a los recursos de la *API* y la protección contra ataques comunes. En el desarrollo de la aplicación se ha utilizado esta característica para poder definir filtros y reglas de seguridad específicos.

La arquitectura del servidor se ha estructurado con un enfoque por capas [16]. Este es un patrón de diseño de software que organiza una aplicación en distintas capas, cada una con responsabilidades específicas y bien definidas, como se muestra en la figura 6.4. Esta estructura modular facilita el desarrollo, mantenimiento y escalabilidad de las aplicaciones al permitir que cada capa se enfoque en una parte particular del proceso. Las capas en las que se ha divido el servidor son las siguientes:

- **Capa de Controladores**. Esta capa se encarga de manejar las solicitudes que provienen del *front-end*. Los controladores actúan como intermediarios entre la capa de presentación y la capa de servicios. Su responsabilidad es recibir las solicitudes, delegar el procesamiento a los servicios adecuados y devolver las respuestas correspondientes.

- **Capa de Servicios**. La capa de servicios contiene la lógica de negocio de la aplicación. Los servicios reciben las solicitudes de los controladores y ejecutan las reglas de negocio necesarias. Aquí se implementan las operaciones y procesos que no están directamente relacionados con la interfaz de usuario o la persistencia de datos, sino que manejan la lógica específica de la aplicación. La capa de servicios actúa como un intermediario entre los controladores y los *DAOs*, procesando datos y coordinando las acciones necesarias.

- **Capa de DAOs (Data Access Objects)**. Esta capa es responsable de la interacción directa con la base de datos. Los *DAOs* realizan las operaciones de *CRUD* (Crear, Leer, Actualizar, Eliminar) y abstraen el acceso a los datos, encapsulando las complejidades de las consultas a la base de datos. Los servicios llaman a los *DAOs* para obtener o almacenar datos, separando así la lógica de acceso a datos de la lógica de negocio.

- **Capa de infraestructura**. Esta capa proporciona los servicios transversales necesarios para el funcionamiento de la aplicación. Incluye la configuración de seguridad, la conexión a la base de datos, la gestión de archivos, y otras funcionalidades que soportan las operaciones de las capas superiores. La capa de infraestructura se encarga de aspectos como la autenticación y autorización, la integración con servicios externos y la configuración de recursos de red.

6.4 Contexto tecnológico

En esta sección, se presenta una lista donde se detallan las herramientas y tecnologías empleadas en el desarrollo de la aplicación. Este análisis abarca tanto el software esencial

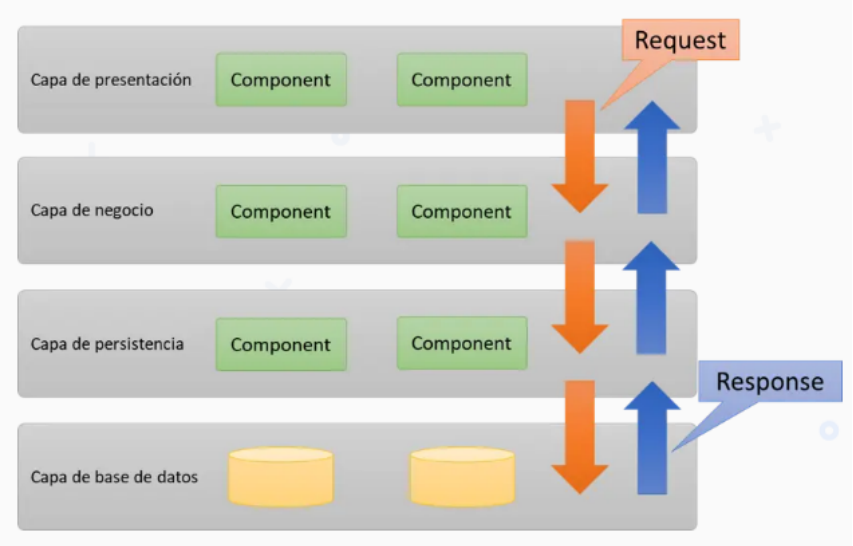

Figura 6.4: Esquema de arquitectura por capas.

Fuente:https://reactiveprogramming.io/blog/es/estilos-arquitectonicos/capas

para la codificación y desarrollo, como herramientas y aplicaciones auxiliares igualmente necesarias durante la fase de desarrollo. Se divide el uso de herramientas y software en tres grupos, el primero orientado al desarrollo de software y al código, el segundo para el sistema e infraestructura, y por último herramientas para recursos y ofimática.

En el primer grupo de software y herramientas para el desarrollo y el código han sido fundamentales las siguientes tecnologías:

- **Eclipse** [30] . Eclipse ha sido el editor de código elegido para desarrollar la parte servidor de la aplicación debido a su amplio número de funcionalidades y versatilidad. La capacidad de *Eclipse* para integrarse con diversos *plugins* y herramientas permite una gran personalización del entorno de desarrollo, adaptándose así a las necesidades específicas del proyecto. Además, sus funcionalidades avanzadas, como el auto completado de código y la navegación entre archivos, contribuyen a una codificación más rápida y menos propensa a errores.

  Por otro lado, Eclipse también ofrece potentes herramientas de depuración y análisis de rendimiento, que son cruciales durante fases de *debugging*. Su integración con sistemas de control de versiones, como *Git* [32], facilita el trabajo entre versiones y el mantenimiento del código, al tiempo que permite un seguimiento detallado de los cambios realizados.

- **Visual Studio Code** [33] . *Visual Studio Code (VSCode)* ha sido el editor de código seleccionado para el desarrollo de la parte cliente de la aplicación. Este entorno de desarrollo destaca por su flexibilidad y por la amplia variedad de extensiones que ofrece, adaptándose perfectamente a las necesidades del desarrollo en *JavaScript y React*. *VSCode* proporciona un entorno ligero, sencillo y muy configurable, ideal para gestionar la estructura de los componentes y el estado de la aplicación de manera eficiente.

  También permite integrar *Git*, lo que simplifica el seguimiento de los cambios en el código y el trabajo entre versiones. Su soporte para el desarrollo *front-end*, junto con las extensiones específicas para *React*, permite una gestión eficaz del código y una mejor organización del proyecto.

- **Java [22] y Spring Boot [15]** . *Java 17 y Spring Boot* han sido las tecnologías troncales empleadas en el desarrollo de la parte servidor de la aplicación. *Java* 17 es la última versión, ofrece mejoras significativas en términos de rendimiento y estabilidad, además de nuevas características que facilitan el desarrollo de aplicaciones punteras y seguras. Permite aprovechar las últimas innovaciones tanto en el lenguaje como en mejoras respecto al manejo de excepciones, además una mayor eficiencia en la ejecución del código.

  Spring Boot, por su parte, complementa a *Java 17* al proporcionar un marco de trabajo que simplifica el desarrollo de aplicaciones basadas en Java. Con su enfoque en la productividad y la facilidad de configuración, *Spring Boot* permite una rápida creación de servicios y aplicaciones web mediante su configuración automática y su amplia gama de herramientas integradas. Esta combinación de *Java 17* y *Spring Boot* garantiza un entorno de desarrollo idóneo, optimizando el rendimiento del servidor y facilitando la implementación de una arquitectura escalable y mantenible.

- **Maven** [25] . *Maven* se ha utilizado para la gestión de dependencias y construcción de la parte servidor de la aplicación. *Maven* proporciona un marco estructurado y consistente para gestionar las dependencias, construir el proyecto y automatizar tareas repetitivas, lo que simplifica significativamente el proceso de desarrollo. Su sistema de gestión de dependencias permite integrar fácilmente bibliotecas y componentes externos, asegurando que el proyecto se mantenga actualizado y libre de conflictos. Además, *Maven* facilita la configuración y el despliegue del proyecto a través de su sistema de construcción basado en un archivo de configuración centralizado, el *POM*, permitiendo los despliegues de manera automatizada optimizando en gran medida el desarrollo.

- **Hibernate** [14] . Software utilizado para la gestión de la persistencia de datos en el desarrollo de la parte servidor de la aplicación. Como una de las bibliotecas de mapeo objeto-relacional *(ORM)* más populares para Java, *Hibernate* facilita la interacción entre la aplicación y la base de datos al permitir la conversión automática entre objetos Java y registros de base de datos. Su capacidad para gestionar transacciones, realizar consultas complejas y optimizar el rendimiento de las operaciones de base de datos simplifica el desarrollo de aplicaciones eficientes y escalables. *Hibernate* no solo mejora la eficiencia en la gestión de datos, sino que también proporciona una capa de abstracción que reduce la necesidad de escribir código SQL manual, aumentando la productividad del desarrollador.

- **Lombok** [34] . Librería que simplifica la escritura de código al generar automáticamente métodos comunes como *getters, setters* y constructores entre otros. Esto no solo acelera el proceso de desarrollo, sino que también mejora la legibilidad y el mantenimiento del código al reducir la cantidad de código repetitivo y redundante. La integración de *Lombok* facilita la creación de clases más limpias y legibles, optimizando la productividad y la calidad del código en el proyecto.

- **Auth0, Spring-framework-security [38] y Jbcrypt [37]**. Estas librerías se han utilizado para garantizar la seguridad en la aplicación. *Auth0* junto con *Spring-framework-security* para proteger las direcciones la *API* y requerir autenticación en las llamadas. *Jbcrypt* para encriptar los datos sensibles.

- **JavaScript [23] y React [11]**. Tecnologías principales para el desarrollo de la parte cliente de la aplicación. *JavaScript*, como el lenguaje de programación principal para el desarrollo web, permite crear una experiencia interactiva y dinámica en el navegador, facilitando la manipulación del *DOM* y la implementación de lógica de cliente.

React, por su parte, es una biblioteca de *JavaScript* que se ha convertido en estándar para construir interfaces de usuario interactivas y eficientes. Desarrollada por *Facebook*, *React* permite crear componentes reutilizables y gestionar el estado de la aplicación de manera efectiva, lo que simplifica el desarrollo de aplicaciones complejas y mejora el rendimiento de la interfaz. Su enfoque basado en componentes y su arquitectura unidireccional de flujo de datos facilitan la creación de aplicaciones web escalables y mantenibles.

- **JSX** [39] . *JSX* es una extensión de sintaxis para *JavaScript* que se utiliza en el desarrollo de aplicaciones con React, y ha sido crucial para el diseño de la interfaz de usuario de la aplicación. *JSX* permite escribir código que combina la lógica de *JavaScript* con una estructura similar a *HTML*, facilitando la creación y el manejo de componentes de manera más intuitiva. Este enfoque simplifica la construcción de interfaces al permitir la definición de la estructura de los componentes directamente dentro del código *JavaScript*, mejorando la legibilidad y el mantenimiento del código.

