Lucas García Díaz

Tomás Robertson Morris

Francisco José Sánchez Praena

Santiago Vilches Romero

ies zaidín-vergeles
granada

Índice

1. Introducción y Finalidades del Proyecto

El presente documento describe el desarrollo de una plataforma web de comercio electrónico orientada a la venta y reserva de videojuegos, denominada *GGDeals*. El proyecto se enmarca en el módulo de Formación en Centros de Trabajo (FCT) del ciclo formativo de Grado Superior en Desarrollo de Aplicaciones Web, y tiene como objetivo aplicar de forma práctica los conocimientos adquiridos a lo largo del curso en un entorno simulado de desarrollo profesional.

1.1 Justificación del Proyecto

La elección de una tienda online de videojuegos como temática del proyecto no es casual. Se trata de un sector en auge, con una amplia base de usuarios y múltiples necesidades funcionales que permiten abarcar distintas áreas del desarrollo web: desde la gestión de usuarios y autenticación, hasta la manipulación de catálogos complejos y operaciones transaccionales. Asimismo, permite incorporar nociones de diseño visual, experiencia de usuario (UX), arquitectura de software, contenedores y pruebas automatizadas, lo que lo convierte en un proyecto integral.

Además, este tipo de plataformas constituye un excelente caso de estudio para aplicar el paradigma cliente-servidor, el desarrollo de APIs RESTful, el uso de frameworks modernos como React y Spring Boot, y la integración de tecnologías ampliamente utilizadas en el mercado laboral como Docker y MySQL.

1.2 Finalidades Generales

Las finalidades del proyecto se pueden desglosar en los siguientes objetivos globales:

1. **Diseñar y desarrollar una aplicación web funcional y atractiva** que permita a los usuarios finales explorar un catálogo de videojuegos, añadir productos al carrito, realizar compras y reservas, y gestionar su historial de transacciones.
2. **Implementar una arquitectura escalable y modular**, separando claramente la lógica del cliente (frontend), del servidor (backend) y de la base de datos.
3. **Fomentar la reutilización de componentes, el uso de buenas prácticas de programación y una codificación mantenible y segura**.
4. **Demostrar competencias técnicas adquiridas durante el ciclo formativo**, como el diseño de bases de datos relacionales, la creación de interfaces responsivas, la programación orientada a objetos, el consumo de servicios web y el trabajo en equipo con control de versiones.

1.3 Ámbito de Aplicación

El ámbito del proyecto abarca tanto el desarrollo del cliente web como del servidor de aplicaciones y la persistencia de datos. En concreto, el sistema implementado permite:

- Autenticación y registro de usuarios con roles diferenciados (usuario y administrador).
- Exploración de videojuegos filtrables por plataforma, edición, características y precio.
- Visualización de detalles de producto, incluyendo imágenes y ediciones disponibles.
- Gestión de un carrito de compra y sistema de reservas.
- Control administrativo de productos, ediciones, claves digitales y disponibilidad.
- Persistencia de datos mediante una base de datos MySQL con modelo relacional normalizado.
- Pruebas unitarias e integración en el backend mediante JUnit y Testcontainers.

El resultado esperado es una plataforma totalmente funcional que, si bien no está destinada a producción comercial, puede ser utilizada como prototipo base para un proyecto real de comercio electrónico.

2. Objetivos del Proyecto

El proyecto GGDeals tiene como propósito general el diseño e implementación de una aplicación web de compraventa y reserva de videojuegos, que proporcione una experiencia de usuario completa, eficiente y moderna. Para alcanzar esta finalidad, se han definido una serie de **objetivos específicos** organizados en tres grandes bloques: funcionales, técnicos y formativos.

2.1 Objetivos Funcionales

Estos objetivos están relacionados con las funcionalidades que debe ofrecer la plataforma desde la perspectiva del usuario final:

1. **Permitir el registro y autenticación de usuarios** mediante un sistema seguro con roles diferenciados (usuarios y administradores).
2. **Mostrar un catálogo de videojuegos completo**, donde los usuarios puedan buscar, explorar y filtrar por diversos criterios como consola/plataforma, género, formato (físico o digital), características, y precio.
3. **Habilitar la compra y reserva de videojuegos**, tanto en formato digital (mediante claves) como físico (con recogida en tienda).
4. **Proporcionar una gestión de carrito de compras**, permitiendo agregar o quitar productos antes de confirmar la operación.
5. **Integrar un sistema de notificaciones** para informar al usuario cuando un juego reservado esté disponible.
6. **Ofrecer un panel administrativo** donde los administradores puedan gestionar el inventario de productos, las claves digitales, las ediciones, los usuarios y las ventas.
7. **Mostrar información multimedia de cada videojuego**, incluyendo carátulas, capturas de pantalla y descripción de contenido.

