ESCUELA DE INGENIERÍA INFORMÁTICA
TRABAJO DE FIN DE GRADO

GRADO EN INGENIERÍA INFORMÁTICA
Mención en Tecnologías de la Información

Actualización de una Plataforma de Juegos para el Aprendizaje Gamificado

---

Autor: Enrique Manuel Ortega Herguedas

Tutores:
Carlos Enrique Vivaracho Pascual, Luis Ignacio Jimenez Gil

A mis padres

Agradecimientos

Quiero agradecer a mi familia por todo su apoyo, paciencia y confianza.

De igual manera, quiero agradecer a mis amigos y compañeros por su apoyo e imprescindible ayuda durante estos años.

Finalmente, quiero agradecer a mis tutores académicos por su ayuda, directrices y por haber logrado convertir la realización de este proyecto en una valiosa experiencia.

Gracias por todo.

Resumen

Este documento presenta un Trabajo de Fin de Grado (TFG) que se centra en un proyecto que busca actualizar el front-end de la plataforma web Gamispace del grupo de investigación (GREIDI, UVa) de la Universidad de Valladolid (UVa), reemplazando la implementación previa en HTML y PHP por el framework React. Esta actualización se ha llevado a cabo con el objetivo de mejorar la mantenibilidad del código, optimizar el rendimiento y proporcionar una experiencia de usuario más dinámica e intuitiva.

Abstract

This document presents a Bachelor's Thesis (TFG) focused on a project aimed at updating the front-end of the Gamispace web platform, developed by the GREIDI research group at the University of Valladolid. The previous implementation in HTML and PHP has been replaced with the React framework. This update was carried out with the goal of improving code maintainability, optimizing performance, and providing a more dynamic and intuitive user experience.

Índice

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

1.1. Motivación

En la educación universitaria, cada vez es más común el uso de herramientas tecnológicas para mejorar el proceso de enseñanza y aprendizaje. En este contexto, un grupo de profesores de la Escuela de Ingeniería Informática de Valladolid desarrolló una plataforma web destinada a los estudiantes de diversas carreras universitarias, con el fin de incorporar gamificación en el aula. Dentro de esta plataforma, los estudiantes pueden acceder a una serie de juegos educativos diseñados para ciertas asignaturas de su plan de estudios; los estudiantes pueden aprender jugando y los docentes pueden recabar información acerca de cómo interactúan los alumnos con los juegos. Es decir, la plataforma tiene una doble función: servir para la innovación educativa basada en juegos serios y para la investigación en ese campo.

Sin embargo, la tecnología empleada en el desarrollo original de la plataforma ha quedado obsoleta, dificultando su mantenimiento, escalabilidad y adaptabilidad a los estándares actuales de diseño y experiencia de usuario. Para modificar las limitaciones y problemas observados tras varios cursos en uso, este trabajo de fin de grado tiene como objetivo principal rediseñar y modernizar completamente la plataforma utilizando tecnologías modernas, garantizando una solución eficiente, flexible y sostenible.

La plataforma actual fue desarrollada con HMTL y PHP en una arquitectura monolítica que presenta varias limitaciones técnicas y funcionales. Las tecnologías usadas dificultan la incorporación de nuevas funcionalidades y actualizaciones. Por otro lado, la arquitectura actual no facilita la escalabilidad ni la modularidad del sistema.

Además, la interfaz carece de elementos modernos que mejoren la interacción del usuario, así como su estética.

1.2. Objetivos

El objetivo principal de este Trabajo de Fin de Grado es actualizar el front-end o la parte de cliente de la plataforma web Gamispace mediante una nueva arquitectura

basada en tecnologías actuales, con el fin de mejorar su mantenibilidad, escalabilidad y experiencia de usuario. Para ello, se plantea rediseñar y modernizar la plataforma, migrando su implementación actual en HTML y PHP a un entorno basado en React y TypeScript, lo que permitirá optimizar su modularidad, rendimiento y usabilidad.

Con esta meta en mente se han definido una serie de objetivos específicos que se detallan a continuación.

- **Análisis del sistema existente**: Evaluar las limitaciones de la versión actual de la plataforma y definir los requisitos técnicos y funcionales de la nueva implementación.
- **Diseño de la nueva arquitectura**: Proponer una estructura basada en componentes reutilizables con React, asegurando una implementación escalable y mantenible.
- **Implementación del front-end**: Desarrollar la interfaz de usuario utilizando React y TypeScript, integrando bibliotecas para el manejo del estado, navegación y estilos.
- **Mejorar la experiencia de usuario**: Optimizar la usabilidad de la plataforma mediante un diseño responsivo y accesible.
- **Integración de funcionalidades clave**: Implementar la autenticación de usuarios, gestión de asignaturas y juegos, y visualización de clasificaciones.
- **Pruebas y validación**: Realizar pruebas de usabilidad y compatibilidad para garantizar el correcto funcionamiento de la plataforma en distintos dispositivos y navegadores.
- **Documentación del proyecto**: Registrar el proceso de actualización, las decisiones técnicas adoptadas y las pruebas realizadas para facilitar futuras mejoras y mantenimiento.

1.3. Estructura del Documento

Para garantizar una comprensión clara y ordenada del desarrollo del proyecto, este documento se ha estructurado en capítulos que siguen una secuencia lógica, abarcando desde la motivación inicial hasta las conclusiones y posibles líneas de mejora. Cada capítulo responde a una fase concreta del proyecto y permite al lector seguir el hilo conductor del trabajo de forma progresiva y coherente. Asimismo, se ha procurado que la estructura facilite tanto la lectura lineal como la consulta puntual de secciones específicas.

La organización del documento es la siguiente:

Capítulo 1: Introducción: Se presenta una visión general del proyecto, incluyendo los objetivos principales, la motivación y la metodología de trabajo adoptada.

Capítulo 2: Contexto: En este capítulo se analiza la situación actual de la plataforma existente, identificando sus limitaciones, y se realiza una revisión de productos similares que han servido como referencia para el diseño de la nueva solución.

Capítulo 3: Planificación: Se expone la planificación temporal del proyecto, el cronograma de desarrollo, los riesgos identificados, así como una estimación preliminar de los recursos necesarios.

Capítulo 4: Análisis: Se detallan los requisitos funcionales y no funcionales del sistema, el modelo de casos de uso y su descripción, y el modelo de dominio que servirá de base para el diseño posterior.

Capítulo 5: Diseño e Implementación: En este capítulo se documentan las decisiones técnicas tomadas, la arquitectura de la solución, los patrones utilizados, los diagramas representativos del sistema, los wireframes de la interfaz y los aspectos clave de la implementación. También se incluyen pruebas realizadas para validar la solución desarrollada.

Capítulo 6: Conclusiones: Se presentan las conclusiones del trabajo, destacando los logros alcanzados, las limitaciones encontradas y posibles líneas de mejora para futuras iteraciones del proyecto.

Capítulo 2

Contexto

2.1. Plataforma Actual

La plataforma actual GamiSpace está orientada al uso de videojuegos serios en el ámbito universitario. Su objetivo principal es facilitar al profesorado la incorporación de juegos educativos en sus asignaturas, sin necesidad de conocimientos técnicos avanzados. La plataforma no se centra en el desarrollo de juegos, sino en su gestión, distribución y análisis. Ha sido concebida como una herramienta abierta, configurable y compatible con múltiples asignaturas y usuarios. Su estructura permite la personalización de la experiencia docente y la recogida de datos sobre la participación y progreso del alumnado [**jenui2023**].

La plataforma ofrece un conjunto amplio de funcionalidades. Entre las más destacadas se encuentran:

- **Gestión de juegos:** Los docentes pueden añadir nuevos videojuegos educativos, modificarlos, ocultarlos temporalmente o eliminarlos según sus necesidades didácticas. Los juegos pueden agruparse en "mundos" temáticos para facilitar su organización y contextualización.
- **Configuración personalizada:** Cada docente puede decidir qué juegos estarán disponibles, en qué momento del curso y con qué configuración específica. Además, se puede controlar el acceso a cada juego y su visibilidad dentro de la plataforma.
- **Analítica de datos:** GamiSpace recopila automáticamente información sobre el uso de los juegos por parte de los estudiantes. Se registran aspectos como puntuaciones obtenidas, tiempo de juego, número de intentos, progreso y nivel de logro. Estos datos permiten realizar un seguimiento detallado y no intrusivo del aprendizaje.
- **Gamificación:** La plataforma permite diseñar competiciones tanto individuales como por equipos. Se generan clasificaciones que tienen en cuenta criterios como la cantidad de juegos completados, las puntuaciones alcanzadas y el tiempo empleado. Esta funcionalidad busca aumentar la motivación y el compromiso del alumnado.

- **Accesibilidad técnica:** Todos los juegos deben ser ejecutables desde el navegador web, lo que garantiza su compatibilidad con diferentes dispositivos y sistemas operativos. El entorno gráfico es personalizable y se adapta a las necesidades específicas del curso.
- **Soporte multiusuario:** La plataforma distingue entre tres perfiles de usuario: estudiantes (jugadores), docentes (gestores y analistas del contenido) y desarrolladores (creadores de nuevos juegos). Esta estructura facilita el trabajo colaborativo y la expansión de la herramienta.
- **Exportación y análisis externo:** Los datos recogidos pueden exportarse para su análisis mediante herramientas externas. Además, está previsto implementar un panel de control interactivo para visualizar métricas directamente desde la propia plataforma.

En el proceso de evolución tecnológica de la aplicación web original, se ha identificado como uno de los principales retos su arquitectura monolítica, la cual dificulta la escalabilidad, el mantenimiento y la integración de nuevas funcionalidades. Por ello, la nueva plataforma surge con el objetivo de abordar este problema, apostando por una separación clara entre el front-end y el back-end, alineándose con las buenas prácticas del desarrollo web moderno.

Esta separación permitirá, en el futuro, una mayor flexibilidad en el desarrollo e integración de servicios, facilitando la adopción de una arquitectura más modular y sostenible. No obstante, debido a limitaciones de tiempo y recursos, el presente proyecto se ha centrado exclusivamente en el rediseño y desarrollo de la parte frontal de la aplicación.

A pesar de esta limitación, se ha trabajado cuidadosamente para que el nuevo front-end esté completamente preparado para integrarse, en una fase posterior, con un back-end desacoplado. Para ello, se han seguido principios de diseño y estructuración que garantizan una fácil conexión futura con servicios API, asegurando así la continuidad y viabilidad del proyecto a medio y largo plazo.

2.2. Productos Similares

En la actualidad, existe una amplia variedad de plataformas digitales que permiten el acceso a juegos con fines educativos o recreativos, muchas de ellas basadas en tecnologías en la nube, lo que elimina la dependencia del hardware del usuario. Compañías como Facebook, Google, Amazon y Microsoft han desarrollado sus propios entornos de juego en esta línea.

No obstante, la mayoría de estas plataformas están orientadas a niveles educativos inferiores, como la educación primaria o secundaria, y no se ajustan a las necesidades de la educación superior. Además, las soluciones que sí están pensadas para niveles más avanzados suelen ser comerciales y restringen el acceso completo a sus contenidos. Herramientas como Minecraft EDU o Smile and Learn son ejemplos de ello.

Por otro lado, muchas de las plataformas existentes no ofrecen la posibilidad de integrar juegos nuevos o, si lo hacen, limitan la creación de estos a plantillas preestablecidas o a ciertos formatos específicos, como sucede con Wordwall, Educaplay o Kahoot. Aquellas que permiten recoger información sobre el rendimiento del jugador, como Socrative o Steam, lo hacen con un enfoque principalmente comercial o centrado en el análisis del propio juego, más que en su valor pedagógico.

El análisis del progreso del alumno dentro del juego es esencial para que los docentes puedan adaptar mejor sus metodologías, pero esta tarea suele requerir herramientas externas como encuestas, que resultan intrusivas y demandan un esfuerzo adicional tanto para profesores como para estudiantes. Como alternativa, han surgido enfoques menos invasivos como las evaluaciones implícitas que registran automáticamente los datos de uso del juego.

A pesar de que algunas investigaciones han utilizado plataformas comerciales para recoger datos de juego, el objetivo habitual ha sido mejorar la jugabilidad o experiencia de usuario, no la retroalimentación educativa. En cambio, la propuesta que aquí se presenta se diferencia claramente: busca recopilar información detallada del rendimiento individual de los estudiantes dentro de los juegos, permitiendo así un análisis más profundo del proceso de aprendizaje, no sólo del funcionamiento del juego.

La plataforma desarrollada combina funcionalidades que hasta ahora no habían coincidido en una sola herramienta: es accesible, permite la incorporación de cualquier tipo de juego siempre que este sea HTML, ofrece un control granular sobre su disponibilidad y, sobre todo, recopila datos específicos de cada jugador. Esto la convierte en una solución potente para aplicar analíticas del juego en contextos educativos universitarios.

2.3. Tecnología Principal

Para el desarrollo del nuevo front-end de la plataforma se ha optado por el uso de React [1], una biblioteca de JavaScript ampliamente adoptada en la industria para la construcción de interfaces de usuario dinámicas y eficientes. Esta decisión se ha tomado tras analizar distintas opciones como Vue.js y Angular, teniendo en cuenta factores como la curva de aprendizaje, la flexibilidad, la comunidad y el ecosistema de herramientas disponibles.

A diferencia de Angular, que es un framework completo y estructurado con una fuerte opinión sobre cómo debe organizarse el código, React proporciona una mayor libertad al desarrollador, al centrarse exclusivamente en la capa de vista (la interfaz de usuario) y permitir elegir de forma más libre el resto de las tecnologías a utilizar (enrutado, gestión de estado, etc.). Esto ofrece una mayor adaptabilidad a las necesidades concretas del proyecto.

En comparación con Vue, que también es ligero y fácil de integrar, React presenta una comunidad más amplia, una madurez más consolidada y una mayor adopción en entornos profesionales, lo que facilita encontrar documentación, librerías externas y soluciones a problemas comunes. Además, React está respaldado por Meta, lo que garantiza soporte a largo plazo y una evolución constante de la herramienta.

La elección de React se justifica también por su arquitectura basada en componentes reutilizables y por el uso del Virtual DOM, que permite actualizaciones rápidas y eficientes de la interfaz, mejorando el rendimiento en aplicaciones dinámicas. Asimismo, React se integra de forma natural con herramientas modernas de desarrollo como Vite, Webpack o TypeScript.

En resumen, React ha sido elegido por su equilibrio entre flexibilidad, rendimiento, comunidad y madurez tecnológica, convirtiéndolo en la opción más adecuada para construir un front-end robusto, escalable y preparado para integrarse en el futuro con un back-end desacoplado.

2.3.1. React

React es una biblioteca de JavaScript desarrollada por Meta (anteriormente Facebook) que se utiliza para construir interfaces de usuario, especialmente en aplicaciones web de una sola página (SPA). Su principal característica es permitir la creación de componentes reutilizables, lo que facilita el desarrollo y mantenimiento de interfaces complejas.

React se basa en el uso de un DOM virtual (Virtual DOM), una representación ligera del DOM real que permite actualizar solo los elementos necesarios de forma eficiente, mejorando así el rendimiento. Además, emplea un enfoque declarativo, lo que significa que los desarrolladores describen el estado de la interfaz y React se encarga de reflejar esos cambios en pantalla.

Otra de sus ventajas es que permite estructurar la interfaz en componentes independientes y encapsulados, cada uno con su propio estado y lógica. Esto favorece la reutilización de código, la organización del proyecto y la colaboración en equipos de desarrollo.

Gracias a su flexibilidad y amplio ecosistema, React se ha convertido en una de las tecnologías más populares para el desarrollo de aplicaciones web modernas.

En este proyecto, al combinarse con TypeScript [2], un superset de JavaScript que añade tipado estático, los archivos que contienen código de React utilizan la extensión .tsx. Esta extensión indica que el archivo incluye tanto lógica escrita en TypeScript como elementos JSX (una sintaxis similar a HTML que se usa para definir la estructura visual de los componentes). Esta combinación mejora la robustez, mantenibilidad y escalabilidad del desarrollo front-end.

2.4. Terminología de React

Este apartado tiene como objetivo facilitar la comprensión del contenido del trabajo, asegurar una lectura más fluida y ayudar al lector a contextualizar adecuadamente las tecnologías y herramientas utilizadas. Entender con claridad estos conceptos permite apreciar mejor tanto el funcionamiento de React como las decisiones técnicas adoptadas durante el desarrollo del proyecto.