- **Redux** [13] . *Redux* ha sido una herramienta esencial para la gestión del estado en la parte cliente de la aplicación, proporcionando una solución para manejar datos y estados complejos el cliente. Como una biblioteca de gestión de estado basada en el patrón *Flux*, *Redux* centraliza el estado de la aplicación en un único almacén, lo que facilita el acceso y la sincronización de datos entre diferentes componentes. Esta centralización permite una gestión más predecible y ordenada del estado global, facilitando la depuración y el seguimiento de los cambios en la aplicación.

- **Toastify** [43]. *Toastify* es una biblioteca utilizada para mostrar notificaciones emergentes en la interfaz de usuario de la aplicación, proporcionando una forma sencilla y estilizada de informar a los usuarios sobre eventos o acciones importantes. Al integrar *Toastify*, se pueden mostrar mensajes de tipo *toast* que aparecen brevemente en la pantalla, sin interrumpir la experiencia del usuario, y se desvanecen automáticamente después de un período de tiempo.

Para el segundo grupo referente al sistema y la infraestructura se han utilizado las siguientes herramientas y tecnologías:

- **Fedora** [41] . El sistema donde se han realizado todas las fases de este proyecto. Ofrece un entorno moderno gracias a su enfoque en la innovación y la integración de tecnologías avanzadas. Como una distribución de Linux, *Fedora* proporciona un sistema operativo seguro y eficiente, con acceso a herramientas y paquetes a través de su gestor de paquetes *DNF*.

- **Podman** [40] . Software utilizado para gestionar el contenedor de la base de datos en el desarrollo de esta aplicación. Al utilizar *Podman*, se ha podido crear y administrar un contenedor aislado para la base de datos, lo que ha facilitado la configuración y el mantenimiento de un entorno de base de datos consistente y reproducible.

- **DBeaver** [36] . Se ha utilizado para la administración de la base de datos *MySQL* en este proyecto. Como una aplicación de administración de bases de datos de código abierto, *DBeaver* proporciona una interfaz gráfica intuitiva que facilita la conexión y gestión de *MySQL*. Permite realizar consultas, gestionar esquemas y analizar datos de manera eficiente, lo que ha sido esencial para asegurar el rendimiento y la integridad de la base de datos.

- **Github** [17]. *GitHub* ha sido esencial para el manejo del código fuente, utilizándose exclusivamente para el control de versiones y la gestión del desarrollo personal. Aunque no se ha usado en un contexto de equipo, *GitHub* ha facilitado la organización y el seguimiento de los cambios en el código, ofreciendo un historial detallado de versiones y permitiendo una gestión eficiente de las ramas de desarrollo. Su capacidad para realizar copias de seguridad del código y su integración con diversas herramientas de desarrollo han sido clave para mantener la calidad y coherencia del proyecto a lo largo de su evolución.

Por último, las herramientas destinadas a gestión de recursos y ofimática:

- **GIMP** [44] . *GIMP* se ha utilizado para preparar las imágenes tanto para la aplicación como para la memoria del proyecto. Esta herramienta de edición de imágenes de código abierto ha facilitado la modificación y el ajuste de las imágenes, asegurando que estén adecuadamente adaptadas para su uso en la aplicación y en la documentación del proyecto.
- **Overleaf** [42] . Overleaf ha sido utilizado para desarrollar la memoria del proyecto en LaTeX. Esta plataforma en línea ha proporcionado un entorno accesible para escribir, editar y compilar el documento, facilitando la gestión del formato y el contenido de manera eficiente. Con sus herramientas integradas y su capacidad para trabajar en tiempo real, Overleaf ha permitido mantener una estructura consistente y profesional en la memoria, optimizando el proceso de redacción y revisión del documento.

6.5 Ejemplos de código

En esta sección se van a mostrar algunos ejemplos de código que corresponden a puntos de especial interés de la aplicación. Los ejemplos mostrados equivalen a la fase de desarrollo, por motivos de seguridad no se mostrarán ejemplos con credenciales del código de producción.

El código de todo el proyecto es accesible para cualquier persona a través de los dos repositorios *Github* correspondientes.

- **Backend**. https://github.com/Alexcas16/TFG-TeamUP-back.git
- **FrontEnd**. https://github.com/Alexcas16/TFG-TeamUP-front.git

Empezando por la **base de datos**. Como se explica en la sección **Modelo de base de datos** 5.2, esta se aloja en un contenedor, por lo que para configurar la conexión entre el servidor y el *DataSource* se realiza la conexión en la clase *AppConfig* 6.1, que se anota con *@Configuration* para indica que la clase se va a utilizar como clase para definir uno o más *beans*, siendo un *bean* un objeto que forma parte del *Spring IoC (Inversion of Control) Container*, de forma muy resumida es un objeto gestionado por *Spring*. En esta clase se define un *bean* del tipo *Datasource*, que será utilizado como fuente de datos para la aplicación.

```java
@Configuration
public class AppConfig {

private static final Long MAX_AGE = 3600L;
private static final int CORS_FILTER_ORDER = -102;

@Bean
public DataSource datasource() {
    return (DataSource) DataSourceBuilder.create()
        .driverClassName("com.mysql.cj.jdbc.Driver")
        .url("jdbc:mysql://127.0.0.1:3306/TeamUP_DB")
        .username("xxxxx")
        .password("xxxxx")
        .build();
}
```

Listing 6.1: Conexión con la base de datos.

Siguiendo en la misma línea, las entidades se definen dentro del paquete *entities* y su función es representar los datos obtenido de la base de datos, para ello se definen las clases con la anotación *@Entity*, cada clase se corresponde con una tabla de la base de datos y las instancias de esta clase representa un registro de la tabla correspondiente. Con esta anotación indicamos que es una entidad *JPA*, un objeto que podemos persistir en la base de datos. La anotación *@Table* se utiliza para especificar la tabla de la base de datos a la que se asocia. Se muestra de ejemplo la clase *UsuarioEntity* 6.2.

```java
@Entity
@Getter @Setter
@Table(name = "usuarios")
public class UsuarioEntity {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(unique = true)
    private String usuario;

    private String email;

    private String password_hash;

    private String img;

    @ManyToMany
    @JoinTable(
        name = "usuario_juego",
        joinColumns = @JoinColumn(name = "usuario_id"),
        inverseJoinColumns = @JoinColumn(name = "juego_id")
    )
    @OrderBy("id ASC")
    private List<JuegoEntity> juegos_ids;
```

Listing 6.2: Ejemplo de entidad.

Para finalizar los ejemplos referentes a la base de datos, se muestra la configuración de una de las clases *DAO (Data Access Object)*. Estas se encuentran en el paquete *daos* y su

función es la de realizar todas las operaciones con la base de datos referentes a su entidad. En este caso se muestra *UsuarioDAOimpl* 6.3, clase encargada de realizar las operaciones *CRUD* para la entidad *UsuarioEntity* que se asocia a la tabla usuarios.

Las anotaciones más relevantes a la hora de configurar estas clases son:

1. *@Repository*. Marca la clase como un *Bean* pero concretando que se utilizará para acceso a datos.

2. *@Transactional*. Indica que los métodos de la clase deben ser ejecutados dentro de una transacción. La transacción es una bloque de operaciones que debe completarse por completo o en caso de error no persistir ningún cambio.

3. *@PersistenceContext*. Se utiliza para inyectar el *EntityManager* en un DAO. El *EntityManager* es una interfaz de *JPA (Java Persistence API)* que gestiona las operaciones de persistencia en la base de datos.

4. *@Autowired*. Es una forma de inyección de dependencias en *Spring*.

```
1   @Repository
2   @Transactional
3   public class UsuarioDAOimpl {
4
5       @PersistenceContext
6       EntityManager em;
7
8       @Autowired
9       private JuegoDAOimpl juegoDAO;
10
11      private static final Logger logger = (Logger) LoggerFactory
            .getLogger(UsuarioDAOimpl.class);
12
13      @Transactional(rollbackOn = TeamUPexception.class)
14      public void guardarUsuario(UsuarioEntity usuario) throws
            Exception {
15          try {
16              em.persist(usuario);
17          } catch (PersistenceException e2) {
18              logger.error("Error al intentar persistir el
                    usuario: " + usuario.getUsuario(), e2);
19
20              Throwable cause = e2.getCause();
21              if (cause instanceof java.sql.
                    SQLIntegrityConstraintViolationException) {
22                  String message = e2.getMessage();
23
24                  if (message.contains("usuarios.usuario")) { //
                        NOMBRE DUPLICADO
25                      throw new TeamUPexception("Nombre de
                            usuario ya existe", CodigoError.
                            USERNAME_YA_EXISTE);
26                  } else if (message.contains("usuarios.email"))
                        { // CORREO DUPLICADO
27                      throw new TeamUPexception("Email ya existe"
                            , CodigoError.EMAIL_YA_EXISTE);
28                  }
29              } else {
```

```
30                    throw new PersistenceException("Error al
                          intentar persistir el usuario", e2);
31                }
32                throw e2;
33            } catch (Exception e) {
34                throw new Exception(e);
35            }
36        }
```

Listing 6.3: Ejemplo de DAO.

Continuando con la **arquitectura del servidor**, se muestran ejemplos de las capas de controladores y servicios. En primer lugar se muestran parte de la clase *LoginController* 6.4, estas clases son las encargadas de recibir las llamadas desde el módulo cliente en este caso, las configuraciones mas relevantes son:

1. *@RestController*. Esta anotación indica a *Spring* que la clase es un controlador *RESTful*, es decir, los métodos devuelven información y no una vista.
2. *@RequestMapping*. Se utiliza para mapear las solicitudes al controlador específico, en este caso el controlador agrupa todas las peticiones que apunten a la ruta */login* y lo distribuye al método que complete el resto de la *URL*.
3. *@PostMapping*. Mapea el método específico con la segunda parte de la *URL* de la petición, en este caso atiende únicamente solicitudes *POST*.

```
1     @RestController
2     @RequestMapping("/login")
3     public class LoginController {
4
5         @Autowired
6         private UserService userService;
7
8         @PostMapping("/tryLogin")
9         public ResponseData tryLogin(@RequestBody Map<String,
              String> loginRequest) {
10            String username = loginRequest.get("username");
11            String password = loginRequest.get("password");
12
13            return userService.validateLogin(username, password);
14        }
```

Listing 6.4: Ejemplo de *Controller*.

El controlador recoge la petición y extrae la información de esta para pasársela al servicio, que obtiene la responsabilidad a partir de este punto y devuelve una respuesta para que el controlador la mande de vuelta al cliente.

Por otro lado, las clases de servicio son las que albergan la lógica necesaria, tratan la información que les llega desde el controlador para conseguir un propósito específico, devolviendo un resultado que puede ser información obtenida de la base de datos, el resultado de aplicar alguna lógica concreta sobre los datos proporcionados o una respuesta alternativa en caso de error.

```java
@Service
public class UserService {

    @Autowired
    private UsuarioDAOimpl usuarioDAO;

    @Autowired
    private UserAuthenticationProvider
       userAuthenticationProvider;

    @Autowired
    private JuegoService juegoService;

    @PostConstruct
    public void init() { // EVITAR DEPENDENCIAS CIRCULARES
        juegoService.setUserService(this);
    }
```

Listing 6.5: Ejemplo de *Service* 1.

```java
public ResponseData validateLogin(String userName, String pass)
    {
    ResponseData res = new ResponseData();
    UsuarioEntity user = null;

    try { // 1. OBTENER USUARIO
        user = usuarioDAO.obtenerUsuarioPorNombre(userName);

        if(!PasswordManager.checkPassword(pass, user.
           getPassword_hash())) { // 1.1.- ENTRA SI PASS
           INCORRECTA
            res.setCode(CONSTASENYA_INCORRECTA); // code 2

        } else { // 2. ASIGNAR TOKEN
            String token = userAuthenticationProvider.
               createToken(user.getUsuario());

            // 3. MANDAR TOKEN
            res.getData().put("token", token);

            // 4. MANDAR RESTO DE CAMPOS NECESARIOS
            rellenarInfoUsuario(res.getData(), user);
            res.setCode(0);
        }
        return res;

    } catch (TeamUPexception e) { // NO EXISTE
        e.printStackTrace();
        res.setCode(e.getCodigoError().getCodigo());
        return res;

    } catch (Exception e) { // ERROR NO CONTROLADO
        e.printStackTrace();
        res.setCode(99);
        return res;
    }
}
```

Listing 6.6: Ejemplo de *Service* 2.

Ambas capturas 6.5 y 6.6 pertenecen a la clase *UserService*, se puede ver que en la declaración de la clase se anota con *@Service*, esto indica a *Spring Boot* que debe gestionar la clase como un *bean* y además declara que estará orientado a manejar la lógica de negocio. La anotación *@PostConstruct* sirve para indicar que el método anotado debe ejecutarse posterior a la inicialización del *bean*, en este caso se utiliza para evitar dependencias circulares con el *bean* de servicio *JuegoService*, ya que este otro *bean* también inyecta la clase *UserService* como dependencia.

En la segunda captura se muestra el código del método *validateLogin*, que dados el nombre y contraseña de un usuario, se encarga de validar que la información sea correcta y de preparar la respuesta para el cliente.

Continuando con la capa de **infraestructura y seguridad**, es especialmente importante la configuración que se hace desde el servidor, empezando por la configuración *CORS (Cross-Origin Resource Sharing)* [21] , un mecanismo de seguridad que regula desde que dominios se aceptan las peticiones *HTTP*, esto es especialmente útil pare evitar ataques del tipo *Cross-Site Scripting (XSS)*. En la figura 6.7 se muestra el filtro *CORS* del servidor.

```
1     @Bean
2     public FilterRegistrationBean<CorsFilter>
          corsFilterRegistration() {
3         UrlBasedCorsConfigurationSource source = new
              UrlBasedCorsConfigurationSource();
4         CorsConfiguration config = new CorsConfiguration();
5         config.setAllowCredentials(true);
6         config.addAllowedOrigin("http://localhost:3000");
7         config.setAllowedHeaders(Arrays.asList(
8             HttpHeaders.AUTHORIZATION,
9             HttpHeaders.CONTENT_TYPE,
10            HttpHeaders.ACCEPT));
11        config.setAllowedMethods(Arrays.asList(
12            HttpMethod.GET.name(),
13            HttpMethod.POST.name(),
14            HttpMethod.PUT.name(),
15            HttpMethod.DELETE.name()));
16        config.setMaxAge(MAX_AGE);
17        source.registerCorsConfiguration("/**", config);
18        FilterRegistrationBean<CorsFilter> bean = new
              FilterRegistrationBean<CorsFilter>(new CorsFilter(source
              ));
19
20        bean.setOrder(CORS_FILTER_ORDER);
21        return bean;
22    }
```

Listing 6.7: Ejemplo de configuración *CORS*.

Con la configuración que se muestra en la imagen previa se restringen las peticiones únicamente al origen *http:localhost:3000*, que es la dirección donde se despliega el cliente en el entorno de desarrollo. Esto se complementa con el *SecurityFilterChain* 6.8, que es una cadena de validación que proporciona *Spring Security* que consiste en aplicar una serie de filtros a las solicitudes http para permitir o no su acceso. En la captura que se muestra a continuación, se añade un filtro personalizado a la cadena y se excluyen de estos filtros las *url* de *"/login/register" y "login/tryLogin"*.

```
    @Bean
    SecurityFilterChain securityFilterChainOwn(HttpSecurity http)
        throws Exception {
         http
              .exceptionHandling(handling -> handling.
                 authenticationEntryPoint(
                 userAuthenticationEntryPoint))
              .addFilterBefore(new JwtAuthFilter(
                 userAuthenticationProvider),
                 BasicAuthenticationFilter.class)
              .csrf(csrf -> csrf.disable())
              .sessionManagement(management -> management.
                 sessionCreationPolicy(SessionCreationPolicy.
                 STATELESS))
              .authorizeHttpRequests(requests -> {
                   requests.requestMatchers("/login/register", "/login
                      /tryLogin").permitAll();
                   requests.anyRequest().authenticated();
              });
              return http.build();     }
```

Listing 6.8: Cadena de filtros.

```
    @Override
    protected void doFilterInternal(HttpServletRequest
        httpServletRequest, HttpServletResponse httpServletResponse,
                                        FilterChain filterChain) throws
                                            ServletException,
                                        IOException {
         String header = httpServletRequest.getHeader(HttpHeaders.
            AUTHORIZATION);

         if (header != null && header.startsWith("Bearer ")) {
              String token = header.substring(7);
              try {
                   SecurityContextHolder.getContext().
                      setAuthentication(
                           userAuthenticationProvider.validateToken(
                              token));
              } catch (AccessDeniedException e) { // TOKEN EXPIRADO
                   httpServletResponse.setStatus(HttpServletResponse.
                      SC_UNAUTHORIZED);
                   httpServletResponse.getWriter().write("Token
                      expirado");
                   return;

              } catch (RuntimeException e) { // EXCEPCIONES GENERALES
                   SecurityContextHolder.clearContext();
                   httpServletResponse.setStatus(HttpServletResponse.
                      SC_INTERNAL_SERVER_ERROR);
                   httpServletResponse.getWriter().write("Error en la
                      autenticacion");
                   return;
              } catch (Exception e) {
                   httpServletResponse.setStatus(HttpServletResponse.
                      SC_INTERNAL_SERVER_ERROR);
                   httpServletResponse.getWriter().write("Error
                      inesperado");
                   return;
```

```
            }
        }
        filterChain.doFilter(httpServletRequest,
            httpServletResponse);
    }
```

Listing 6.9: Filtro de seguridad personalizado.

```
@RequiredArgsConstructor
@Component
public class UserAuthenticationProvider {

    @Value("${security.jwt.token.secret-key:secret-key}")
    private String secretKey;

    @Autowired
    private UsuarioDAOimpl usuarioDAO;

    @PostConstruct
    protected void init() {
        secretKey = Base64.getEncoder().encodeToString(
            secretKey.getBytes());
    }

    public String createToken(String login) {
        Date now = new Date();
        Date validity = new Date(now.getTime() + 3600000); // 1
            HORA

        Algorithm algorithm = Algorithm.HMAC256(secretKey);
        return JWT.create()
            .withSubject(login)
            .withIssuedAt(now)
            .withExpiresAt(validity)
            .sign(algorithm);
    }

    public Authentication validateToken(String token) throws
        Exception {
        try {
            Algorithm algorithm = Algorithm.HMAC256(secretKey);
            JWTVerifier verifier = JWT.require(algorithm).build
                ();
            DecodedJWT decoded = verifier.verify(token);
            UsuarioEntity user = usuarioDAO.
                obtenerUsuarioPorNombre(decoded.getSubject());

            return new UsernamePasswordAuthenticationToken(user
                , null, Collections.emptyList());

        } catch (Exception e) {
            throw new AccessDeniedException("Token expirado");
        }
    }
}
```

Listing 6.10: Verificación del token.

El filtro que se añade a la cadena 6.9 es el encargado de validar que en la solicitud al servidor se incluye una cabecera de autorización y un token válido. Si el token ya está

caducado, se remite el error correspondiente al cliente para que este pueda actuar en consecuencia, en caso de que pase el filtro, se continua con la cadena de validación.

Este filtro llama a *validateToken* 6.10, que es la función encargada de comprobar que el token es valido, para ello verifica que se ha encriptado con la misma clave que el servidor y comprueba si tiene más de una hora de antigüedad.

Por último, la parte en el cliente que se encarga de la gestión del token de acceso. Esto se define en el interceptor *Axios* 6.11 que siempre que haya un token disponible en *localStorge* lo incluye en las solicitudes al servidor.

```
export const getAuthToken = () => {
    return window.localStorage.getItem('auth_token');
};

export const setAuthHeader = (token) => {
    if (token !== null) {
        window.localStorage.setItem("auth_token", token);
    } else {
        window.localStorage.removeItem("auth_token");
    }
};

export const removeAuthHeader = () => {
    window.localStorage.removeItem("auth_token");
};


export const request = (method, url, data) => {
    let headers = {};
    if (getAuthToken() !== null && getAuthToken() !== "null" &&
        getAuthToken() !== "undefined") {
        headers = {'Authorization': `Bearer ${getAuthToken()
            }`};
    }

    return axios({
        method: method,
        url: url,
        headers: headers,
        data: data
    });
};
```

Listing 6.11: Gestión del token desde el cliente.

Pasando al **código del cliente**, es interesante primero mencionar la estructura del proyecto, donde cada componente esta en la ruta *src/components* y consta de su archivo *.jsx y .css*. Están separados los archivos para *redux*, que se hayan en *src/redux* y para *Axios*, que se encuentran en *src/helpers*. Las imágenes se encuentran dentro del directorio */public*.