Este glosario básico también ayudará a entender mejor los conceptos utilizados a lo largo del desarrollo del proyecto, especialmente en la lectura del código fuente.

- `.tsx`: Extensión utilizada para archivos de React que combinan TypeScript con JSX (una sintaxis que permite escribir elementos HTML dentro del código JavaScript/TypeScript).
- **Componente**: Unidad funcional reutilizable de interfaz en React. Puede ser de tipo función o clase, aunque el enfoque moderno es mediante funciones.
- **Hook**: Es una función especial que te permite acceder a las características de React (como el estado y el ciclo de vida) desde componentes funcionales
- **Custom Hook**: Hook personalizado creado por el desarrollador para reutilizar lógica entre componentes.
- **Estado**: Datos internos de un componente que puede cambiar con el tiempo y provocan que el componente se vuelva renderizar.

Capítulo 3

Planificación

3.1. Metodología de Trabajo

Para la planificación y desarrollo de este proyecto se ha optado por utilizar la metodología en cascada [3], un enfoque tradicional en la gestión de proyectos de software que se caracteriza por su estructura secuencial, donde cada fase debe completarse antes de iniciar la siguiente. Este modelo divide el proceso en varias etapas bien definidas: primero se realiza un análisis detallado de los requisitos funcionales y no funcionales del sistema, luego se lleva a cabo el diseño de la arquitectura y modelos de datos, seguido por la fase de implementación donde se desarrolla el código basado en las especificaciones previas. Posteriormente, se realizan pruebas para asegurar que el sistema cumple con los requisitos establecidos, y finalmente, se despliega la solución y se contempla el mantenimiento necesario. Se muestra un ejemplo de esta metodología en la Figura 3.1.

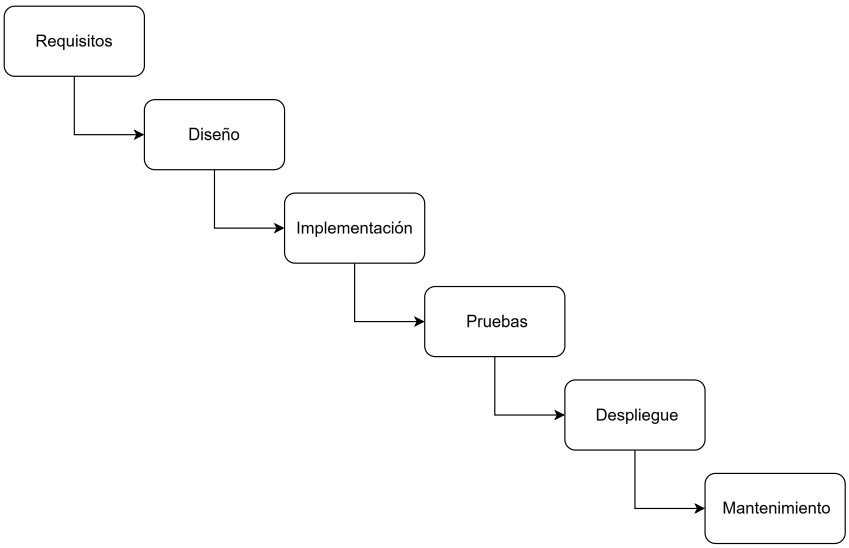

Figura 3.1: Diagrama Metodología en Cascada.

3.2. Riesgos

Para identificar, clasificar y valorar los riesgos del proyecto, se ha seguido un enfoque alineado con la metodología propuesta por el Instituto Nacional de Ciberseguridad (INCIBE), en su guía Análisis de riesgos en 5 pasos sencillos (2023)[4]. En dicha metodología, los riesgos se valoran en función de dos parámetros principales:

- **Probabilidad**: se refiere a la posibilidad de que ocurra un evento que afecte negativamente al proyecto. Se clasifica cualitativamente como baja, media o alta.
- **Impacto**: representa la magnitud de las consecuencias que tendría dicho evento si llegara a materializarse. También se clasifica en los niveles leve, moderado o crítico.

A partir de esta clasificación, se han identificado los principales riesgos del proyecto y se han establecido medidas de mitigación adecuadas. La Tabla 3.1 resume los riesgos detectados, su valoración y las acciones propuestas.

3.3. Cronograma del Proyecto

En esta sección, se detallan las fases del proyecto y su duración.

3.3.1. Planificación Estimada

En la Tabla 3.2 se puede ver la planificación estimada y en la Figura 3.2 el correspondiente diagrama de Gantt.

Fase	Tarea	Duración	Horas
1. Requisitos	Recopilación y especificación de requisitos, identificación de componentes y vistas y consideraciones de accesibilidad y usabilidad.	4 semanas	38h
2. Análisis	Análisis de casos de uso y flujo de usuario.	4 semanas	36h
3. Diseño	Diseño de la arquitectura del front-end, selección de tecnologías y diseño de wireframes.	4 semanas	40h
4. Implementación	Desarrollo de componentes UI y lógica de navegación.	12 semanas	120h
5. Pruebas	Pruebas de interfaz y validación de comportamiento con datos simulados.	3 semanas	30h
6. Documentación	Redacción de documentación del código, informe final y entrega	4 semanas	36h

Tabla 3.2: Fases del proyecto. Planificación Estimada.

ID	Riesgo	Probabilidad	Impacto	Descripción y Mitigación
R1	Falta de experiencia en React y TypeScript	Media	Moderado	El equipo puede encontrar dificultades al aprender nuevas tecnologías. Se mitiga con formación previa y consulta de documentación oficial.
R2	Problemas de compatibilidad con el back-end	Media	Crítico	Dado que el back-end sigue en desarrollo, pueden surgir incompatibilidades. Se mitiga con pruebas constantes y una comunicación fluida con el equipo back-end.
R3	Dificultad en la migración de datos	Alta	Moderado	Migrar datos del sistema antiguo al nuevo puede generar errores. Se mitiga con pruebas de migración y respaldos frecuentes.
R4	Desajuste en los tiempos de desarrollo	Media	Crítico	Si el desarrollo se retrasa, puede afectar la entrega. Se mitiga con una planificación realista y revisiones periódicas del cronograma.
R5	Problemas de rendimiento en la nueva plataforma	Baja	Crítico	La nueva implementación podría no ser eficiente. Se mitiga optimizando código y usando tecnologías potentes y modernas.
R6	Falta de disponibilidad de recursos o soporte técnico	Baja	Moderado	Pueden surgir problemas sin acceso a apoyo técnico. Se mitiga con documentación detallada.

Tabla 3.1: Tabla de riesgos y mitigación

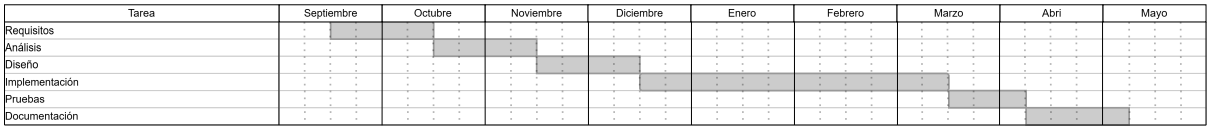

Figura 3.2: Diagrama de Gantt. Planificación Estimada.

3.3.2. Planificación Real

En la Tabla 3.3 se muestra el desarrollo final real del trabajo a realizar, con la correspondiente actualización del diagrama de Gantt en la Figura 3.3.

El proyecto ha experimentado un retraso acumulado de cinco semanas respecto a la planificación inicial. Las principales causas han sido la complejidad no prevista durante el análisis de los casos de uso y flujos, que requirió una comprensión más profunda del sistema actual y sus necesidades. En la fase de diseño del front-end, surgieron decisiones técnicas que requirieron más tiempo de evaluación para garantizar una arquitectura es-

calable y sostenible. La implementación también se extendió debido a ajustes continuos, pruebas y resolución de problemas derivados del desarrollo incremental. Finalmente, la redacción del informe tomó más tiempo de lo esperado por la necesidad de documentar detalladamente cada etapa y justificar decisiones técnicas clave.

Con respecto a la fase de pruebas, el proceso resultó ser notablemente más corto que el estipulado en la planificación estimada.

Fase	**Tarea**	**Duración**	**Horas**
1. Requisitos	Recopilación y especificación de requisitos, identificación de componentes y vistas, consideraciones de accesibilidad y usabilidad.	4 semanas	38h
2. Análisis	Análisis de casos de uso y flujo de usuario. Revisión y ampliación tras ajustes en el diseño.	5 semanas	45h
3. Diseño	Diseño de la arquitectura del front-end, selección de tecnologías y diseño de wireframes. Se retomaron decisiones en función de los ajustes de análisis.	5 semanas	42h
4. Implementación	Desarrollo de componentes UI y lógica de navegación. La duración aumentó por la curva de aprendizaje con React y TypeScript.	14 semanas	140h
5. Pruebas	Pruebas de interfaz y validación de comportamiento con datos simulados.	3 semanas	20h
6. Documentación	Redacción de documentación del código, informe final y entrega. Se amplió debido a la complejidad y cantidad de componentes desarrollados.	5 semanas	40h

Tabla 3.3: Fases del Proyecto. Planificación Real.

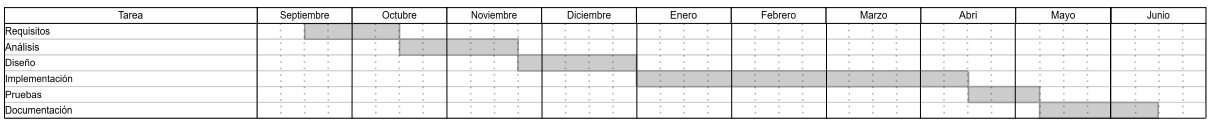

Figura 3.3: Diagrama de Gantt. Planificación Real.

3.4. Costes Aproximados

En este apartado se realiza una estimación de los costes aproximados que habría supuesto el desarrollo del proyecto, si hubiese sido llevado a cabo por una empresa. Para ello, se consideran los gastos asociados al personal necesario (desarrolladores, diseñadores, testers y gestores de proyecto), así como los costes de infraestructura, herramientas y servicios utilizados durante el proceso de desarrollo. El objetivo es ofrecer una visión rea-

lista del valor económico del proyecto, basada en tarifas de mercado y en la planificación seguida.

3.4.1. Costes de Personal

El principal gasto en un proyecto de software es el equipo de desarrollo.

El equipo consta de los siguientes integrantes; los salarios indicados están en bruto [5]:

- Desarrollador Full Stack (40000 €/año)
- Desarrollador UI/UX (37000 €/año)
- Tester QA (36000 €/año)
- Proyect Manager (35000 €/año)

Los costes aproximados de personal se pueden ver en la Tabla 3.4.

Fase	Duración (semanas)	Desarrollador	Diseñador UI/UX	Tester QA	Project Manager	Coste Total (€)
Análisis	4	0 h	160 h	0 h	160 h	5.645,60 €
Diseño	4	0 h	160 h	0 h	160 h	5.645,60 €
Implementación	12	480 h	80 h	0 h	240 h	13.855,60 €
Integración	4	160 h	0 h	80 h	80 h	6.290,80 €
Pruebas	3	60 h	0 h	120 h	60 h	4.635,60 €
Documentación	4	40 h	0 h	40 h	160 h	4.534,40 €
Total	31 semanas	740 h	400 h	240 h	860 h	**40.607,60 €**

Tabla 3.4: Costes de personal actualizada según salarios anuales

3.4.2. Coste de Infraestructura y Herramientas

Se consideran los gastos de servidores, hosting APIs, herramientas de diseño y pruebas, en la Tabla 3.5 se muestran los costes relacionados.

Concepto	Tecnología elegida	Coste anual (€)
Dominio Web (1 año)	Namecheap / GoDaddy	15 €
Hosting (front-end)	AWS S3 + CloudFront	600 €
Servidor back-end	AWS EC2 (t3a.medium)	1.200 €
Base de datos	Amazon RDS (MySQL)	600 €
Herramientas de diseño	Figma Professional	300 €
Herramientas de desarrollo	GitHub Team Plan	252 €
Total		**2.967 €**

Tabla 3.5: Costes anuales de infraestructura y herramientas

Capítulo 4

Análisis

En este capítulo se define y estructura el análisis previo necesario para la actualización de la plataforma Gamispace. Para garantizar una implementación efectiva, es fundamental comprender los requisitos del sistema, tanto a nivel funcional como no funcional, y establecer una base sólida para el diseño e implementación del proyecto.

En primer lugar, se detallarán los requisitos funcionales, que describen las características y funcionalidades esenciales que debe ofrecer la plataforma. A continuación, se abordarán los requisitos no funcionales, que establecen criterios de calidad, rendimiento, seguridad y usabilidad.

Además, se presentará el modelo de casos de uso, que refleja la interacción entre los usuarios y el sistema, permitiendo visualizar las diferentes acciones que pueden realizar dentro de la plataforma. Por último, se definirá el modelo de dominio, que identifica los principales conceptos y entidades involucrados en el desarrollo, proporcionando una visión estructurada de los datos y su relación dentro del sistema.

Este análisis servirá como base para la toma de decisiones durante la fase de diseño e implementación, asegurando que la actualización de Gamispace se realice de manera estructurada y alineada con los objetivos del proyecto.

4.1. Requisitos Funcionales

El sistema será utilizado por distintos tipos de usuarios:

- Usuario Jugador
- Usuario Invitado
- Usuario Profesor
- Usuario Desarrollador
- Usuario Administrador

En este sistema, las asignaturas y los juegos pueden encontrarse en diferentes estados:

- Estado Abierto/Cerrado: Este estado decide si el usuario puede acceder o no al elemento.
- Estado Visible/Oculto: Este estado decide si el usuario puede ver o no el elemento.

4.1.1. Requisitos Funcionales Front-End

En las tablas que se muestran a continuación aparecen los requisitos funcionales recogidos tras diversas iteraciones con el cliente (los profesores involucrados en la actividad).

RF-Front-01: Autenticación
El sistema debe permitir a cualquier usuario iniciar sesión a través de un formulario.

RF-Front-02: Diferenciación de Usuarios
El sistema debe poder diferenciar entre cinco tipos de usuario diferentes, cuyas funcionalidades son distintas: 'Jugador', 'Invitado', 'Administrador', 'Desarrollador' y 'Profesor'.

RF-Front-03: Acceder Asignatura
El sistema debe mostrar a todo usuario la lista de asignaturas habilitadas para él y permitir acceder a cada una de ellas mostrando los juegos que contienen.

RF-Front-04: Acceder Juego
El sistema debe permiter a todo usuario acceder a los juegos y jugar a ellos.

RF-Front-05: Mostrar Clasificaciones
El sistema debe mostrar al usuario las clasificaciones con puntuación y estrellas obtenidas.

RF-Front-06: Mostrar Progreso
El sistema debe mostrar al usuario información sobre su evolución en los juegos: cuántos ha completado y cuál es su puntuación y estrellas actuales.

RF-Front-07: Crear Juego
El sistema debe permitir al usuario 'Desarrollador' y 'Administrador' crear un nuevo juego.

RF-Front-08: Editar Datos de Juego
El sistema debe permitir al usuario 'Desarrollador' y 'Administrador' modificar los datos de un juego.

RF-Front-09: Editar Estado de Juego
El sistema debe permitir al usuario 'Desarrollador' y 'Administrador' modificar el estado de un juego. También se le debe permitir modificar el estado de visibilidad al usuario 'Profesor' pero exclusivamente sobre sus juegos.

RF-Front-10: Crear Usuario
El sistema debe permitir al usuario 'Administrador' crear un nuevo usuario.

RF-Front-11: Editar Datos de Usuario
El sistema debe permitir al usuario 'Administrador' modificar los datos de un usuario.

RF-Front-12: Crear Asignatura
El sistema debe permitir al usuario 'Administrador' crear una nueva asignatura.

RF-Front-13: Editar Datos de Asignatura
El sistema debe permitir al usuario 'Administrador' modificar los datos de una asignatura.

RF-Front-14: Editar Estado de Asignatura
El sistema debe permitir al usuario 'Administrador' modificar los datos de una asignatura. También se le debe permitir modificar el estado de visibilidad al usuario 'Profesor' pero exclusivamente sobre sus asignaturas.

RF-Front-15: Exportar Clasificaciones
El sistema debe permitir al usuario 'Administrador' exportar los datos de las clasificaciones a un fichero externo.