Los componentes de la aplicación siguen una estructura similar, en primer lugar se traen las dependencias necesarias de bibliotecas externas o de otros componentes, después se declara el componente, en este se define la lógica para auto gestionar el estado de este y en la parte final se devuelve la representación de este componente en el *DOM*, finalmente se exporta el componente para poder ser utilizado por otros componentes en caso de necesitarlo.

A continuación se muestra un ejemplo del componente *App.jsx* 6.12, encargado de alojar el resto de componentes. Este componente no contiene lógica ya que no la requiere. Se puede consultar el código de cualquier componente en el repositorio de *GitHub* 6.5.

```
// REACT
import React from 'react';
import { BrowserRouter as Router, Route, Routes } from 'react-
    router-dom';

// MIS COMPONENTES
import Login from '../Login/Login';
import Register from '../Register/Register';
import Home from '../Home/Home';

// AXIOS
import AxiosInterceptor from '../../helpers/axios_helper';

//////////////////
//  FIN IMPORTS //
//////////////////

function App() {

  // DOM
  return (
    <Router>
      <AxiosInterceptor>
        <div className="App">
          <Routes>
            <Route path="/login" element={<Login />} />
            <Route path="/register" element={<Register />} />
            <Route path="/Home" element={<Home />} />
            <Route path="/" element={<Login />} /> {/* Ruta por
                defecto */}
          </Routes>
        </div>
      </AxiosInterceptor>
    </Router>
  )
}

export default App
```

Listing 6.12: Componente *App*.

Por último, es especialmente interesante la integración de *Redux* con la aplicación. El elemento central de *Redux* es el componente *store*, que es único y mantiene el estado de la aplicación. Posteriormente se definen los *slices*, que son clases que implementan *reducers*, estos últimos son los métodos que definen las acciones que disparan los cambios sobre el estado global. Por ejemplo, en el *userSlice*, se define el *reducer addGame*, cuando se llame a este *reducer* se le pasa el id del usuario y el id del juego, después se añade el juego dado a la lista de juegos del usuario. De esta forma se actualiza el estado global de la aplicación y cualquier otro componente dependiente de la lista de juegos es sensible a los cambios y se actualiza al realizarse algún cambio. Adicionalmente, el *store* de la aplicación se configura de manera que se guarda en el *localStorage* para mantener los datos en las recargas de la página. Las figuras 6.13, 6.14 y 6.15 muestran parte de la configuración para *Redux*.

```
// REDUX
import { configureStore } from '@reduxjs/toolkit';
import { persistStore, persistReducer } from 'redux-persist';
import storage from 'redux-persist/lib/storage';

// MIS COMPONENTES
import rootReducer from './rootReducer';

const persistConfig = {
  key: 'xxxx',
  storage,
};

const persistedReducer = persistReducer(persistConfig,
   rootReducer);

export const store = configureStore({
  reducer: persistedReducer,
});

export const persistor = persistStore(store);
```

Listing 6.13: Componente *store*.

```
// REDUX
import { combineReducers } from 'redux';
import userReducer from './usersSlice';
import postsReducer from './postsSlice';
import gamesReducer from './gamesSlice';
import chatsReducer from './chatsSlice';

const rootReducer = combineReducers({
  user: userReducer,
  posts: postsReducer,
  games: gamesReducer,
  chats: chatsReducer
});

export default rootReducer;
```

Listing 6.14: Ejemplo *rootReducer*.

```
// REDUX
import { createSlice } from '@reduxjs/toolkit';

//////////////////
//  FIN IMPORTS //
//////////////////

const initialState = {
  currentUser: null,
  users: {}
};

const userSlice = createSlice({
  name: 'user',
  initialState,
  reducers: {
    setUser: (state, action) => {
      const { id, name, img, juegos, chats} = action.payload;
      state.currentUser = id;
      state.users[id] = { name, img, juegos: juegos, chats:
          chats };
    },
    clearUser: (state) => {
      state.currentUser = null;
    },
    addChat: (state, action) => {
      const { userId, chat } = action.payload;
      state.users[userId].chats.push(chat);
    },
    addGame: (state, action) => {
      const { userId, game } = action.payload;
      state.users[userId].juegos.push(game);
      state.users[userId].juegos.sort((a, b) => a.id - b.id);
    },
    removeChat: (state, action) => {
      const { userId, chatId } = action.payload;
      state.users[userId].chats = state.users[userId].chats.
          filter(chat => chat.id !== chatId);
    },
    removeGame: (state, action) => {
      const { userId, gameId } = action.payload;
      state.users[userId].juegos = state.users[userId].juegos.
          filter(game => game.id !== gameId);
    },
  }
});

export const { setUser, clearUser, addChat, addGame, removeChat
    , removeGame } = userSlice.actions;
export default userSlice.reducer;
```

Listing 6.15: Ejemplo de *userSlice*.

CAPÍTULO 7
Producto desarrollado

En este capitulo se mostrará paso a paso con el uso de imágenes como se navega por la aplicación y se llevan a cabo las diferentes funcionalidades que se ofrecen. Comenzando por la primera interfaz de la aplicación, la pantalla de *Login* 7.1. Desde está ventana el usuario puede pulsar al encale para ir a la vista de registro para r o completar el formulario del *login* para acceder a la app.

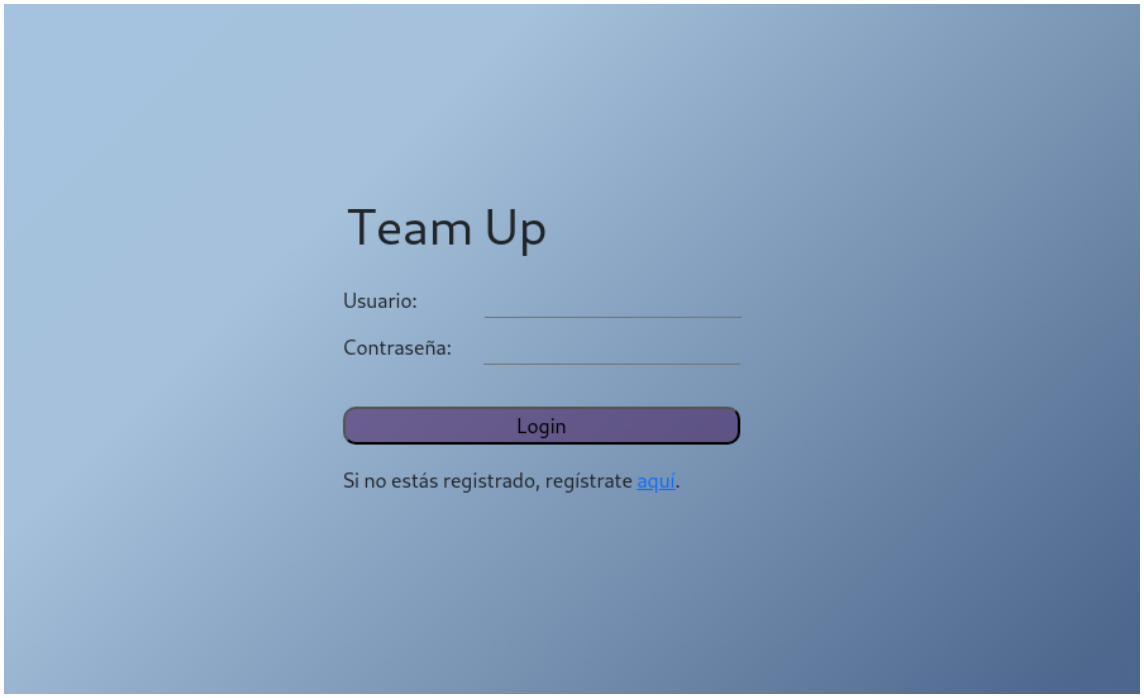

Figura 7.1: Página de *Login*.

Con un usuario registrado y desde la vista de *Login*, si se introducen las credenciales del usuario y son correctas, se navega a la vista de *Home* 7.3. Este proceso puede llegar a tardar unos pocos segundos ya que se solicitan recursos al servidor, por este motivo se muestra una mascara de carga mientras se transiciona a la nueva vista.

Esta es la vista principal de la aplicación, en este caso muestra como sería el *Home* de un usuario sin ningún juego y ningún chat en sus listas. Se puede apreciar el juego "Lupa" que cuando está seleccionado, muestra el enlace que se ve en pantalla "Busca juegos aquí y añádelos a tu lista", haciendo clic en el enlace se muestra el buscador de juegos, este es el escenario 4.1.

Figura 7.2: Página de *Registro*.

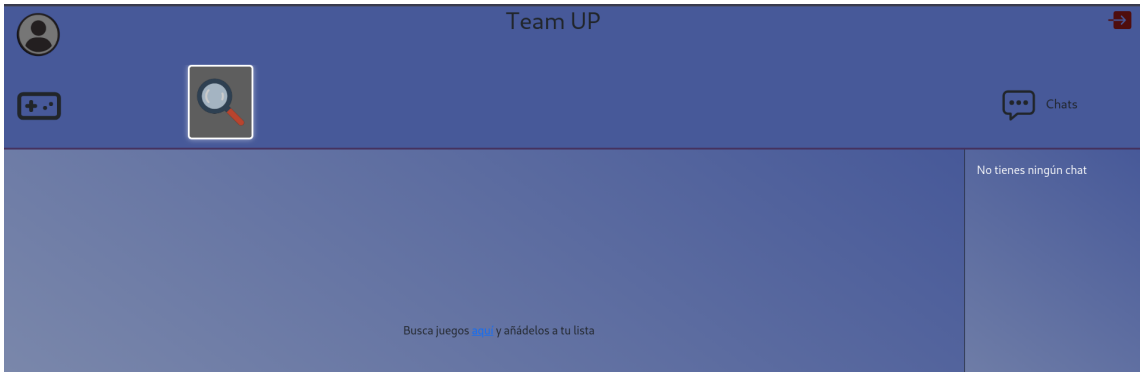

Figura 7.3: Página *Home*.

En el componente *GameFInder* 7.4 el usuario puede filtrar por nombre en la barra de búsqueda, cuando se hace clic sobre un juego, este se elimina de la ventana de búsqueda y se muestra un *toast* para indicar que el juego se ha añadido a la biblioteca del usuario, reproduciendo el escenario 4.2. Este proceso se puede repetir hasta que el usuario haya añadido todos los juegos de su interés, una vez finalizado, se puede cerrar la ventana desde la "x" de la esquina superior derecha y se cierra el componente. Para navegar entre los juegos disponibles se muestra una barra de scroll vertical en el lateral derecho del componente. Al cerra el componente se actualiza la lista de juegos del usuario y se regresa al *Home*. Por defecto al volver se selecciona el juego que esté a la izquierda de la lista del usuario, se ordenan por un *id* interno. Esto se describe en el escenario 4.3.

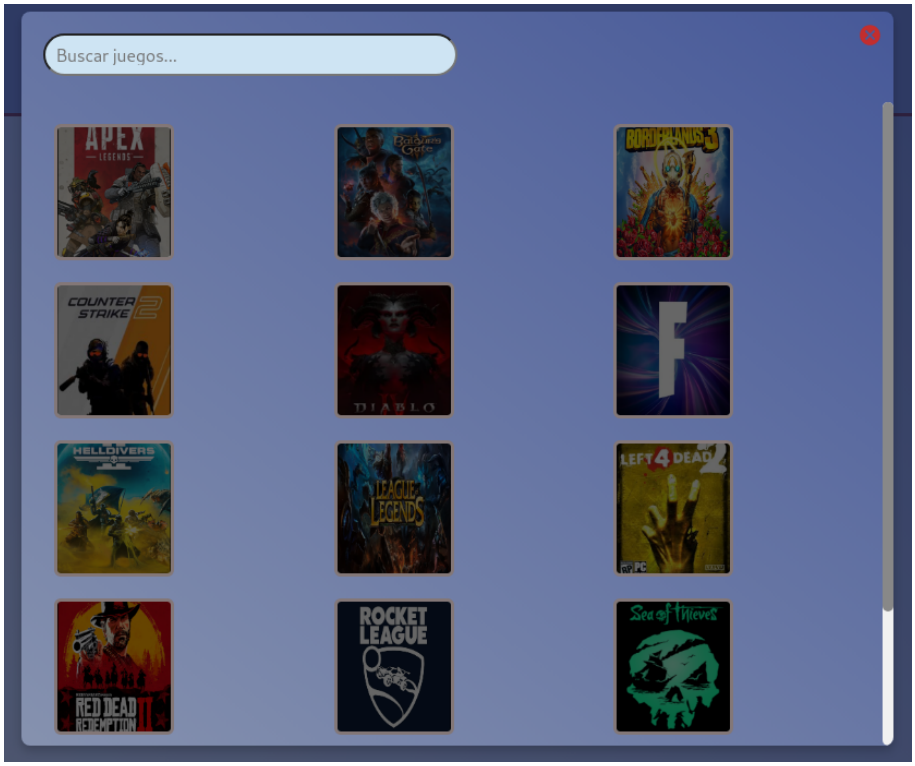

Figura 7.4: Componente *GameFinder*.

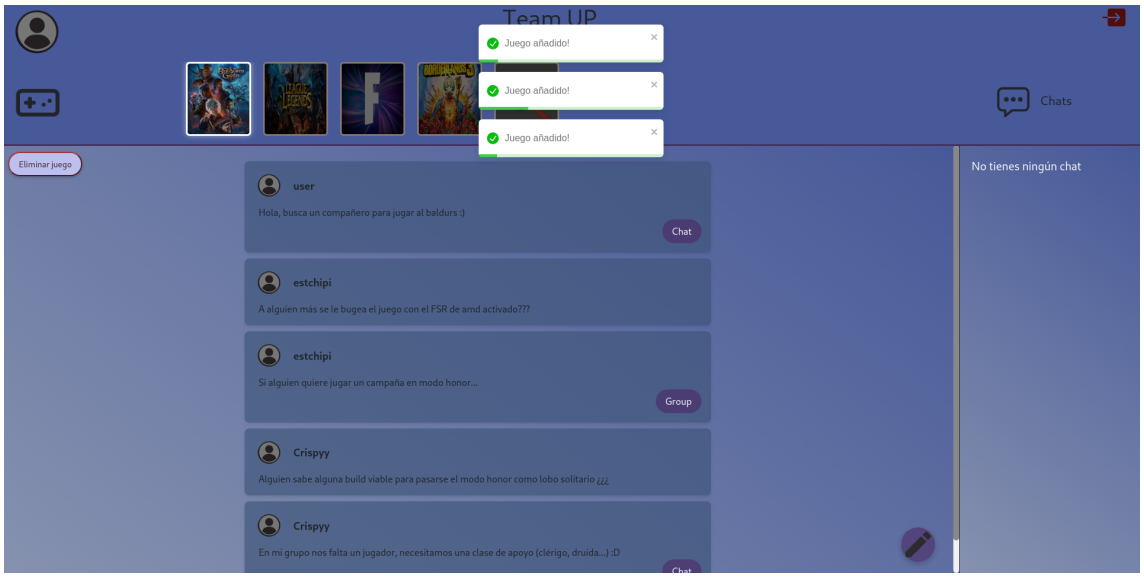

Figura 7.5: Componente *Home* con foro.

Al volver al *Home* 7.5, se muestra el foro correspondiente al juego más a la izquierda, en este caso corresponde con el juego *Baldur's Gate III*, en el foro se muestran los *posts* de los usuarios, en la imagen se puede apreciar la lista de juegos del usuario actualizada, el botón en la parte inferior derecha para publicar un *post*, y los botones de dos publicaciones, una con la invitación a un chat y otra con la invitación a un grupo. También se resalta el botón para eliminar el juego de la biblioteca, en la esquina superior izquierda, al pulsarlo se muestra un toast para indicar el éxito y se elimina el juego de la lista, como indica el escenario 4.4. También se muestran los *toast* de éxito tras realizar la operación de añadir un juego. Si se pulsa sobre el icono de *Crear Post* se abre una ventana flotante para escribir una publicación en el foto en cuestión, como se muestra en la figura 7.6 y se

explica en el escenario 4.5.

Figura 7.6: Componente *WritePost*.

En este componente se puede selecciona el numero de participantes deseados, si se mantiene a 0 se envía un *Post* sin invitación, si se selecciona 1 se crea un post con invitación a chat, si se selecciona un número mayor se crea un *Post* con invitación a un grupo, con el numero de participantes seleccionado, con un máximo de 7. Para poder enviar la publicación hay un mínimo de tres caracteres. Estas acciones corresponden con los escenarios 4.6 y 4.7.

Para unirse unirse a un chat o grupo deben cumplirse algunas condiciones, en primer lugar no ser el creador del post, no pertenecer ya al chat o grupo en cuestión y por último que el haya huecos disponibles. Siguiendo la imagen de ejemplo, si un usuario pulsa sobre el botón de "Chat" de un post y se cumplen las condiciones descritas, se muestra un toast para indicar el éxito de la operación y el nuevo chat aparece en la lista del usuario 7.7, reproduciendo así el escenario 4.8 o 4.9

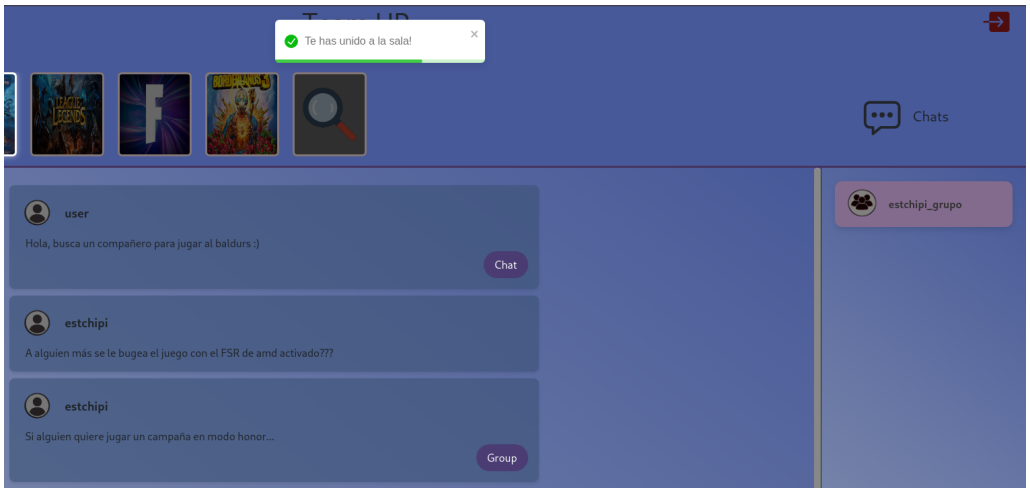

Figura 7.7: Unirse a un chat.

Una vez en la lista de chats, el usuario puede seleccionar cualquiera de sus chats disponibles, en cuyo caso se sustituirá el panel central del *Home* por la sala de chat correspondiente, como sucede en la figura 7.8. Esto se corresponde con el escenario 4.10

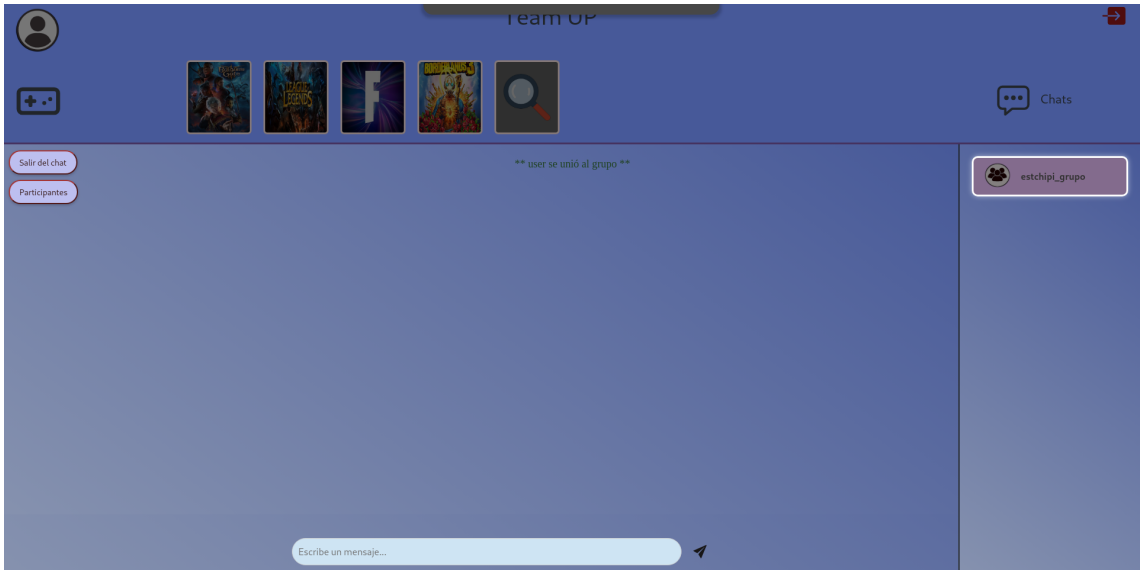

Figura 7.8: Componente *SalaChat*.