RF-Front-16: Crear Tema
El sistema debe permitir al usuario 'Administrador' crear un nuevo tema.

4.1.2. Requisitos Funcionales Back-End

Aunque cae fuera de los objetivos de este trabajo de fin de grado, la herramienta desarrollada ha sido realizada teniendo siempre en cuenta la parte del back-end de la plataforma. Cuando se implemente un back-end en el futuro, este debería cumplir como mínimo con estos requisitos.

RF-Back-01: Guardar Puntuación
El sistema debe guardar la puntuación del usuario cada vez que juegue.

RF-Back-02: No mostrar Puntuación en Clasificaciones
El sistema debe permitir al usuario 'Desarrollador' o 'Administrador' acceder a sus juegos y a la plataforma, pero sin incluir sus datos en las clasificaciones.

RF-Back-03: Múltiples Asignaturas
El sistema debe gestionar múltiples asignaturas simultáneamente, asociando a cada una sus usuarios y juegos.

RF-Back-04: Credenciales 'Jugador'
El sistema debe permitir que un usuario jugador pertenezca a varias asignaturas con las mismas credenciales, gestionando su acceso de manera unificada.

RF-Back-05: Compartir Juegos
El sistema debe permitir que un mismo juego sea compartido por varias asignaturas.

RF-Back-06: Autenticación de Usuario
El sistema debe implementar la autenticación de usuario con nombre de usuario y contraseña, debe redirigir a cada usuario a las vistas y funcionalidades correspondientes a su rol.

4.2. Requisitos no Funcionales

A continuación se describen los requisitos no funcionales del sistema, los cuales definen criterios de calidad que deben cumplirse durante su desarrollo.

RNF-01: FrontEnd en React y TypeScript
La interfaz de usuario debe ser implementada utilizando React con TypeScript y debe seguir los principios de diseño responsivo.

RNF-02: BackEnd en Node.js con Express
El backend debe implementarse con Node.js y el framework Express para gestionar la lógica del servidor y la API de comunicación.

RNF-03: Base de datos en MySQL
La plataforma debe utilizar el gestor de bases de datos MySQL para almacenar la información de usuarios, juegos, partidas y configuración de la plataforma.

RNF-04: Cifrado de Contraseñas
Las contraseñas de los usuarios deben almacenarse de forma segura mediante el mecanismo de cifrado robusto bcrypt.

RNF-05: Uso de JWT para autenticación
La autenticación debe basarse en JSON Web Tokens (JWT) para el manejo seguro de sesiones de usuario en la plataforma..

RNF-06: Escalabilidad
La plataforma debe ser escalable para soportar un crecimiento en el número de usuarios y juegos sin comprometer el rendimiento

4.2.1. Requisitos de Datos

En este apartado se detallan los requisitos de datos del sistema, especificando la información que debe ser almacenada, procesada y gestionada.

RD-01: Datos de Usuario
El sistema debe guardar los datos básicos de usuario, a saber: ID, grupo, tipo de usuario, nombre y contraseña.

RD-02: Datos de Asignatura
El sistema debe guardar los datos de las asignaturas, a saber: ID, nombre, URL de la imagen de asignatura, URL de la imagen de fondo, posición, estado 'abierto/cerrado', estado 'visible/oculto'.

RD-03: Datos de Juego
El sistema debe guardar los datos de los juegos, a saber: ID, ID de asignatura a la que pertenecen, URL de la imagen del juego, nombre, puntuación máxima obtenible, estado 'abierto/cerrado', estado 'visible/oculto', posición, ID del usuario que lo ha subido, estado 'nuevo' y estado 'subido'.

RD-04: Datos de Partida de Usuario
El sistema debe mantener un registro de las partidas de los usuarios, guardando: ID, ID del usuario, ID del juego, puntuación, tiempo, un estado completado/no completado, una fecha y hora de comienzo y fecha y hora de finalización.

RD-05: Datos de Evolución en Juego
El sistema debe mantener un registro de la evolución del usuario en cada juego, guardando: ID, ID del usuario, ID del juego, fecha y hora y un campo de texto que guardará en un formato prefijado información sobre los niveles jugados y puntuación obtenida en cada intento.

RD-06: Datos de Tema
El sistema debe guardar los datos del tema de la plataforma, a saber: ID del tema, color primario, color secundario, color de texto, icono de puntuación e icono de asignaturas completadas.

4.3. Modelo de Casos de Uso

En la Figura 4.1 se presentan los casos de uso encontrados para el sistema.

Figura 4.1: Diagrama de Casos de Uso.

La descripción detallada de cada caso de uso se muestra en la tablas 4.1 a 4.23.

CU-01: Iniciar Sesión

Descripción:
Autenticar al usuario en la plataforma, otorgando acceso a las vistas y funcionalidades correspondientes a su rol.

Actor Primario:
Usuario (Jugador, Invitado, Profesor, Desarrollador o Administrador).

Precondiciones:

- El usuario debe tener una cuenta registrada en el sistema.
- La conexión al servidor debe estar activa para validar las credenciales.

Flujo Principal:

1. El usuario solicita acceder a la plataforma.
2. El sistema solicita el nombre de usuario y la contraseña.
3. El usuario introduce el nombre de usuario y la contraseña.
4. El sistema valida los datos ingresados:
   a) Comprueba si el nombre de usuario existe en la base de datos.
   b) Verifica si la contraseña ingresada coincide con la contraseña cifrada almacenada.
5. Si las credenciales son correctas, el sistema genera un **JSON Web Token (JWT)** y lo almacena en el cliente (cookie o localStorage).
6. El sistema redirige al usuario a la vista Home, donde se muestran las asignaturas asociadas al usuario.

Flujo Alternativo A1 (Error de autenticación):

A. Si el nombre de usuario no existe o la contraseña no coincide, el sistema:
   - Muestra un mensaje de error indicando que las credenciales son inválidas.
   - Salta al paso dos.

Postcondiciones:

- Si las credenciales son válidas, el usuario queda autenticado y puede acceder a las funcionalidades según su rol.
- Si las credenciales son inválidas, el usuario permanece en la vista de inicio de sesión.

Tabla 4.1: Caso de Uso CU-01: Iniciar Sesión

CU-02: Cerrar Sesión
Descripción: Cerrar sesión en la plataforma
Actor Primario: Usuario (Jugador, Invitado, Profesor, Desarrollador o Administrador).
Precondiciones: ▪ El usuario debe estar autenticado en la plataforma.
Flujo Principal: 1. El usuario solicita cerrar sesión en la plataforma. 2. El sistema cierra la sesión. 3. El sistema redirige al usuario a la vista Login
Postcondiciones: ▪ El usuario ya no está autenticado en la plataforma. ▪ El usuario se encuentra en la vista Login

Tabla 4.2: Caso de Uso CU-02: Cerrar Sesión

CU-03: Acceder Asignatura
Descripción: Se accede a un asignatura y se muestran sus juegos correspondientes.
Actor Primario: Usuario (Jugador, Invitado, Profesor, Desarrollador o Administrador).
Precondiciones: - El usuario debe estar autenticado en la plataforma.
Flujo Principal: 1. El usuario solicita acceder a una asignatura. 2. El sistema comprueba si la asignatura está abierta. 3. El sistema redirige al usuario a la vista GameSelector, donde se muestran los juegos asociados a la asignatura seleccionada.
Flujo Alternativo A1 (Asignatura cerrada): A. Si el estado de la asignatura es cerrado, el sistema: - Muestra que la asignatura no es accesible. - Salta al paso uno.
Postcondiciones: - El usuario se encuentra en la vista GameSelector.

Tabla 4.3: Caso de Uso CU-03: Acceder Asignatura

CU-04: Acceder Juego
Descripción: Se accede a un juego y se este se carga.
Actor Primario: Usuario (Jugador, Invitado, Profesor, Desarrollador o Administrador).
Precondiciones: ▪ El usuario debe estar autenticado en la plataforma.
Flujo Principal: 1. El usuario solicita acceder a un juego. 2. El sistema comprueba si el juego está abierto. 3. El sistema redirige al usuario a la vista Play, donde se carga el juego seleccionado.
Flujo Alternativo A1 (Juego cerrado): A. Si el estado del juego es cerrado, el sistema: ▪ Muestra que el juego no es accesible. ▪ Salta al paso uno.
Postcondiciones: ▪ El usuario se encuentra en la vista Play.

Tabla 4.4: Caso de Uso CU-04: Acceder Juego

CU-05: Acceder Ranking
Descripción: Se accede al ranking, permite visualizarlo por jugador, por equipo, por jugador en base a juego y por grupo en base a juego.
Actor Primario: Usuario (Jugador, Invitado, Profesor, Desarrollador o Administrador).
Precondiciones: ▪ El usuario debe estar autenticado en la plataforma.
Flujo Principal: 1. El usuario solicita acceder al ranking. 2. El sistema redirige al usuario a la vista Ranking. 3. El sistema muestra los datos del ranking, a saber: nombre de jugador, nombre de grupo, puntuacion total y número de asignaturas completadas.
Postcondiciones: ▪ El usuario se encuentra en la vista Ranking.

Tabla 4.5: Caso de Uso CU-05: Acceder Ranking

CU-06: Crear Asignatura
Descripción: Se crea una nueva asignatura.
Actor Primario: Usuario Administrador.
Precondiciones: ▪ El usuario debe estar autenticado en la plataforma.
Flujo Principal: 1. El administrador solicita crear una nueva asignatura. 2. El sistema solicita los datos de la nueva asignatura, a saber: nombre, imagen, imagen de fondo, puntuación máxima. 3. El usuario introduce los datos y confirma. 4. El sistema comprueba que todos los datos se han rellenado. 5. El sistema crea la nueva asignatura.
Flujo Alternativo A1 (Datos sin rellenar): A. Si el usuario deja algún campo vacío al rellenar los datos de la nueva asignatura, el sistema: ▪ Muestra un mensaje de campos requeridos. ▪ Salta al paso dos.
Postcondiciones: ▪ La asignatura ha sido creada en el sistema.

Tabla 4.6: Caso de Uso CU-06: Crear Asignatura

CU-07: Editar Asignatura
Descripción: Se editan los datos de una asignatura.
Actor Primario: Usuario Administrador.
Precondiciones: ▪ El usuario debe estar autenticado en la plataforma.
Flujo Principal: 1. El administrador solicita editar una asignatura. 2. El sistema solicita los datos de la asignatura, a saber: nombre, imagen, imagen de fondo, puntuación máxima. 3. El usuario introduce los datos y confirma. 4. El sistema comprueba que todos los datos se han rellenado. 5. El sistema edita la asignatura con los nuevos datos.
Flujo Alternativo A1 (Datos sin rellenar): A. Si el usuario deja algún campo vacío al rellenar los datos de la nueva asignatura, el sistema: ▪ Muestra un mensaje de campos requeridos. ▪ Salta al paso dos.
Postcondiciones: ▪ La asignatura ha sido editada.

Tabla 4.7: Caso de Uso CU-07: Editar Asignatura

CU-08: Eliminar Asignatura
Descripción: Se elimina una asignatura.
Actor Primario: Usuario Administrador.
Precondiciones: ▪ El usuario debe estar autenticado en la plataforma.
Flujo Principal: 1. El administrador solicita eliminar una asignatura. 2. El sistema solicita confirmación. 3. El usuario confirma. 4. El sistema elimina la asignatura.
Postcondiciones: ▪ La asignatura ha sido eliminada del sistema.

Tabla 4.8: Caso de Uso CU-08: Eliminar Asignatura

CU-09: Abrir Asignatura
Descripción: Se cambia el estado de una asignatura a Abierto.
Actor Primario: Usuario Administrador.
Precondiciones: ▪ El usuario debe estar autenticado en la plataforma.
Flujo Principal: 1. El administrador solicita abrir una asignatura. 2. El sistema cambia el estado de la asignatura a Abierto.
Postcondiciones: ▪ El estado de la asignatura ha cambiado a Abierto.

Tabla 4.9: Caso de Uso CU-09: Abrir Asignatura

CU-10: Cerrar Asignatura
Descripción: Se cambia el estado de una asignatura a Cerrado.
Actor Primario: Usuario Administrador.
Precondiciones: ▪ El usuario debe estar autenticado en la plataforma.
Flujo Principal: 1. El administrador solicita cerrar una asignatura. 2. El sistema cambia el estado de la asignatura a Cerradp.
Postcondiciones: ▪ El estado de la asignatura ha cambiado a Cerrado.

Tabla 4.10: Caso de Uso CU-10: Cerrar Asignatura

CU-11: Ocultar Asignatura
Descripción: Se cambia el estado de una asignatura a Oculto.
Actor Primario: Usuario Administrador.
Precondiciones: ▪ El usuario debe estar autenticado en la plataforma.
Flujo Principal: 1. El administrador solicita ocultar una asignatura. 2. El sistema cambia el estado de la asignatura a Oculto.
Postcondiciones: ▪ El estado de la asignatura ha cambiado a Oculto.

Tabla 4.11: Caso de Uso CU-11: Ocultar Asignatura

CU-12: Mostrar Asignatura
Descripción: Se cambia el estado de una asignatura a Visible.
Actor Primario: Usuario Administrador.
Precondiciones: ▪ El usuario debe estar autenticado en la plataforma.
Flujo Principal: 1. El administrador solicita mostrar una asignatura. 2. El sistema cambia el estado de la asignatura a Visible.
Postcondiciones: ▪ El estado de la asignatura ha cambiado a Visible.

Tabla 4.12: Caso de Uso CU-12: Mostrar Asignatura

CU-13: Crear Juego
Descripción: Se crea un nuevo juego.
Actor Primario: Usuario Administrador.
Precondiciones: ▪ El usuario debe estar autenticado en la plataforma.
Flujo Principal: 1. El administrador solicita crear un nuevo juego. 2. El sistema solicita los datos del nuevo juego, a saber: nombre, imagen, puntuación máxima y asignatura al a que pertenece. 3. El usuario introduce los datos y confirma. 4. El sistema comprueba que todos los datos se han rellenado. 5. El sistema crea el nuevo juego.
Flujo Alternativo A1 (Datos sin rellenar): A. Si el usuario deja algún campo vacío al rellenar los datos del nuevo juego, el sistema: ▪ Muestra un mensaje de campos requeridos. ▪ Salta al paso dos.
Postcondiciones: ▪ El juego ha sido creado en el sistema.

Tabla 4.13: Caso de Uso CU-13: Crear Juego

CU-14: Editar Juego
Descripción: Se editan los datos de un juego.
Actor Primario: Usuario Administrador y Desarrollador.
Precondiciones: ▪ El usuario debe estar autenticado en la plataforma.
Flujo Principal: 1. El administrador solicita editar un juego. 2. El sistema solicita los datos del juego, a saber: nombre, imagen, puntuación máxima y asignatura al a que pertenece. 3. El usuario introduce los datos y confirma. 4. El sistema comprueba que todos los datos se han rellenado. 5. El sistema edita los datos del juego.
Flujo Alternativo A1 (Datos sin rellenar): A. Si el usuario deja algún campo vacío al rellenar los datos del juego, el sistema: ▪ Muestra un mensaje de campos requeridos. ▪ Salta al paso dos.
Postcondiciones: ▪ El juego ha sido editado.

Tabla 4.14: Caso de Uso CU-14: Editar Juego

CU-15: Eliminar Juego
Descripción: Se elimina un juego.
Actor Primario: Usuario Administrador.
Precondiciones: ▪ El usuario debe estar autenticado en la plataforma.
Flujo Principal: 1. El administrador solicita eliminar un juego. 2. El sistema solicita confirmación. 3. El usuario confirma. 4. El sistema elimina el juego.
Postcondiciones: ▪ El juego ha sido eliminada del sistema.

Tabla 4.15: Caso de Uso CU-15: Eliminar Juego

CU-16: Abrir Juego
Descripción: Se cambia el estado de un juego a Abierto.
Actor Primario: Usuario Administrador.
Precondiciones: ▪ El usuario debe estar autenticado en la plataforma.
Flujo Principal: 1. El administrador solicita abrir un juego. 2. El sistema cambia el estado del juego a Abierto.
Postcondiciones: ▪ El estado del juego ha cambiado a Abierto.

Tabla 4.16: Caso de Uso CU-16: Abrir Juego

CU-17: Cerrar Juego
Descripción: Se cambia el estado de un juego a Cerrado.
Actor Primario: Usuario Administrador.
Precondiciones: ▪ El usuario debe estar autenticado en la plataforma.
Flujo Principal: 1. El administrador solicita cerrar un juego. 2. El sistema cambia el estado del juego a Cerradp.
Postcondiciones: ▪ El estado del juego ha cambiado a Cerrado.

Tabla 4.17: Caso de Uso CU-17: Cerrar Juego

CU-18: Ocultar Juego
Descripción: Se cambia el estado de un juego a Oculto.
Actor Primario: Usuario Administrador.
Precondiciones: ▪ El usuario debe estar autenticado en la plataforma.
Flujo Principal: 1. El administrador solicita ocultar un juego. 2. El sistema cambia el estado del juego a Oculto.
Postcondiciones: ▪ El estado del juego ha cambiado a Oculto.

Tabla 4.18: Caso de Uso CU-18: Ocultar Juego

CU-19: Mostrar Juego
Descripción: Se cambia el estado de un juego a Visible.
Actor Primario: Usuario Administrador.
Precondiciones: ▪ El usuario debe estar autenticado en la plataforma.
Flujo Principal: 1. El administrador solicita mostrar un juego. 2. El sistema cambia el estado del juego a Visible.
Postcondiciones: ▪ El estado del juego ha cambiado a Visible.

Tabla 4.19: Caso de Uso CU-19: Mostrar Juego

CU-20: Crear Usuario
Descripción: Se crea un nuevo usuario.
Actor Primario: Usuario Administrador.
Precondiciones: ▪ El usuario debe estar autenticado en la plataforma.
Flujo Principal: 1. El administrador solicita crear un nuevo usuario. 2. El sistema solicita los datos del nuevo usuario, a saber: nombre, contraseña, grupo y rol. 3. El usuario introduce los datos y confirma. 4. El sistema comprueba que todos los datos se han rellenado. 5. El sistema crea el nuevo usuario.
Flujo Alternativo A1 (Datos sin rellenar): A. Si el usuario deja algún campo vacío al rellenar los datos del nuevo usuario, el sistema: ▪ Muestra un mensaje de campos requeridos. ▪ Salta al paso dos.
Postcondiciones: ▪ El usuario ha sido creado en el sistema.

Tabla 4.20: Caso de Uso CU-20: Crear Usuario

CU-21: Editar Usuario
Descripción: Se editan los datos de un usuario.
Actor Primario: Usuario Administrador.
Precondiciones: ▪ El usuario debe estar autenticado en la plataforma.
Flujo Principal: 1. El administrador solicita editar un usuario. 2. El sistema solicita los datos del usuario, a saber: nombre, contraseña, grupo y rol. 3. El usuario introduce los datos y confirma. 4. El sistema comprueba que todos los datos se han rellenado. 5. El sistema edita los datos del usuario.
Flujo Alternativo A1 (Datos sin rellenar): A. Si el usuario deja algún campo vacío al rellenar los datos del usuario, el sistema: ▪ Muestra un mensaje de campos requeridos. ▪ Salta al paso dos.
Postcondiciones: ▪ El usuario ha sido editado.

Tabla 4.21: Caso de Uso CU-21: Editar Usuario

CU-22: Eliminar Usuario
Descripción: Se elimina un usuario.
Actor Primario: Usuario Administrador.
Precondiciones: ▪ El usuario debe estar autenticado en la plataforma.
Flujo Principal: 1. El administrador solicita eliminar un usuario. 2. El sistema solicita confirmación. 3. El usuario confirma. 4. El sistema elimina el usuario.
Postcondiciones: ▪ El usuario ha sido eliminada del sistema.

Tabla 4.22: Caso de Uso CU-22: Eliminar usuario

CU-23: Crear Tema
Descripción: Se crea un nuevo tema.
Actor Primario: Usuario Administrador.
Precondiciones: ▪ El usuario debe estar autenticado en la plataforma.
Flujo Principal: 1. El administrador solicita crear un nuevo tema. 2. El sistema solicita los datos del nuevo tema, a saber: color primario, color secundario, color del texto, icono de puntuación e icono de asignaturas completadas. 3. El usuario introduce los datos y confirma. 4. El sistema comprueba que todos los datos se han rellenado. 5. El sistema crea el nuevo tema.
Flujo Alternativo A1 (Datos sin rellenar): A. Si el usuario deja algún campo vacío al rellenar los datos del nuevo tema, el sistema: ▪ Muestra un mensaje de campos requeridos. ▪ Salta al paso dos.
Postcondiciones: ▪ El tema ha sido creado en el sistema.

Tabla 4.23: Caso de Uso CU-23: Crear tema

4.3.1. Modelo de Dominio

En este apartado se describe el modelo de dominio del sistema.

El modelo de dominio presentado está compuesto por tres entidades principales: User, Subject y Game, reflejando la estructura lógica del sistema y sus relaciones.

La entidad User representa a los usuarios de la plataforma. Cada usuario tiene un identificador, un nombre, un rol (por ejemplo, estudiante o administrador), una puntuación total acumulada y un contador de asignaturas completadas. Los usuarios pueden estar relacionados con múltiples asignaturas, y a su vez, una asignatura puede pertenecer a varios usuarios, lo que refleja una relación muchos a muchos.

La entidad Subject representa las asignaturas disponibles. Cada asignatura tiene atributos como identificador, nombre, imágenes (principal y de fondo), posición, y dos indicadores booleanos: uno para indicar si está abierta y otro para su visibilidad. Cada asignatura contiene uno o más juegos, estableciendo una relación uno a muchos con la entidad Game.

La entidad Game representa los juegos asociados a cada asignatura. Cada juego tiene su identificador, nombre, imagen, puntuación máxima posible, posición, y una referencia al usuario que lo ha subido (idUser). Además, cuenta con varias banderas booleanas para definir su estado: si está abierto, visible, si es nuevo y si ha sido cargado correctamente. La relación con Subject es de muchos a uno, ya que un juego pertenece a una única asignatura.

En la Figura 4.2 se muestra el modelo de dominio descrito.

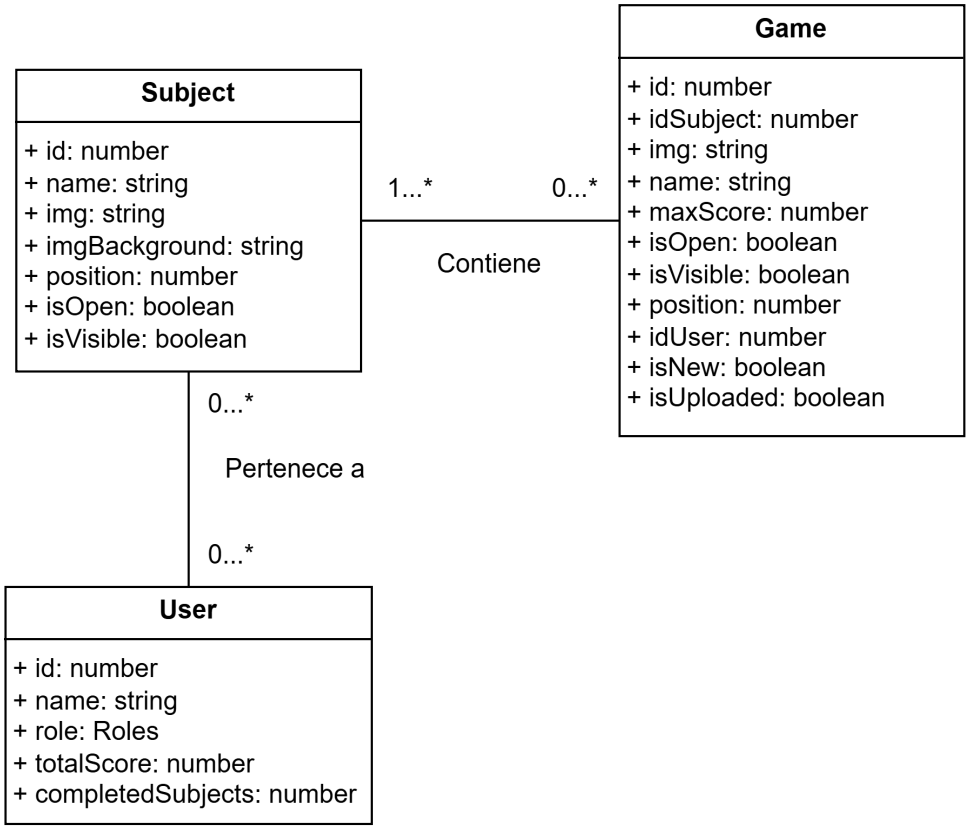

Figura 4.2: Modelo de Dominio.

4.3.2. Diagrama de Secuencia

Para entender cómo se comunicarán las diferentes partes del proyecto en un futuro se pone como ejemplo el diagrama de secuencia de Iniciar Sesión, este diagrama se ve en la Figura 4.3.

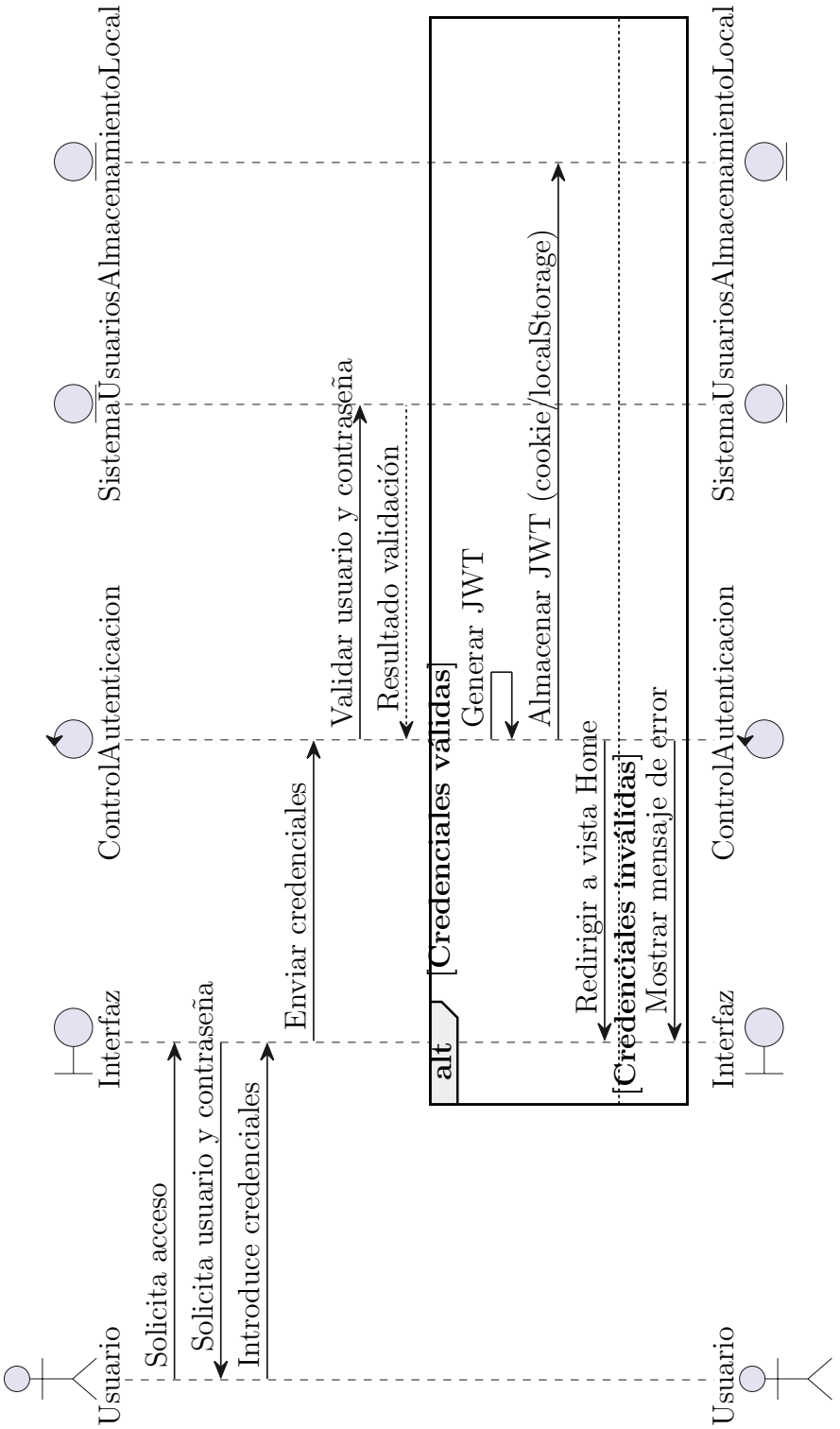

Figura 4.3: Diagrama de Secuencia de Iniciar Sesión.

Capítulo 5

Diseño e Implementación

5.1. Diseño

En este apartado se especifican las decisiones de diseño tomadas.

5.1.1. Patrón Arquitectónico

Este proyecto se ha realizado siguiendo una arquitectura en capas [6], separando el sistema en distintas partes con responsabilidades bien definidas. Esta elección responde a la necesidad de desarrollar un front-end independiente que, en un futuro, se conectará a un back-end mediante API REST, siguiendo un enfoque modular y escalable.

La aplicación está dividida en tres capas principales:

- **Capa de Presentación**: Se encarga de la interfaz de usuario y la interacción con el sistema.
- **Capa de Lógica de Negocio**: En un futuro contendrá el back-end encargado del procesamiento de datos y la aplicación de reglas de negocio
- **Capa de Datos**: Gestionará la persistencia de la información en una base de datos.

Este enfoque permite una mayor flexibilidad y mantenibilidad, ya que cada capa puede evolucionar de manera independiente. Además, el uso de API REST para la comunicación entre el front-end y el back-end garantiza una integración eficiente y facilita la posibilidad de expandir el sistema en el futuro. Se puede ver una representación de este patrón en la Figura 5.1.

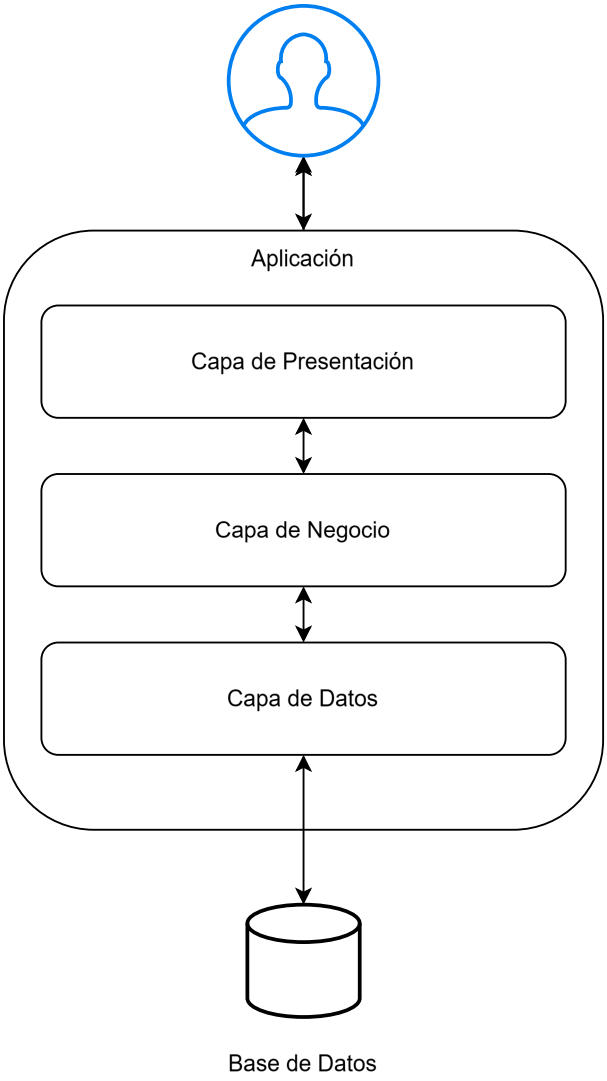

Figura 5.1: Diagrama de Arquitectura de la plataforma.

5.1.2. Estructura modular

Durante el desarrollo del front-end de la plataforma, se ha optado por una arquitectura basada en módulos funcionales, conocida comúnmente como estructura por funcionalidades (feature-based folder structure)[7]. Este patrón consiste en organizar el proyecto agrupando todos los elementos relacionados con una misma vista o funcionalidad dentro de una misma carpeta, lo cual permite una mayor cohesión interna y una separación clara entre distintas partes del sistema.