En la sala chat se muestran los últimos 50 mensajes que han mandado los usuarios, en la imagen se aprecian los botones para abandonar el chat y para consultar los participantes en la esquina superior izquierda, estas acciones se corresponden a los escenarios 4.13 y 4.12. Se puede ver también la barra para escribir y enviar un mensaje, cuando se pone el foco sobre esta se expande para permitir al usuario mayor comodidad al escribir mensajes largos, esto se corresponde con el escenraio 4.11. También se aprecia el mensaje de bienvenida que se muestra cuando un usuario entra al grupo, de manera alterna se manda otro mensaje de despedida cuando lo abandona. Los mensajes escritos por el usuario activo se muestran a la derecha, y para este mismo usuario se muestran a la izquierda los mensajes del resto de usuarios. Los mensajes se ordenan por fecha de envío, en el componente *Mensaje* se muestra la foto y nombre de usuario, también el mes, día y hora de envío, en el formato *dd/mm hh:mm*. Por último el botón de *Cerrar sesión* que se encuentra en la esquina superior derecha de la aplicación. Al pulsar sobre este se borran todos los datos guardados del usuario en cuestión y se redirige la aplicación a la pantalla de *Login* con un *toast* para indicar que se ha cerrado sesión correctamente.

CAPÍTULO 8
Validación y despliegue

En este capitulo se detalla la validación de la interfaz final. También se explica como se realiza el despliegue completo de la aplicación a un entorno de producción.

8.1 Validación

Gracias a la metodología *Scrum*, se ha podido probar el código entregado al final de cada sprint, lo cual ha permitido asegurar que todo funcionase correctamente antes de abordar el siguiente hito del proyecto. Este enfoque iterativo ha facilitado identificar y solucionar problemas rápidamente, manteniendo siempre la calidad del software. Al realizar estas revisiones constantes, se ha garantizado que cada nueva funcionalidad se integre sin inconvenientes. Una vez finalizado la fase de desarrollo, se han realizado pruebas de todas las funcionalidades exhaustivamente para su posterior despliegue a producción.

A continuación, se presenta una evaluación heurística de la interfaz de la aplicación basada en los 10 principios de usabilidad de Jakob Nielsen [20]. Esta evaluación tiene como objetivo analizar la usabilidad de la interfaz de usuario, asegurando que el diseño de la aplicación sea intuitivo, eficiente y fácil de usar. Cada principio se aborda a través de una pregunta, seguida de una observación sobre cómo la aplicación maneja dicho aspecto, y una conclusión que resume la efectividad de la interfaz en ese principio específico.

- ¿La aplicación mantiene informados a los usuarios sobre lo que está sucediendo?

  Observación: La aplicación proporciona retroalimentación visual inmediata cuando se añaden o completan tareas. Se muestran notificaciones a modo de *toast* para las acciones importantes.

  Conclusión: Sí, la visibilidad del estado del sistema es adecuada, manteniendo informados a los usuarios sobre sus acciones y el estado actual.

- ¿Utiliza la aplicación un lenguaje familiar para los usuarios y sigue convenciones del mundo real?

  Observación: La interfaz utiliza términos comunes para gestionar tareas, como “eliminar”, “consultar” y “buscar”. Además, emplea iconos familiares (por ejemplo, una “x”, un avión de papel, una lupa, etc).

  Conclusión: Sí, la aplicación utiliza un lenguaje e iconos que facilitan la experiencia del usuario.

- ¿Pueden los usuarios deshacer o rehacer fácilmente sus acciones?

Observación: La aplicación permite deshacer acciones como eliminar juegos recién añadidos y lo mismo en cuanto a chats. También permite la navegación con las flechas del navegador para volver a la página anterior o posterior.

Conclusión: Sí, los usuarios tienen control sobre sus acciones y pueden corregir errores rápidamente.

- ¿La aplicación sigue convenciones de diseño consistentes y familiares?

  Observación: La interfaz mantiene un diseño consistente en todas las páginas, con elementos de navegación y botones colocados uniformemente. Se respetan los estándares comunes de diseño web.

  Conclusión: Sí, la aplicación es coherente y sigue estándares reconocidos, lo que facilita la navegación.

- ¿La aplicación ayuda a prevenir errores antes de que ocurran?

  Observación: Se limitan las acciones de los usuarios para guiarlos a un correcto uso de la interfaz, como limitaciones en los formularios o avisos emergentes.

  Conclusión: Sí, se implementan medidas para ayudar a prevenir errores comunes.

- ¿Los usuarios pueden reconocer opciones y funciones fácilmente en lugar de tener que recordarlas?

  Observación: Las opciones de menú están claramente visibles y etiquetadas, y los iconos tienen etiquetas explicativas junto a ellos.

  Conclusión: Sí, el diseño prioriza el reconocimiento visual, facilitando la interacción del usuario.

- ¿Se evita la sobrecarga de información y se mantiene un diseño limpio y enfocado?

  Observación: La interfaz es minimalista y utiliza un diseño simple con colores suaves. La información esencial está destacada, mientras que los elementos secundarios están minimizados.

  Conclusión: Sí, el diseño es estético y evita el desorden, mejorando la experiencia de usuario.

- ¿La aplicación proporciona mensajes de error claros y útiles?

  Observación: Los mensajes de error son específicos y ofrecen sugerencias para resolver el problema. Por ejemplo, si un usuario introduce información incorrecta en el formulario de registro.

  Conclusión: Sí, los mensajes de error son claros y proporcionan orientación para resolver los problemas.

En resumen, la evaluación heurística de la interfaz de la aplicación basada en los principios de usabilidad de Jakob Nielsen revela un diseño bien pensado y orientado al usuario. La aplicación cumple con los estándares de visibilidad del estado del sistema, uso de lenguaje familiar, control del usuario, consistencia en el diseño y prevención de errores. Además, promueve la facilidad de reconocimiento sobre el recuerdo, evita la sobrecarga de información y proporciona mensajes de error claros y útiles. Estas cualidades aseguran una experiencia de usuario intuitiva, eficiente y agradable, facilitando la interacción y mejorando la satisfacción del usuario.

8.2 Despliegue

En esta sección se explica como una vez terminado el desarrollo se prepara la configuración para producción y se despliega la aplicación por completo para que sea accesible para cualquier persona desde un dominio público.

Es importante mencionar que el código adaptado para producción tiene ciertas diferencias con el código de la fase de desarrollo, como credenciales reales y algunas adaptaciones para su correcto funcionamiento. El código de producción se lanza desde un repositorio privado de *Github*, de manera que cuando se hace un *push* al repositorio con nuevas versiones, estas se integran directamente en en la aplicación y se actualiza su estado.

El despliegue se ha realizado en tres fases, una para cada módulo de la aplicación, en primer lugar el cliente o *front-end*. Para ello se ha utilizado la plataforma *Vercel* [18], una plataforma para servir aplicaciones en la nube, especialmente pensada para aplicaciones *front-end* y *frameworks* modernos entre los que se encuentra *React*.

Para los módulos de *back-end* y la base de datos se ha utilizado *RailWay* [19], con el módulo del servidor al igual que ocurre con el cliente, se despliega desde un repositorio privado de *GitHub* ya que este contiene información sensible como las credenciales de acceso para la base de datos y la clave para cifrar los tokens.

Railway es una plataforma que ofrece la posibilidad para desplegar aplicaciones en la nube, y como sucede con *Vercel*, permite una integración continua desde un repositorio *Github*, de manera que los cambios en el repositorio se ven reflejados en la aplicación desplegada. La base de datos también se lanza en esta plataforma, pero al no desplegarse desde un repositorio de código es necesario una vez lanzado el servicio para alojar la base de datos, ejecutar el *SQL* para la creación y inserción de datos básico. En el anexo 10 se muestra el *script populate_db.sql*.

Mencionar también que las plataformas utilizadas proporcionan un certificado *SSL* lo que hace que el entorno de producción funcione sobre el protocolo *HTTPS*, que es un estándar de seguridad indispensable en la actualidad para garantizar la integridad y privacidad de la información.

Una vez desplegadas los diferentes módulos que componen la aplicación, **es posible acceder a ella a través del dominio https://teamup-frontend-pi.vercel.app/**

CAPÍTULO 9
Conclusiones y trabajos futuros.

En este último capitulo se presentan las conclusiones obtenidas tras toda el proceso de desarrollo del proyecto así como una lista de funcionalidad para implementar en la aplicación que no se han integrado en la primera versión.

9.1 Conclusiones

El proyecto llevado a cabo ha explorado todo el proceso de creación de una aplicación, desde la persistencia de los datos hasta la interfaz del usuario. Se han utilizado tecnologías y arquitecturas que tienen una gran demanda en el ámbito empresarial. Desde el nacimiento de la idea se le ha ido dando forma poco a poco con cada *sprint* del desarrollo hasta finalmente concluir en una aplicación que cumple todos los requisitos definidos en un principio.

Por el lado del servidor, el potencial de *Java* como uno de los lenguajes con más relevancia para desarrollar código orientado al *back-end* combinado con *Spring Boot* y con *Maven* ofrecen una experiencia de desarrollo accesible y ambiciosa a partes iguales. Este conjunto de herramientas software combinado con otras igualmente importantes como *Github* o Eclipse han permitido llevar a cabo un desarrollo de calidad, especialmente en cuanto a prácticas en la metodología de trabajo y en la arquitectura de la aplicación.

Respecto al cliente, se exploró la posibilidad de usar otros *frameworks* para el desarrollo como *Extjs* [45] o *Angular* [46] , finalmente se eligió *React*. Ha sido sin duda una elección acertada, por un lado ha permitido explorar todo lo que ofrece esta librería en cuanto a facilidad para el diseño de interfaces y mejoras respecto a la versión base de *JavaScript*. No es casualidad que *React* se encuentra en la cabeza de la lista de tecnologías más utilizados en el ámbito de la programación, tanto por las funcionalidades que ofrece, la relativa sencillez de su uso y el gran apoyo de la comunidad.