Concretamente, en este proyecto cada vista principal se encuentra ubicada en la carpeta modules, y cada una de estas vistas contiene su propio conjunto de subcarpetas internas como components, api o hooks, así como su correspondiente archivo index.tsx, que actúa como punto de entrada del módulo. Esta estructura permite que cada vista gestione de forma autónoma sus propios recursos, evitando dependencias innecesarias con otras secciones del sistema. Además, se ha definido un módulo independiente denominado *shared*, que alberga todos aquellos elementos reutilizables y transversales a la aplicación,

como componentes visuales genéricos, custom hooks reutilizables o estilos compartidos.

El uso de este patrón modular presenta múltiples ventajas, especialmente en proyectos de cierta envergadura. Facilita la escalabilidad, ya que nuevos módulos pueden incorporarse sin afectar la organización existente. Mejora también la mantenibilidad, al permitir localizar y modificar el código relacionado con una funcionalidad de forma rápida e intuitiva. Asimismo, fomenta la reutilización, la separación de responsabilidades y la colaboración en entornos de desarrollo en equipo.

En resumen, la adopción de una estructura modular basada en funcionalidades no solo contribuye a una mejor organización del proyecto, sino que también prepara el sistema para futuras extensiones y facilita la integración con un back-end que pueda desarrollarse posteriormente. Se puede ver un ejemplo simple de esta estructura en la Figura 5.2.

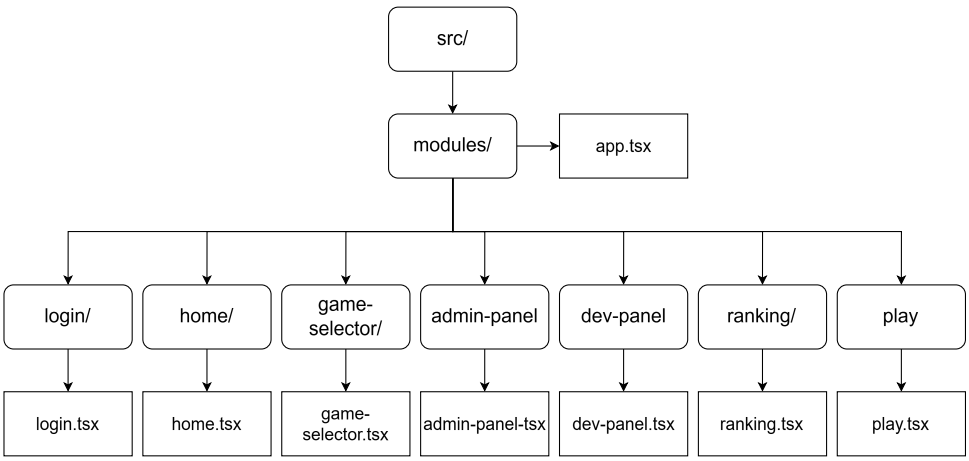

Figura 5.2: Diagrama de módulos de la plataforma.

La estructura de carpetas interna de un módulo tiene la forma representada en la Figura 5.3.

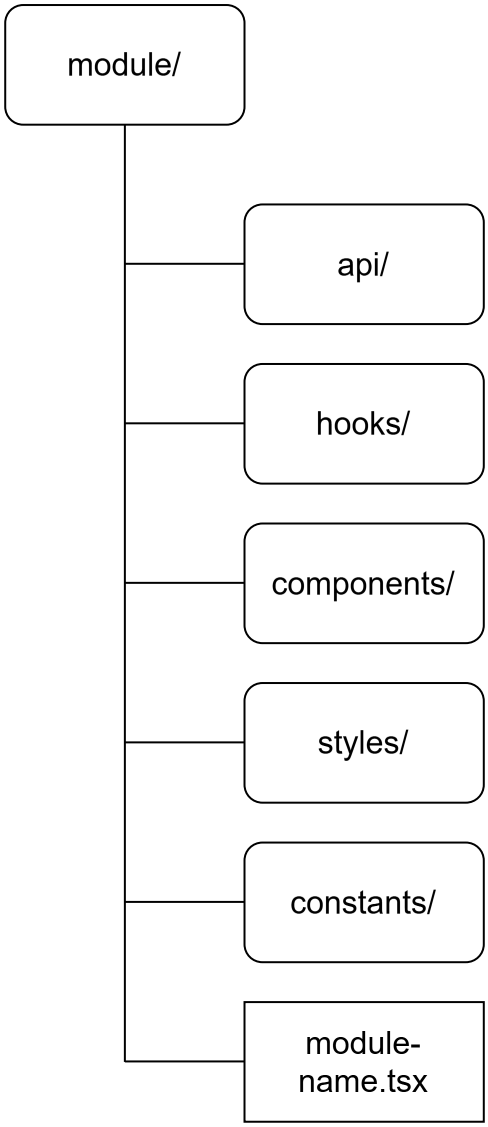

Figura 5.3: Diagrama de Módulo estándar.

5.1.3. Patrón Repositorio

En la arquitectura de esta plataforma web se ha implementado el Patrón Repositorio [8] con el objetivo de desacoplar la lógica de acceso a datos del resto de la aplicación y facilitar el mantenimiento, escalabilidad y reutilización del código.

Cada módulo del front-end, que en esta arquitectura corresponde a una vista específica, cuenta con una carpeta **api/** que implementa de forma estructurada el patrón repositorio. Esta carpeta contiene los siguientes elementos:

Carpeta domain/: Contiene el archivo donde se definen las clases del dominio, es decir, las representaciones lógicas de las entidades que maneja la aplicación. Estas clases son utilizadas en toda la aplicación como modelo de referencia.

Carpeta dto/: Contiene el archivo que define las interfaces que representan los datos tal como son enviados y recibidos desde el back-end (Data Transfer Object - DTO). Estas

interfaces reflejan la estructura cruda de los datos provenientes del servidor.

Carpeta mapper/: Contiene el archivo que implementa las funciones que se encargan de convertir entre DTOs y objetos de dominio, y viceversa. Este mapeo garantiza que la lógica de negocio opere siempre sobre objetos del dominio, aislando así la aplicación de los detalles específicos de la estructura de datos del back-end.

Carpeta interface/: Contiene una interfaz TypeScript que define los métodos que debe implementar cualquier repositorio para ese módulo. Generalmente, estos métodos incluyen operaciones comunes como get, create, update, delete, entre otros.

Carpeta repository/: Implementa la interfaz previamente definida. Cada método del repositorio se encarga de realizar las llamadas a la API correspondientes y, cuando es necesario, transformar los datos usando las funciones del mapper. De esta forma, el repositorio actúa como intermediario entre el back-end y el resto del front-end, gestionando tanto la comunicación como la transformación de los datos.

La Figura 5.4 representa mediante un diagrama la comunicación entre estos elementos.

En ciertas partes del proyecto, se ha necesitado utilizar las mismas clases de dominio y métodos del repositorio en módulos distintos, en este caso se han movido los archivos a la carpeta de **api/** del módulo **shared/**.

Esta organización permite separar claramente las responsabilidades de cada parte del código, reduciendo el acoplamiento entre componentes y facilitando la evolución del software. Además, habilita la posibilidad de realizar tests unitarios sobre la lógica de negocio sin necesidad de depender directamente del back-end.

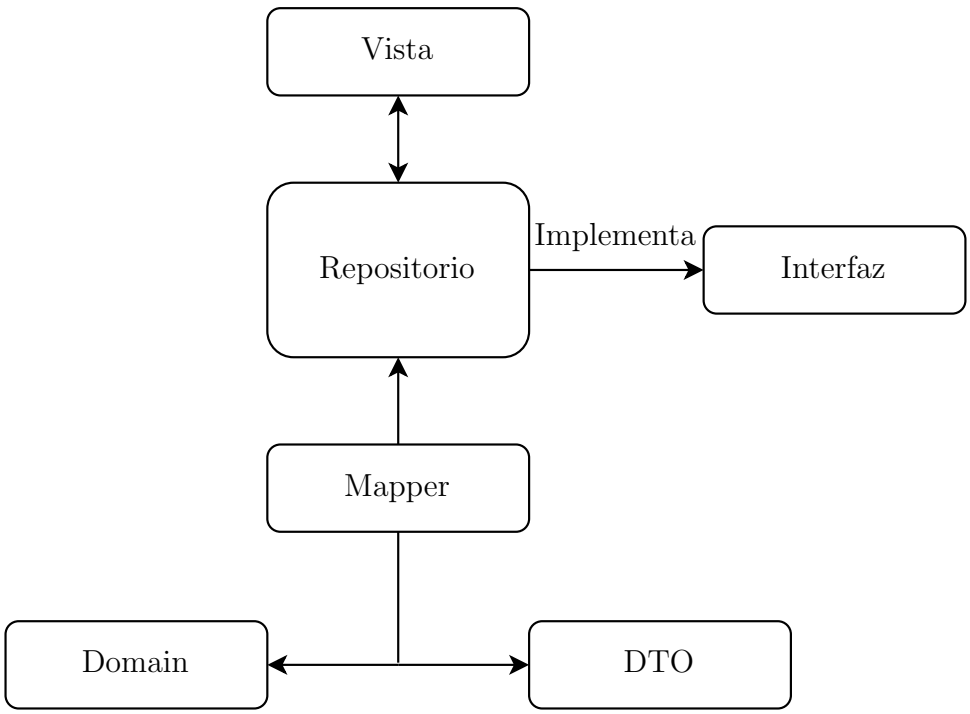

Figura 5.4: Diagrama de Patrón Repositorio.

En las figuras 5.3 a 5.7 se presentan una serie de diagramas para entender el funcio-

namiento de las clases de dominio y de las interfaces DTO dentro de cada una de las carpetas de api.

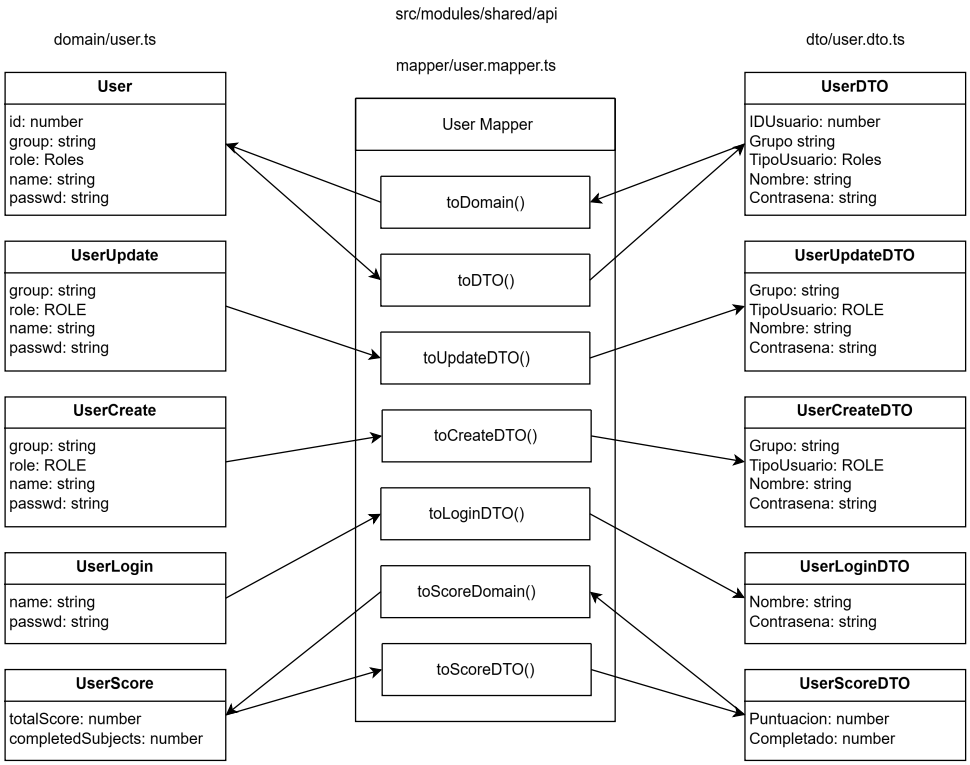

Figura 5.5: Diagrama de Carpeta API de Usuario.

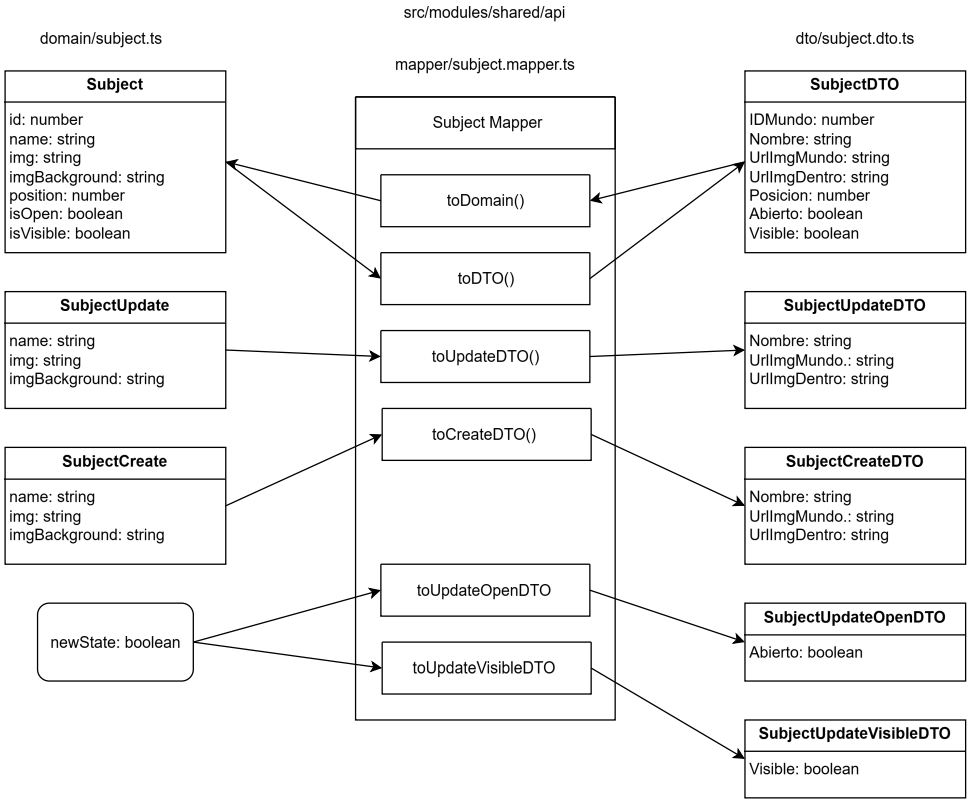

Figura 5.6: Diagrama de Carpeta API de Asignatura.

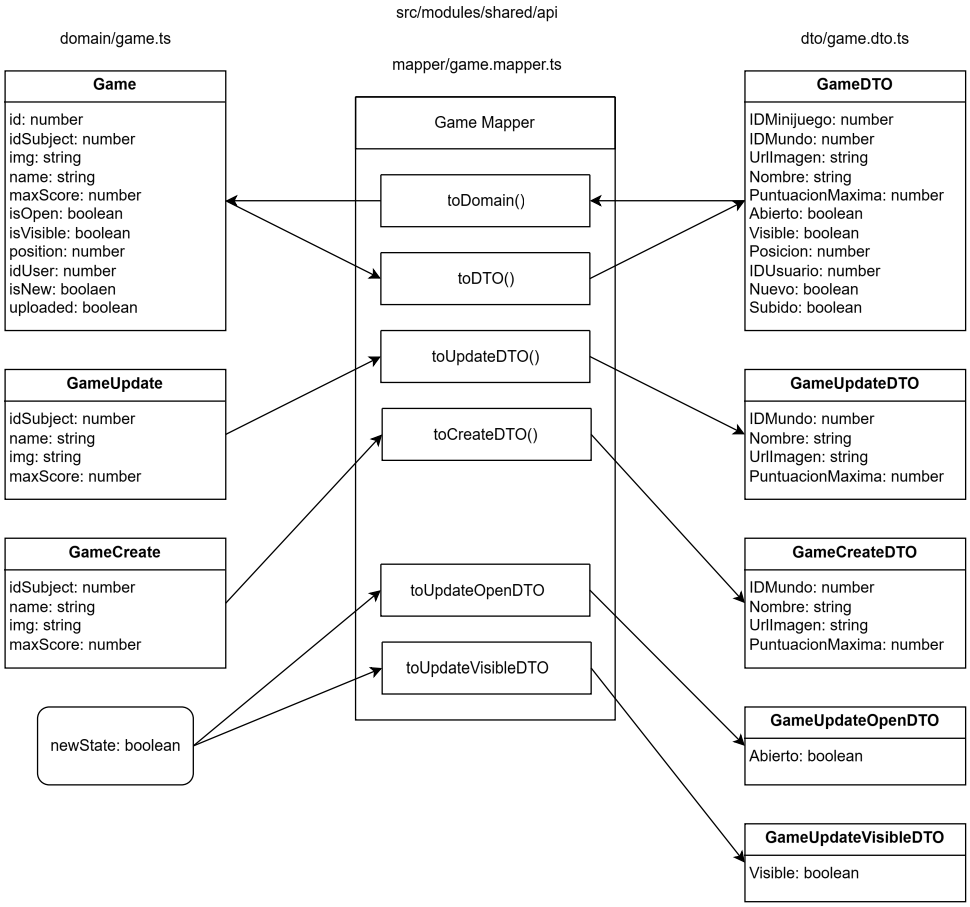

Figura 5.7: Diagrama de Carpeta API de Juego.

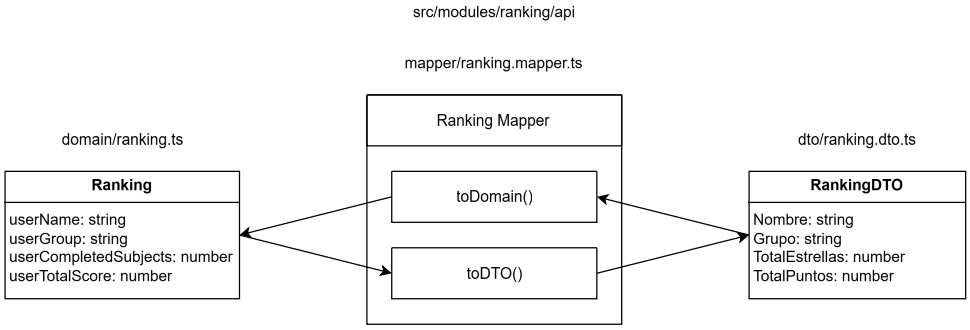

Figura 5.8: Diagrama de Carpeta API de Ranking.

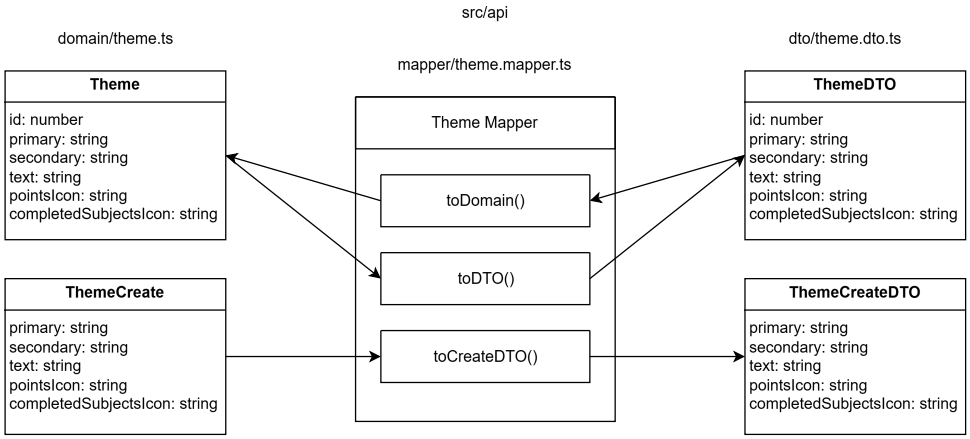

Figura 5.9: Diagrama de Carpeta API de Tema.

5.1.4. La Interfaz

Antes de abordar el desarrollo definitivo de la interfaz de usuario, se realizaron varios bocetos iniciales con el objetivo de explorar diferentes ideas de diseño y distribución de los elementos. Estos prototipos tempranos permitieron validar conceptos de usabilidad, flujo de navegación y organización de la información de forma rápida y visual.

En las figuras 5.10 a 5.20 se presentan los bocetos elaborados durante la fase inicial del proyecto. Cabe destacar que estas representaciones no reflejan necesariamente el aspecto final de la aplicación, sino que sirvieron como punto de partida para el diseño iterativo posterior.

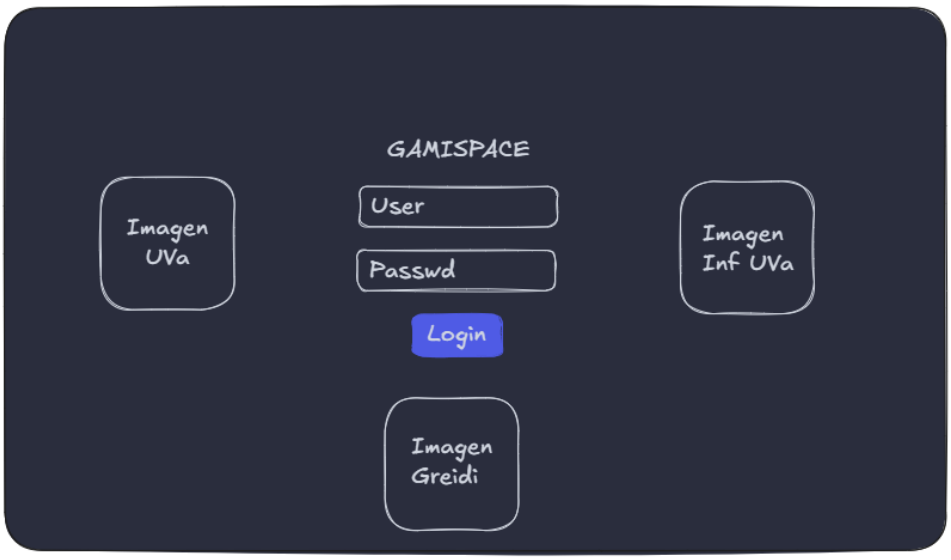

Figura 5.10: Vista Login.

Figura 5.11: Vista Home.

Figura 5.12: Vista GameSelector.

Figura 5.13: Vista Play.

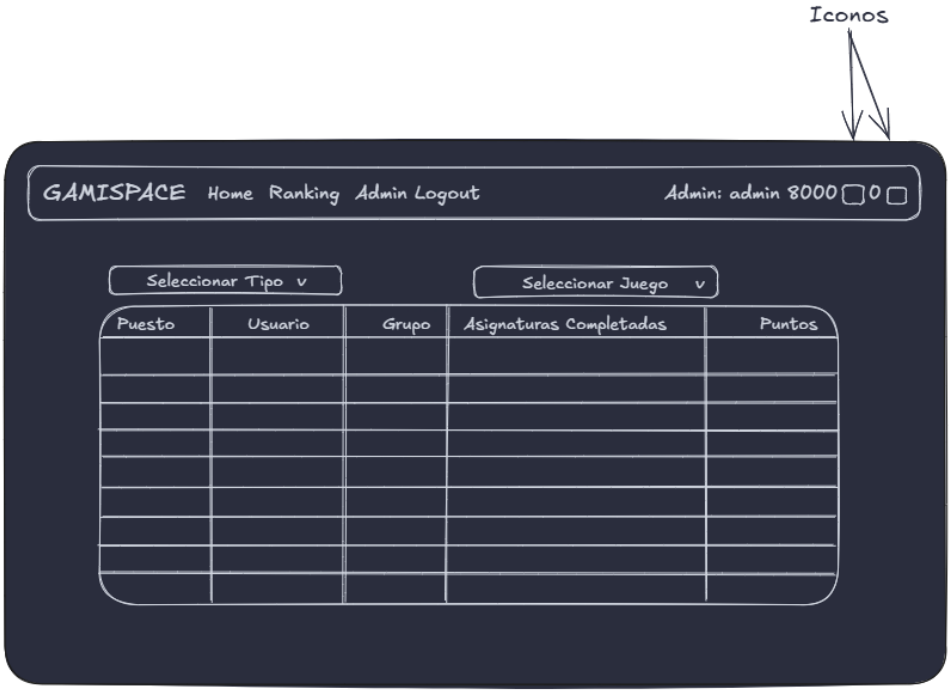

Figura 5.14: Vista Ranking.

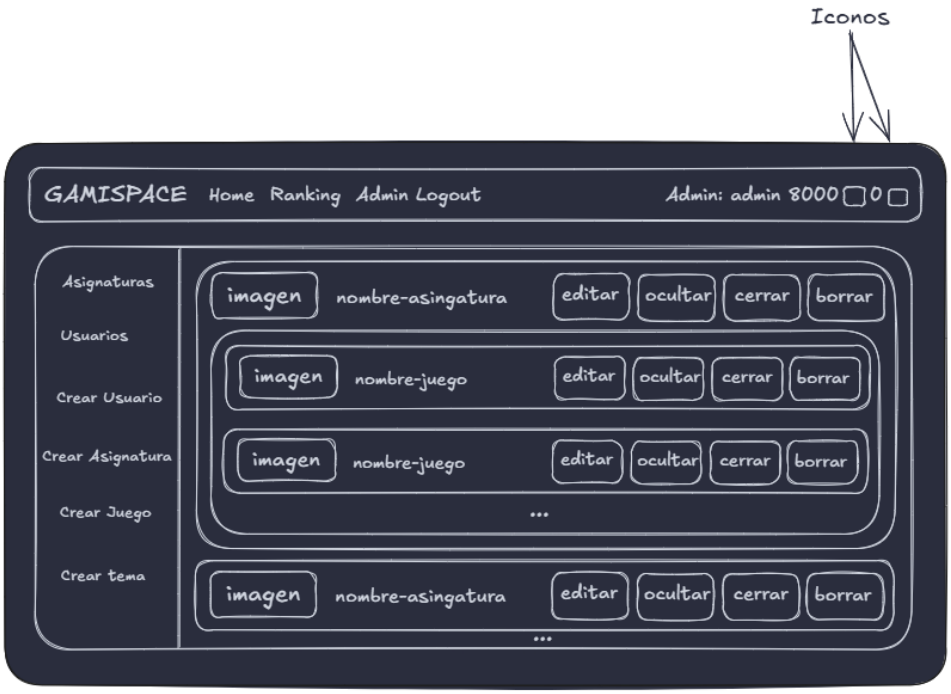

Figura 5.15: Vista AdminPanel - Tab de Asignaturas.

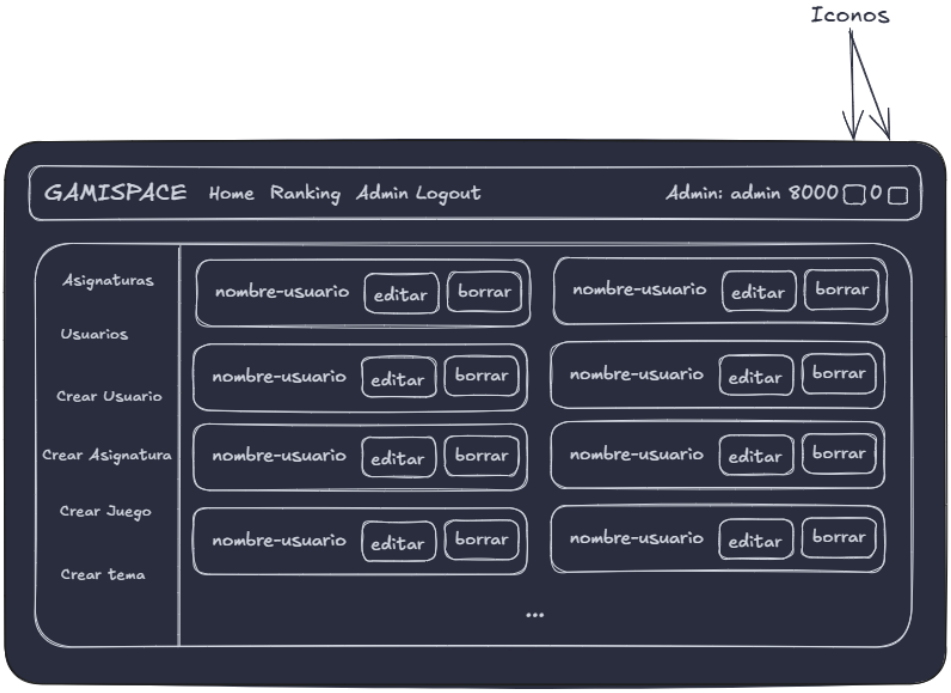

Figura 5.16: Vista AdminPanel - Tab de Usuarios.

Figura 5.17: Vista AdminPanel - Tab de Creación de Usuario.

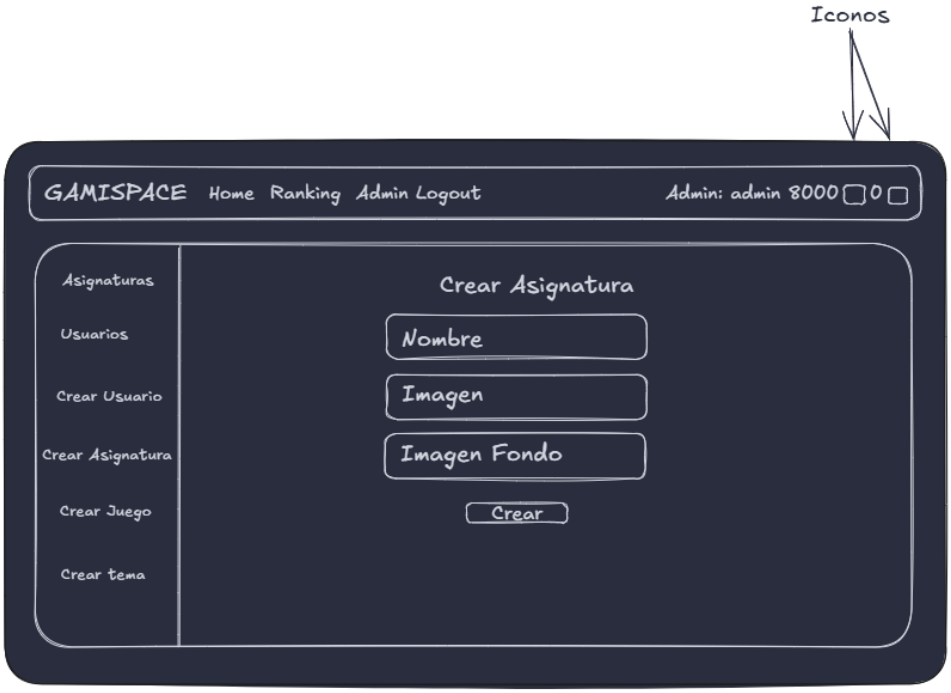

Figura 5.18: Vista AdminPanel - Tab de Creación de Asignatura.

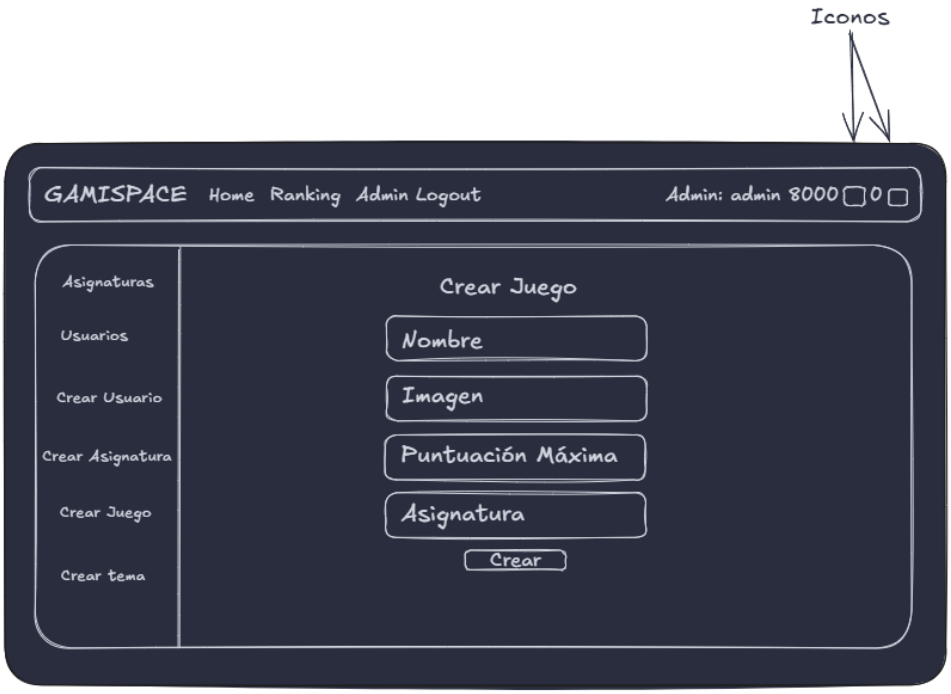

Figura 5.19: Vista AdminPanel - Tab de Creación de Juego.