Para concluir, mi opinión personal respecto a este proyecto es muy positiva. Ha sido complejo compaginar el desarrollo del proyecto con un trabajo a tiempo completo, no obstante estoy orgulloso del resultado final. Por un lado he afianzado algunas *soft-skills* como: resolución de problemas, adaptabilidad, gestión del tiempo, creatividad, paciencia y perseverancia y autonomía entre otras. También me ha permitido adquirir nuevos conocimientos técnicos como el aprendizaje de nuevos *frameworks* como es el caso de *React* y otras bibliotecas interesantes. En general creo que la realización del proyecto me ha ayudado a progresar en mi carrera profesional, también me proporciona un proyecto interesante que añadir en mi portafolio de cara al ámbito profesional además de impulsar mis ganas a la hora de querer realizar proyectos independientes.

9.2 Trabajos futuros

Durante el planteamiento y desarrollo de la aplicación, han ido surgiendo ideas y funcionalidades que se han descartado mayormente por limitaciones en cuanto al tiempo. Una lista de algunas funcionalidades que no se han podido implementar en esta primera versión de la aplicación pero es viable integrar son:

- **Validación del correo y recuperar contraseña**. Implementar un servicio para validar el correo proporcionado por los usuarios, que iría de la mano de la posibilidad de recuperar la contraseña en caso de perdida.
- **Fotos personalizadas**. Permitir a los usuarios cambiar su foto de perfil y foto de chats o grupos. Una primera aproximación podría ser simplemente añadir la posibilidad de elegir una foto entre un conjunto dado y en un futuro permitir a los usuarios subir su propio contenido.
- **Permitir multimedia en las publicaciones o mensajes**. Sería una mejora considerable en cuanto a la calidad y libertad de comunicación de los usuarios de la aplicación.
- **Añadir filtros y moderación**. Un punto con cierta complejidad pero necesario para garantizar que la aplicación sea un lugar agradable para los usuarios.
- **Eliminar publicaciones**. Permitir a los usuarios la opción de borrar sus propias publicaciones.
- **Lista de bloqueados**. Permitir a los usuarios bloquear a otros usuarios.
- **Mensajes pendientes**. Una alerta para indicar que hay mensajes pendientes de leer en el chat.
- **Copias de seguridad**. De manera periódica, implementar algún proceso para comprimir y almacenar los datos de la aplicación al mismo tiempo que se vacía el almacenamiento. Esto cumpliría la función de mantener un volumen de datos manejable y de poder mantener un histórico de datos.

Un cambio más ambicioso sería migrar la arquitectura de la aplicación a una orientación para *micro-servicios*, se valoró aplicar esta arquitectura desde un primer momento pero se descarto por la complejidad y volumen de trabajo extra que hubiese requerido su implementación.

9.3 Relación con los estudios cursados

Este trabajo tiene como fin aplicar los conocimientos adquiridos durante los estudios del grado para demostrar la capacidad de diseñar soluciones a problemas reales. El proyecto se centra en una aplicación web, lo cual ha requerido un sólido entendimiento de asignaturas relacionadas con redes, desarrollo web y programación. Además, también han sido esenciales otras asignaturas orientadas a metodologías de trabajo y gestión de proyectos para asegurar un desarrollo exitoso. Las asignaturas con mayor relevancia son:

- **Ingeniería del Software (ISW)**: Metodologías para el trabajo en equipos de desarrollo y diseño de la arquitectura del software.

- **Gestión de Proyectos (GPR)**: Planificación de tiempos y distribución de recursos en un proyecto.
- **Desarrollo Web (DEW)**: Creación y diseño de interfaces *front-end*, y gestión de peticiones HTTP.
- **Integración de Aplicaciones (IAP)**: Uso de *APIs REST*, comunicaciones asíncronas y manejo de flujos de datos.
- **Desarrollo Centrado en el Usuario (DCU)**: Diseño y evaluación de interfaces de usuario para aplicaciones web.

Bibliografía

[1] Luis J. Muñoz Hernández, Jorge P. Montoya Gómez, y Claudia J. García Chavarría. Impacto de las redes sociales e internet en la salud. *Revista Médica Clínica Las Condes*, 26(2):117–125, junio, 2015. https://acortar.link/HES08u.

[2] Émile Durkheim. *La división del trabajo social*. Biblioteca Nueva, Clásicos del pensamiento económico y social / Biblioteca Nueva Minerva, 2012.

[3] Erving Goffman. *La presentación de la persona en la vida cotidiana*. Amorrortu, Biblioteca de sociología, 2004.

[4] Wikipedia contributors. *Discord*. Consultado el 15 de febrero de 2024, de https://es.wikipedia.org/wiki/Discord.

[5] Wikipedia contributors. *Reddit*. Consultado el 24 de febrero de 2024, de https://en.wikipedia.org/wiki/Reddit.

[6] Wikipedia contributors. *Twitter*. Consultado el 25 de febrero de 2024, de https://es.wikipedia.org/wiki/Twitter.

[7] Fogges official website. *Fogges*. Consultado el 26 de febrero de 2024, de https://fogges.com/es.

[8] Twich official website. *Twich*. Consultado el 27 de febrero de 2024, de https://www.twitch.tv.

[9] Atlassian. *Qué es Scrum?*. Consultado el 4 de marzo de 2024, de https://www.atlassian.com/es/agile/scrum.

[10] Lucidchart contributors. *Lucidchart*. Consultado el 5 de marzo de 2024, de https://www.lucidchart.com/pages/.

[11] React contributors. *React*. Consultado el 5 de marzo de 2024, de https://reactjs.org.

[12] Juan D. *Arquitectura de una sola página*. Consultado el 10 de abril de 2024, de https://juanda.gitbooks.io/webapps/content/spa/arquitectura_de_un_spa.html.

[13] Redux Toolkit contributors. *Redux Toolkit*. Consultado el 13 de mayo de 2024, de https://redux-toolkit.js.org.

[14] Hibernate contributors. *Hibernate*. Consultado el 15 de abril de 2024, de https://hibernate.org

[15] Pivotal Software. *Spring Boot*. Consultado el 28 de marzo de 2024, de https://spring.io/projects/spring-boot.

[16] IBM. *Arquitectura de tres capas*. Consultado el 29 de marzo de 2024, de https://www.ibm.com/es-es/topics/three-tier-architecture.

[17] GitHub contributors. *GitHub*. Consultado el 28 de marzo de 2024, de https://github.com.

[18] Vercel contributors. *Vercel*. Consultado el 10 de junio de 2024, de https://vercel.com.

[19] Railway contributors. *Railway*. Consultado el 4 de julio de 2024, de https://railway.app.

[20] Digriup contributors. *La evaluación heurística y las normas de usabilidad según Nielsen*. Consultado el 20 de Agosto de 2024, de https://digrup.com/es/blog/la-evaluacion-heuristica-y-las-normas-de-usabilidad-segun-nielsen.

[21] MDN contributors. *CORS*. Consultado el 9 de marzo de 2024, de https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS.

[22] Oracle. *Java*. Consultado el 30 de agosto de 2024, de https://www.java.com/es/.

[23] ECMA International. *ECMAScript Language Specification*. Consultado el 9 de marzo de 2024, de https://tc39.es/ecma262/.

[24] Apache Software Foundation. *Apache Tomcat*. Consultado el 2 de marzo de 2024, de https://tomcat.apache.org/.

[25] Apache Software Foundation. *Apache Maven*. Consultado el 15 de febrero de 2024, de https://maven.apache.org/.

[26] Amazon Web Services. *RESTful API*. Consultado el 8 de marzo de 2024, de https://aws.amazon.com/es/what-is/restful-api/.

[27] npm, Inc. *npm*. Consultado el 15 de enero de 2024, de https://www.npmjs.com/.

[28] JSON.org. *JSON*. Consultado el 20 de febrero de 2024, de https://www.json.org/json-en.html.

[29] Takezoe, Naoki. *Amateras Update Site*. Consultado el 3 de septiembre de 2024, de https://takezoe.github.io/amateras-update-site/.

[30] Eclipse Foundation. *Eclipse Downloads*. Consultado el 17 de enero de 2024, de https://www.eclipse.org/downloads/.

[31] JSON Web Tokens. *JWT*. Consultado el 15 de marzo de 2024, de https://jwt.io/.

[32] Git SCM. *Git*. Consultado el 13 de marzo de 2024, de https://git-scm.com/.

[33] Microsoft. *Visual Studio Code*. Consultado el 16 de febrero de 2024, de https://code.visualstudio.com/.

[34] Project Lombok. *Lombok*. Consultado el 26 de abril de 2024, de https://projectlombok.org/.

[35] Oracle. *Java Persistence API (JPA)*. Consultado el 15 de marzo de 2024, de https://www.oracle.com/java/technologies/persistence-jsp.html.

[36] DBeaver Corporation. *DBeaver*. Consultado el 28 de febrero de 2024, de https://dbeaver.io/.

[37] Auth0 contributors. *Bcrypt Overview*. Consultado el 8 de marzo de 2024, de https://auth0.com/blog/what-is-bcrypt/.

[38] Auth0 contributors. *Spring Boot Security Overview*. Consultado el 7 de marzo de 2024, de https://auth0.com/docs/quickstart/backend/java-spring-security5/.

[39] React contributors. *Introducing JSX*. Consultado el 4 de abril de 2024, de https://es.legacy.reactjs.org/docs/introducing-jsx.html.

[40] Podman contributors. *Podman*. Consultado el 7 de febrero de 2024, de https://podman.io/.

[41] Fedora Project. *Fedora*. Consultado el 17 de enero de 2024, de https://getfedora.org/.

[42] Overleaf. *Overleaf, Online LaTeX Editor*. Consultado el 10 de enero de 2024, de https://www.overleaf.com/.

[43] Toastify contributors. *Toastify*. Consultado el 10 de abril de 2024, de https://fkhadra.github.io/react-toastify/.

[44] GIMP Development Team. *GIMP - GNU Image Manipulation Program*. Consultado el 13 de marzo de 2024, de https://www.gimp.org/.

[45] Sencha Inc. *Ext JS*. Consultado el 15 de febrero de 2024, de https://www.sencha.com/products/extjs/.

[46] Angular contributors. *Angular*. Consultado el 20 de febrero de 2024, de https://angular.io/.