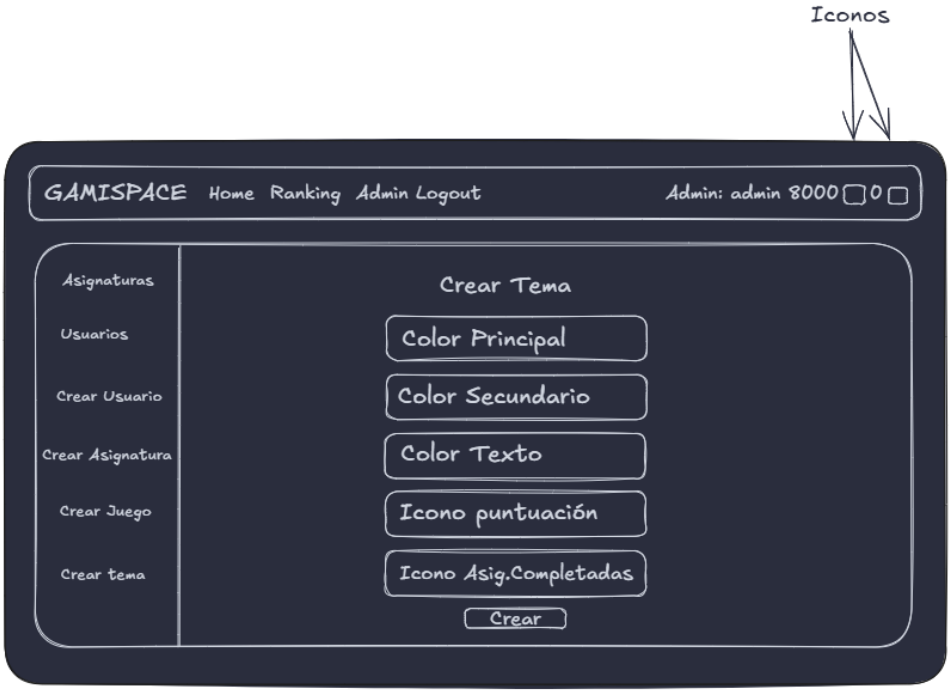

Figura 5.20: Vista AdminPanel - Tab de Creación de Tema.

5.2. Implementación

5.2.1. Tecnologías Usadas

El rediseño de la plataforma se realizará utilizando tecnologías modernas, seleccionadas por su robustez, escalabilidad y amplia adopción en la industria:

- **React**: Se ha elegido como framework principal debido a su enfoque basado en componentes reutilizables, su eficiencia en el manejo del DOM virtual y su amplia comunidad de soporte [1].
- **TypeScript**: Se ha optado por este lenguaje en lugar de JavaScript para mejorar la seguridad del código mediante tipado estático, facilitando la detección de errores en tiempo de desarrollo [2].
- **Bootstrap y CSS**: Se han utilizado para estilizar la interfaz y garantizar un diseño responsivo y accesible. Bootstrap permite agilizar la maquetación sin necesidad de desarrollar componentes desde cero [9] [10].
- **React Router DOM**: Implementado para gestionar la navegación dentro de la plataforma, proporcionando una experiencia fluida entre diferentes secciones [1].
- **Zod**: Utilizado para la validación de datos y esquemas de forma declarativa, asegurando que los datos manipulados en la aplicación cumplan con las estructuras esperadas [11].
- **Framer Motion y React Spring**: Se han empleado para mejorar la experiencia visual con animaciones fluidas y dinámicas [1] [12].
- **Storybook**: Se ha integrado como herramienta para el desarrollo y documentación de componentes de forma aislada, lo que facilita su prueba, revisión visual y reutilización [13].

5.2.2. Estructura del código

A continuación, se describe la organización de los principales directorios del proyecto:

src/: Código fuente principal, contiene el código principal de la aplicación y está dividido en las siguientes carpetas:

- **api/**: Contiene la funciones relacionadas con las llamadas a la API del servicio REST. Aquí se definen los servicios para la comunicación con el back-end.
- **app/**: Incluye los archivos principales de la aplicación:
  - **App.tsx**: Componente raíz de la aplicación.
  - **index.tsx**: Punto de entrada donde se monta el componente principal en el DOM.

    - **Routes.tsx**: Gestión de la navegación y enrutamiento de la aplicación.
    - **ProtectedRoute.tsx**: Componente para manejar rutas protegidas (autenticación/autorización).

- **assets/**: Contiene recursos estáticos como imágenes, íconos y fuentes.
- **constants/**: Almacena valores constantes que se reutilizan en la aplicación, evitando la duplicación de código.
- **context/**: Contiene los contextos globales de la aplicación utilizando React Context API para la gestión de estados globales.
- **hooks/**: Carpeta donde se encuentran los custom hooks, funciones reutilizables que encapsulan lógica específica.
- **pages/**: Carpeta que almacena los módulos o vistas de la aplicación; cada vista está separada en su propia carpeta. Cada una de estas carpetas puede contener una arquitectura similar al resto del proyecto, por ejemplo, la carpeta "home/"puede contener a su vez sus carpetas çomponents/", "hooks/.º "styles/".
    - **adminPanel/**: Panel de administración.
    - **dev-panel/**: Panel de desarrollador.
    - **game-selector/**: Selector de juegos.
    - **home/**: Página principal, selector de asignaturas.
    - **login/**: Página de inicio de sesión.
    - **not-Found/**: Página de error 404.
    - **play/**: Página donde se ejecutan los juegos.
    - **ranking/**: Página con la clasificación de jugadores.
    - **shared/**: Contiene componentes reutilizables que pueden ser usados en varias páginas, como botones, textos o inputs.
- **services/**: Aquí se ubican los servicios de la aplicación.
- **styles/**: Almacena archivos de estilos globales.

Además de los directorios mencionados, el proyecto cuenta con varios archivos de configuración, entre ellos:

- **.eslintrc.json, .eslint.config.js**: Configuración de ESLint para mantener un código limpio y consistente.
- **.prettier.json**: Configuración de Prettier para el formateo automático del código.
- **tsconfig.json**: Configuración de TypeScript.
- **package.json**: Lista de dependencias y scripts del proyecto.
- **README.md**: Documentación general del proyecto.

5.2.3. Uso de StoryBook

Para facilitar la documentación, visualización y reutilización de los componentes de interfaz desarrollados, se ha integrado **Storybook** en el proyecto. Storybook es una herramienta de desarrollo que permite crear y mostrar de forma aislada los componentes de una aplicación front-end, sin necesidad de ejecutarla en su totalidad.

Gracias a esta herramienta, ha sido posible construir una **librería de componentes reutilizables** y mantener una documentación visual interactiva de cada uno de ellos. Esto ha permitido verificar su comportamiento, estilo y estado en distintos contextos (por ejemplo, botones de diferentes tipos, inputs o links), todo ello sin depender de la lógica del resto de la aplicación.

La visualización en tiempo real de los componentes, junto con la posibilidad de simular props, eventos y estados, ha facilitado la detección temprana de errores y la mejora del diseño de la interfaz.

En resumen, Storybook ha contribuido de manera significativa a mantener una arquitectura de front-end ordenada, documentada y orientada a la reutilización de componentes UI, lo que resulta especialmente útil en proyectos de mediano o gran tamaño como este.

5.2.4. La Interfaz Final

En este apartado se muestran las vistas de la interfaz final. En las figuras 5.21 a 5.31 se presentan las capturas de las vistas elaboradas durante la fase final del proyecto.

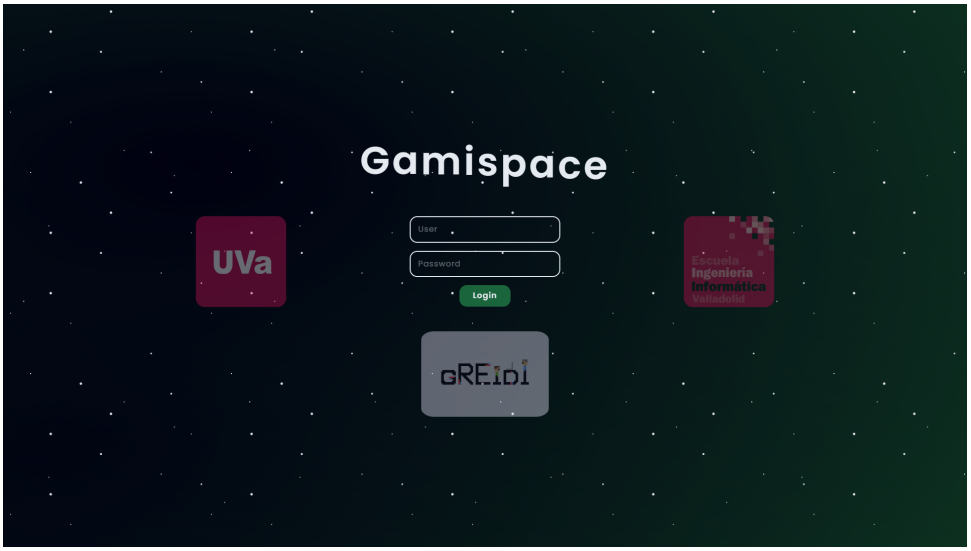

Figura 5.21: Vista Login

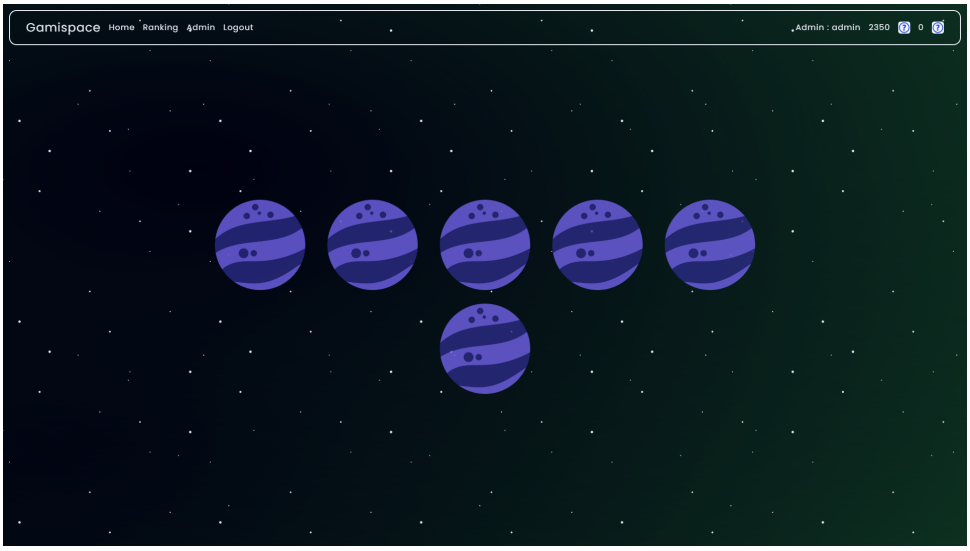

Figura 5.22: Vista Home.

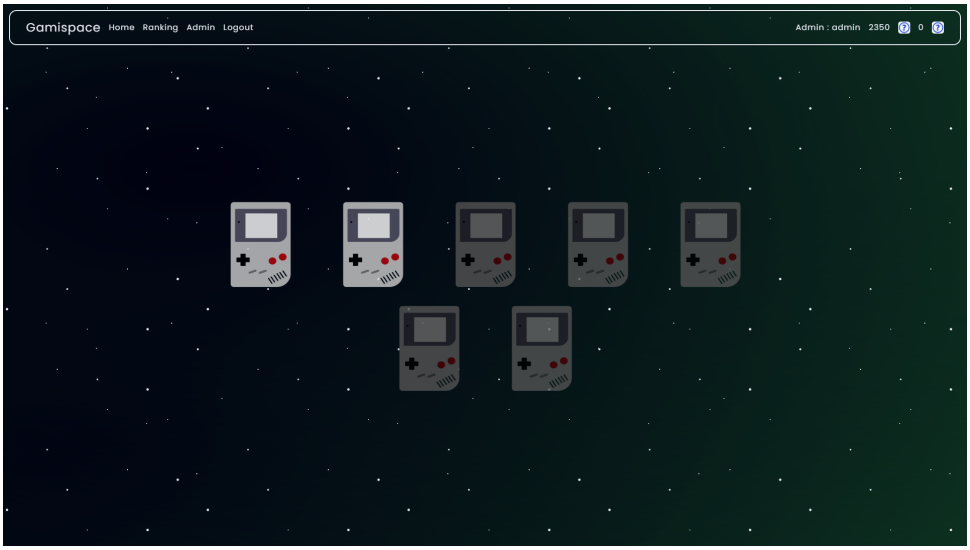

Figura 5.23: Vista GameSelector.

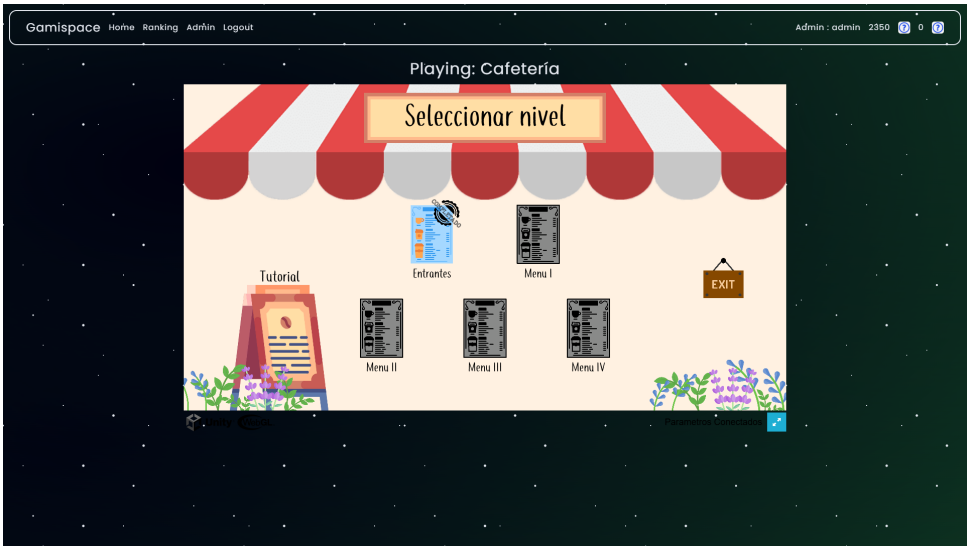

Figura 5.24: Vista Play.

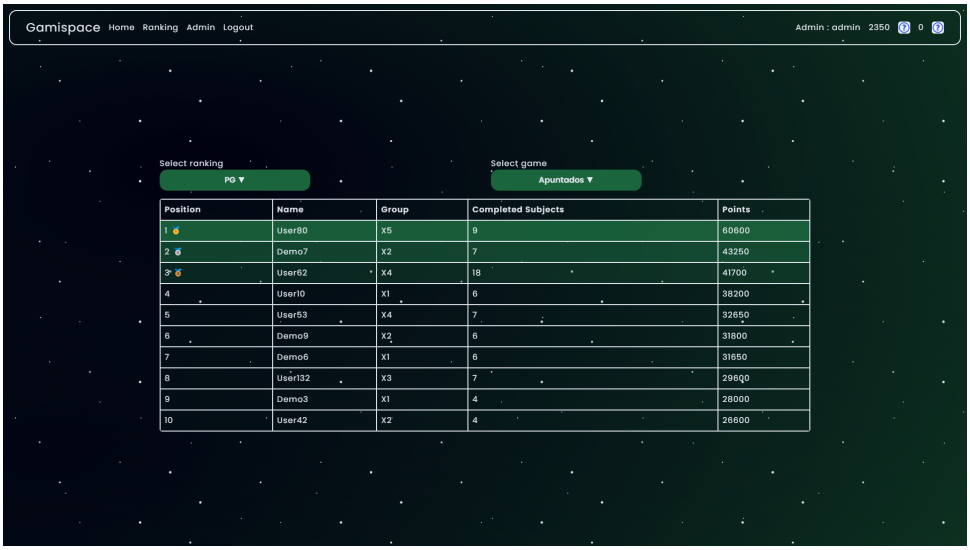

Figura 5.25: Vista Ranking.

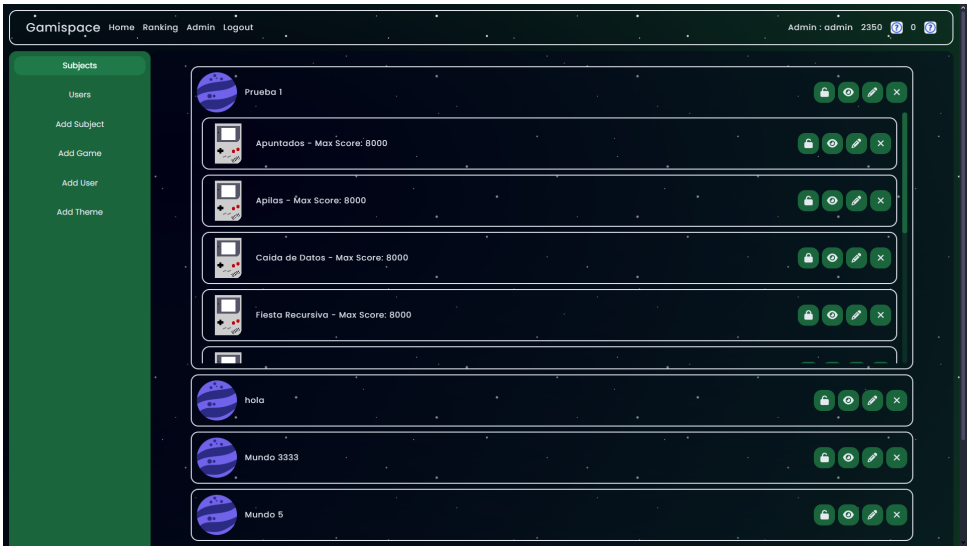

Figura 5.26: Vista AdminPanel - Tab de Asignaturas.

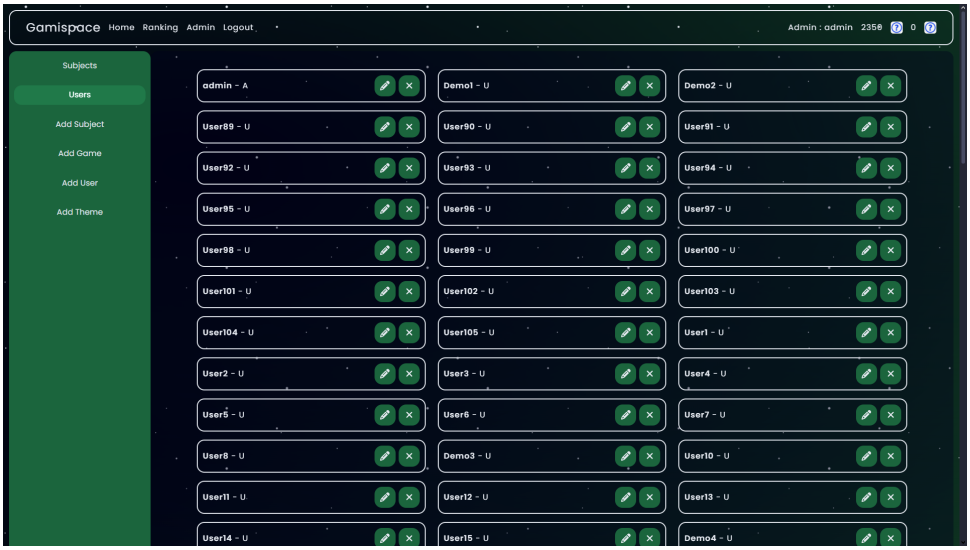

Figura 5.27: Vista AdminPanel - Tab de Usuarios.

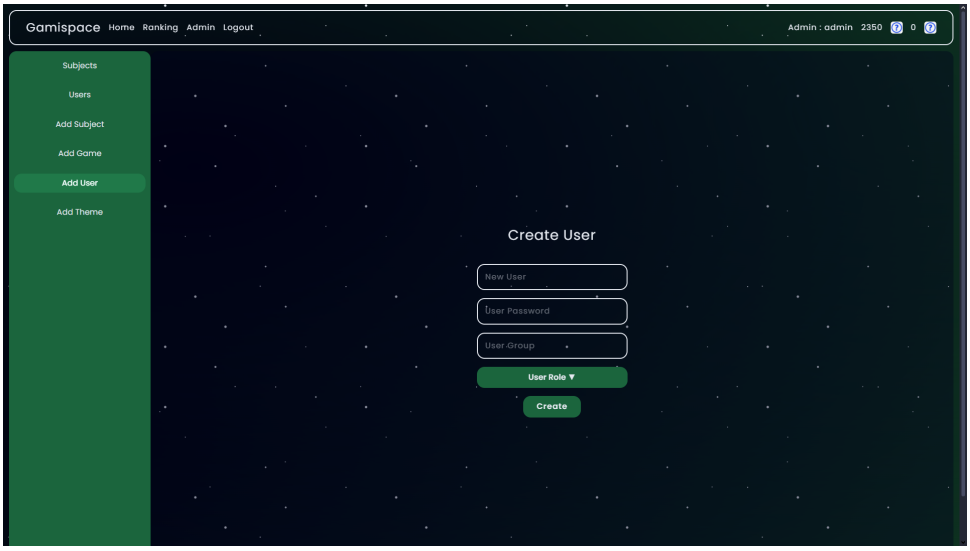

Figura 5.28: Vista AdminPanel - Tab de Creación de Usuario.

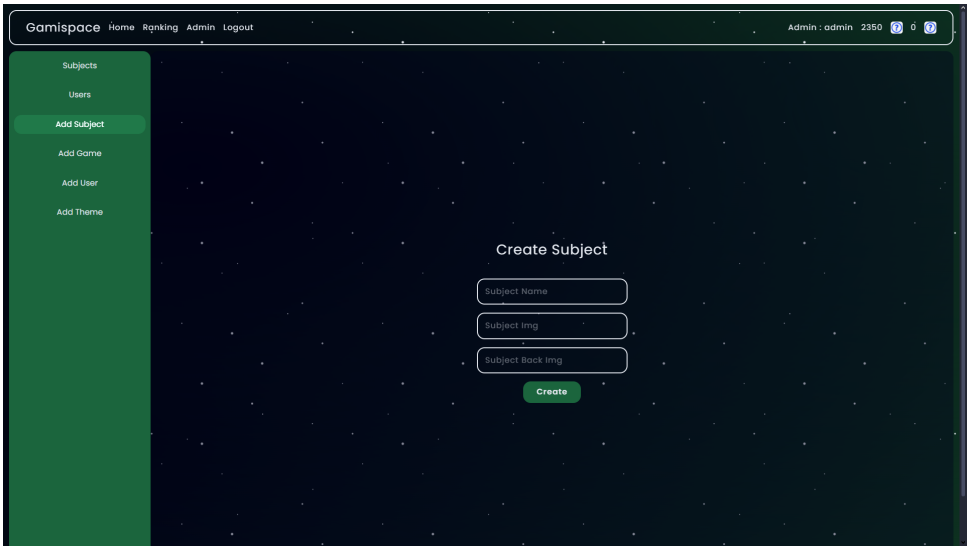

Figura 5.29: Vista AdminPanel - Tab de Creación de Asignatura.

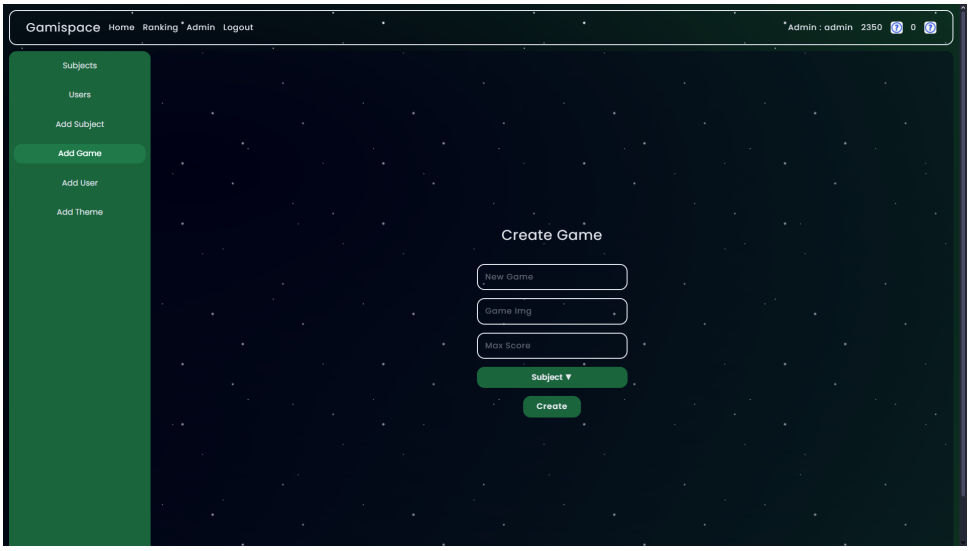

Figura 5.30: Vista AdminPanel - Tab de Creación de Juego.

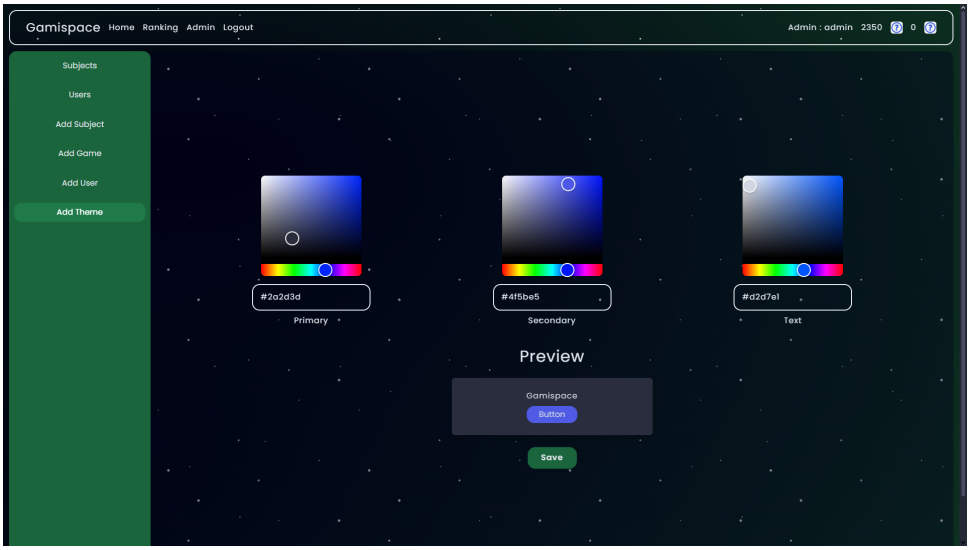

Figura 5.31: Vista AdminPanel - Tab de Creación de Tema.

5.2.5. Pruebas

Durante esta fase del proyecto, las pruebas del sistema se han llevado a cabo utilizando la **base de datos actualmente en uso en la plataforma original**. El objetivo principal de estas pruebas ha sido verificar el funcionamiento técnico de las funcionalidades desarrolladas.

Se ha optado por aplicar pruebas de tipo **caja negra**, evaluando que cada funcionalidad responda de forma correcta ante diferentes entradas, sin considerar la lógica interna. Estas pruebas están definidas en la Tabla 5.1.

Funcionalidad	**Estado**	**Observaciones**
Registro y login de usuario	Parcialmente funcional	Actualmente se permite el login para los usuarios de tipo Administrador y Desarrollador, el resto de tipos de usuario no existen actualmente en la base de datos actual, con la que se han realizado las pruebas; por lo tanto no se han podido probar al pertenecer a la futura implementación del back-end.
Visualización de asignaturas	Funcional	El modelo de datos actual permite la visualización completa.
Acceso a asignatura y visualización de juegos	Funcional	Comprobado que el flujo funciona correctamente con la base actual.
Acceso a juego e interacción con él	Funcional	Se permite acceder a los juegos, aunque al no existir el back-end todos los enlaces apuntan a un mismo juego, usado para las pruebas.
Clasificación individual y grupal	Funcional	Se pueden consultar todos los tipos de ranking existentes sin error.
Gestión de asignaturas	Funcional	Se pueden consultar y modificar las asignaturas y sus juegos.
Gestión de usuarios	Parcialmente Funcional	La consulta y eliminación de usuarios funciona, la modificación no.
Creación de asignatura	Funcional	El formulario de creación de asignatura recoge los datos de una asignatura correctamente.
Creación de juego	Funcional	El formulario de creación de juego recoge los datos de un juego correctamente.
Creación de usuario	Funcional	El formulario de creación de usuario recoge los datos de un usuario correctamente.
Creación de temas	Funcional	Se permite simular la creación de un tema.

Tabla 5.1: Resumen de funcionalidades probadas con base de datos actual

Capítulo 6

Conclusiones

El desarrollo de este Trabajo de Fin de Grado ha permitido diseñar e implementar el front-end de una plataforma educativa centrada en la gamificación del aprendizaje universitario. A lo largo del proyecto se ha abordado la planificación, diseño y desarrollo de una solución moderna y escalable, basada en tecnologías actuales como React y TypeScript, lo que ha supuesto un avance importante tanto en términos de arquitectura como de experiencia de usuario.

Durante el proceso, se han enfrentado diversos desafíos, especialmente relacionados con la falta de experiencia previa en algunas de las tecnologías utilizadas y con la necesidad de reajustar ciertas fases del proyecto. Estas dificultades, sin embargo, han contribuido significativamente al aprendizaje y a la adquisición de competencias técnicas clave para el desarrollo de aplicaciones web modernas.

La plataforma desarrollada sienta las bases para un sistema más completo que podrá, en el futuro, integrarse con un back-end mediante APIs REST, alojar múltiples juegos didácticos y ofrecer una experiencia competitiva y educativa a los estudiantes. Además, se ha intentado seguir una estructura modular y el uso de buenas prácticas de desarrollo para intentar asegurar que el sistema pueda evolucionar fácilmente y adaptarse a nuevas necesidades.

En definitiva, el proyecto ha cumplido con los objetivos planteados inicialmente, aportando una solución funcional, estructurada y con un enfoque hacia la escalabilidad, la usabilidad y el aprendizaje basado en el juego. Creemos que este trabajo no solo representa un avance técnico, sino también una propuesta con potencial real de aplicación en entornos educativos universitarios.

Bibliografía

[1] React Team. *Documentación oficial de React*. Referencia técnica consultada en línea. Meta. 2024.

[2] TypeScript Team. *Documentación oficial de TypeScript*. Guía técnica consultada en línea. Microsoft. 2024.

[3] LeaderTask. *Explicación de la metodología en cascada*. Guía de gestión de proyectos consultada en línea. LeaderTask. 2024.

[4] Instituto Nacional de Ciberseguridad (INCIBE). *Análisis de riesgos en 5 pasos sencillos*. Inf. téc. Guía institucional consultada en línea. INCIBE, 2023.

[5] Talent.com. *Informe salarial de desarrolladores Full Stack en España*. Datos de mercado consultados en línea. 2024.

[6] Mark Richards. *Software Architecture Patterns*. Libro consultado para patrones arquitectónicos. O'Reilly Media, 2015.

[7] Ahsan Bilal. *Arquitectura modular para aplicaciones React de gran escala*. Artículo técnico consultado en línea. 2022.

[8] Plain English. *Aplicación de patrones Repository y Adapter en React*. Artículo consultado en línea. 2024.

[9] Bootstrap Team. *Guía oficial de Bootstrap*. Documentación técnica consultada en línea. Bootstrap. 2024.

[10] Mozilla Developer Network. *Documentación de CSS*. Referencia técnica consultada en línea. Mozilla. 2024.

[11] Zod Maintainers. *Guía de validación de esquemas con Zod*. Documentación técnica consultada en línea. Zod. 2024.

[12] React Spring Contributors. *React Spring - Biblioteca de animaciones*. Referencia técnica consultada en línea. React Spring. 2024.

[13] Storybook Team. *Documentación oficial de Storybook*. Guía de usuario consultada en línea. Storybook. 2024.