CAPÍTULO 10
Anexo

En este anexo se muestra el *script populate_db.sql*. Ofrece la infraestructura e información básica para la base de datos de la aplicación.

```
DROP TABLE IF EXISTS `chats`;
CREATE TABLE `chats` (
  `id` bigint NOT NULL AUTO_INCREMENT,
  `name` varchar(255) DEFAULT NULL,
  `img` varchar(255) DEFAULT NULL,
  `tipo` varchar(255) DEFAULT NULL,
  `post_id` bigint DEFAULT NULL,
  PRIMARY KEY (`id`),
  UNIQUE KEY `post_id` (`post_id`),
  CONSTRAINT `chats_ibfk_1` FOREIGN KEY (`post_id`) REFERENCES `posts` (`id`) ON DELETE SET NULL
) ENGINE=InnoDB AUTO_INCREMENT=44 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;

DROP TABLE IF EXISTS `foros`;
CREATE TABLE `foros` (
  `id` bigint NOT NULL AUTO_INCREMENT,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=1000 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;


LOCK TABLES `foros` WRITE;
INSERT INTO `foros` VALUES (1),(2),(3),(4),(5),(6),(7),(8),(9),(10),(11),(12),(13),(14),(999);
UNLOCK TABLES;


DROP TABLE IF EXISTS `juegos`;
CREATE TABLE `juegos` (
  `id` bigint NOT NULL,
  `nombre` varchar(255) NOT NULL,
  `img` varchar(255) DEFAULT NULL,
  `foro_id` bigint DEFAULT NULL,
  PRIMARY KEY (`id`),
  UNIQUE KEY `nombre` (`nombre`),
  UNIQUE KEY `foro_id` (`foro_id`),
  CONSTRAINT `juegos_ibfk_1` FOREIGN KEY (`foro_id`) REFERENCES `foros` (`id`) ON DELETE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;

```

```
LOCK TABLES `juegos` WRITE;
INSERT INTO `juegos` VALUES (1,'BaldursGate','/images/games/bg3.png',1),(2,'League of Legends','/images/games/Lol.png',2),(3,'Valorant','/images/games/valorant.png',3),(4,'CSGO2','/images/games/csgo2.png',4),(5,'Rocket League','/images/games/RL.png',5),(6,'Fornite','/images/games/fornite.png',6),(7,'Stardew Valley','/images/games/stardew.png',7),(8,'Hell Divers','/images/games/HellDivers.png',8),(9,'Apex Legends','/images/games/Apex.png',9),(10,'Red Dead Redemption 2','/images/games/rdr2.png',10),(11,'Sea of Thieves','/images/games/SoT.png',11),(12,'Boorderlands 3','/images/games/B3.png',12),(13,'Diablo IV','/images/games/DiabloIV.png',13),(14,'Left 4 dead 2','/images/games/l4d2.png',14),(999,'Search games','/images/games/search.png',999);
UNLOCK TABLES;


DROP TABLE IF EXISTS `msgs`;
CREATE TABLE `msgs` (
  `id` bigint NOT NULL AUTO_INCREMENT,
  `post_id` bigint DEFAULT NULL,
  `usuario_id` bigint DEFAULT NULL,
  `chat_id` bigint DEFAULT NULL,
  `texto` varchar(255) NOT NULL,
  `time_stamp` timestamp NULL DEFAULT NULL,
  PRIMARY KEY (`id`),
  UNIQUE KEY `post_id` (`post_id`),
  KEY `usuario_id` (`usuario_id`),
  KEY `chat_id` (`chat_id`),
  CONSTRAINT `msgs_ibfk_1` FOREIGN KEY (`post_id`) REFERENCES `posts` (`id`) ON DELETE CASCADE,
  CONSTRAINT `msgs_ibfk_2` FOREIGN KEY (`usuario_id`) REFERENCES `usuarios` (`id`) ON DELETE CASCADE,
  CONSTRAINT `msgs_ibfk_3` FOREIGN KEY (`chat_id`) REFERENCES `chats` (`id`) ON DELETE CASCADE
) ENGINE=InnoDB AUTO_INCREMENT=56 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;

DROP TABLE IF EXISTS `posts`;
CREATE TABLE `posts` (
  `id` bigint NOT NULL AUTO_INCREMENT,
  `num_users` bigint DEFAULT NULL,
  `usuario_id` bigint DEFAULT NULL,
  `foro_id` bigint DEFAULT NULL,
  `tipo` varchar(255) DEFAULT NULL,
  PRIMARY KEY (`id`),
  KEY `usuario_id` (`usuario_id`),
  KEY `foro_id` (`foro_id`),
  CONSTRAINT `posts_ibfk_1` FOREIGN KEY (`usuario_id`) REFERENCES `usuarios` (`id`) ON DELETE CASCADE,
  CONSTRAINT `posts_ibfk_2` FOREIGN KEY (`foro_id`) REFERENCES `foros` (`id`) ON DELETE CASCADE
) ENGINE=InnoDB AUTO_INCREMENT=29 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;

DROP TABLE IF EXISTS `usuario_chat`;
CREATE TABLE `usuario_chat` (
  `id` bigint NOT NULL AUTO_INCREMENT,
  `usuario_id` bigint NOT NULL,
  `chat_id` bigint NOT NULL,
```

```
78    PRIMARY KEY (`id`),
79    KEY `chat_id` (`chat_id`),
80    KEY `usuario_chat_ibfk_1` (`usuario_id`),
81    CONSTRAINT `usuario_chat_ibfk_1` FOREIGN KEY (`usuario_id`)
          REFERENCES `usuarios` (`id`) ON DELETE CASCADE,
82    CONSTRAINT `usuario_chat_ibfk_2` FOREIGN KEY (`chat_id`)
          REFERENCES `chats` (`id`) ON DELETE CASCADE
83  ) ENGINE=InnoDB AUTO_INCREMENT=166 DEFAULT CHARSET=utf8mb4 COLLATE=
      utf8mb4_0900_ai_ci;
84
85  DROP TABLE IF EXISTS `usuario_juego`;
86  CREATE TABLE `usuario_juego` (
87    `usuario_id` bigint NOT NULL,
88    `juego_id` bigint NOT NULL,
89    PRIMARY KEY (`usuario_id`,`juego_id`),
90    KEY `juego_id` (`juego_id`),
91    CONSTRAINT `usuario_juego_ibfk_1` FOREIGN KEY (`usuario_id`)
          REFERENCES `usuarios` (`id`) ON DELETE CASCADE,
92    CONSTRAINT `usuario_juego_ibfk_2` FOREIGN KEY (`juego_id`)
          REFERENCES `juegos` (`id`) ON DELETE CASCADE
93  ) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
94
95  DROP TABLE IF EXISTS `usuarios`;
96  CREATE TABLE `usuarios` (
97    `id` bigint NOT NULL AUTO_INCREMENT,
98    `usuario` varchar(255) NOT NULL,
99    `email` varchar(255) NOT NULL,
100   `password_hash` varchar(255) NOT NULL,
101   `img` varchar(255) DEFAULT NULL,
102   PRIMARY KEY (`id`),
103   UNIQUE KEY `usuario` (`usuario`),
104   UNIQUE KEY `email` (`email`)
105 ) ENGINE=InnoDB AUTO_INCREMENT=26 DEFAULT CHARSET=utf8mb4 COLLATE=
      utf8mb4_0900_ai_ci;
```

CAPÍTULO 11
Anexo 2

En este apartado, se desarrolla la relación de este trabajo de fin de grado los con objetivos de desarrollo sostenibles que se muestran en la imagen 11.1

OBJETIVOS DE DESARROLLO SOSTENIBLE

Grado de relación del trabajo con los Objetivos de Desarrollo Sostenible (ODS).

Objetivos de Desarrollo Sostenibles	Alto	Medio	Bajo	No Procede
ODS 1. **Fin de la pobreza.**				x
ODS 2. **Hambre cero.**				x
ODS 3. **Salud y bienestar.**		x		
ODS 4. **Educación de calidad.**				x
ODS 5. Igualdad de género.			x	
ODS 6. **Agua limpia y saneamiento.**				x
ODS 7. Energía asequible y no contaminante.				x
ODS 8. **Trabajo decente y crecimiento económico.**				x
ODS 9. Industria, innovación e infraestructuras.			x	
ODS 10. **Reducción de las desigualdades.**			x	
ODS 11. **Ciudades y comunidades sostenibles.**				x
ODS 12. Producción y consumo responsables.				x
ODS 13. Acción por el clima.				x
ODS 14. Vida submarina.				x
ODS 15. **Vida de ecosistemas terrestres.**				x
ODS 16. **Paz, justicia e instituciones sólidas.**				x
ODS 17. **Alianzas para lograr objetivos.**				x

Figura 11.1: Tabla de *ODS's*.

Reflexión sobre la relación del TFG/TFM con los ODS y con el/los ODS más relacionados.

El trabajo de fin de grado se conecta fuertemente con el **ODS 3: Salud y bienestar**. La plataforma creada busca generar un entorno saludable, tanto a nivel mental como emocional, para los usuarios que comparten su pasión por los videojuegos. Al eliminar métricas de popularidad como los "likes" o "seguidores", se reduce la presión social, permitiendo que los usuarios interactúen sin la ansiedad que suelen generar las redes sociales tradicionales. Además, al no incluir micro-pagos ni publicidad, se crea un espacio libre de presiones comerciales, lo que facilita que los usuarios disfruten de una interacción genuina y sin distracciones. Esto no solo mejora su experiencia dentro de la plataforma, sino que también contribuye de manera directa a su bienestar mental, evitando problemas como la competitividad tóxica o la exclusión social.

Sin embargo, aunque el trabajo se relaciona claramente con el ODS 3, mantiene una vinculación indirecta con los **ODS: 5: Igualdad de género, 9: Industria, innovación e infraestructuras y 10: Reducción de las desigualdades**.

A pesar de que la plataforma es inclusiva y accesible para todos los usuarios, sin discriminar a nadie, su objetivo principal no es abordar específicamente la igualdad de género (**ODS 5**). No se enfoca en reducir las desigualdades de género en la industria tecnológica o en el mundo de los videojuegos, aunque ciertamente promueve un entorno respetuoso y equitativo.

Por otro lado, la relación con el **ODS 9** es más bien marginal. Aunque el desarrollo de la plataforma utiliza tecnologías modernas y sigue principios de innovación, su propósito no es directamente impulsar el crecimiento industrial o fomentar la innovación tecnológica a gran escala. En realidad, el enfoque principal es proporcionar una herramienta social para los aficionados a los videojuegos, por lo que su impacto en la industria o las infraestructuras es limitado.

En cuanto al **ODS 10**, aunque la plataforma promueve un espacio más equitativo para todos sus usuarios, la conexión es menos evidente. Si bien se eliminan las jerarquías basadas en la popularidad y se fomenta un entorno inclusivo, la plataforma no está específicamente diseñada para reducir las desigualdades sociales o económicas a gran escala. Por lo tanto, su contribución a este ODS es más indirecta.