2.2 Objetivos Técnicos

Estos objetivos están orientados al diseño e implementación de la arquitectura y la tecnología empleada en el desarrollo del proyecto:

1. **Aplicar el principio de separación de responsabilidades (SoC)** mediante la distribución en capas: frontend (cliente), backend (API) y base de datos.
2. **Desarrollar una API RESTful segura y estructurada** utilizando el framework Spring Boot.
3. **Implementar el frontend en React con TypeScript**, utilizando Vite como herramienta de construcción para una mayor eficiencia y velocidad de desarrollo.
4. **Utilizar JWT (JSON Web Tokens)** para la autenticación y gestión de sesiones de los usuarios registrados.
5. **Diseñar un modelo de datos relacional en MySQL**, completamente normalizado y adaptado a las necesidades de la aplicación.
6. **Contenerizar la aplicación mediante Docker**, incluyendo el backend, el frontend y la base de datos, y gestionar la orquestación con Docker Compose.
7. **Desarrollar pruebas unitarias e integración** utilizando JUnit y Testcontainers para asegurar el correcto funcionamiento del backend.
8. **Aplicar buenas prácticas de desarrollo** como el uso de control de versiones con Git, patrones de diseño, principios SOLID y documentación del código.

2.3 Objetivos Formativos

Este proyecto también tiene objetivos relacionados con el aprendizaje y la consolidación de conocimientos adquiridos en el ciclo formativo:

1. **Consolidar el uso de tecnologías avanzadas** que, aunque no todas han sido impartidas en el aula, son ampliamente utilizadas en entornos profesionales (Spring Boot, Docker, JWT, React con TypeScript).
2. **Adquirir experiencia en el desarrollo completo de una aplicación web**, abarcando tanto el cliente como el servidor y la base de datos.
3. **Fomentar el trabajo en equipo y el uso de herramientas colaborativas** como Git y GitHub.
4. **Aprender a resolver problemas técnicos reales**, adaptarse a cambios y mejorar la capacidad de autogestión.
5. **Desarrollar competencias en el diseño visual y la experiencia de usuario**, siguiendo un mockup predefinido como guía para el frontend.

3. Descripción General del Proyecto

El proyecto GGDeals consiste en el desarrollo de una tienda online de videojuegos que permite la compra y reserva de productos en formato físico o digital. Se trata de una aplicación web de tipo cliente-servidor, dividida en tres capas fundamentales: **front-end** (interfaz de usuario), **back-end** (lógica de negocio y API REST) y **base de datos** (persistencia de la información).

El sistema está inspirado en plataformas reales como *G2A*, *InstantGaming* y *GG.deals*, adaptando su funcionalidad a un entorno de desarrollo académico con tecnologías modernas, código propio, y un diseño gráfico personalizado basado en un mockup.

3.1 Usuarios Objetivo

La aplicación está diseñada para dos tipos de usuarios principales:

- **Usuarios registrados (clientes):** pueden explorar el catálogo de videojuegos, filtrar por plataforma, precio, características o edición, añadir productos al carrito, completar compras y gestionar reservas.
- **Administradores:** tienen acceso a un panel de gestión donde pueden añadir, editar o eliminar productos, controlar claves digitales, consultar estadísticas de ventas y gestionar usuarios.

3.2 Funcionalidades Principales

Las principales funcionalidades que ofrece la plataforma son:

Funcionalidad	Descripción
Autenticación	Registro y login de usuarios mediante credenciales, gestionado por JWT.
Catálogo de videojuegos	Listado de productos con filtros por plataforma, precio, edición, etc.
Compra y reserva de juegos	Sistema de compra inmediata y reserva con seguimiento de disponibilidad.
Carrito de compra	Añadir/quitar productos y proceder al pago.
Gestión de claves digitales	Administración de réplicas (códigos de activación) por edición y plataforma.
Panel de administración	Acceso exclusivo para administradores con opciones de CRUD para todos los recursos.
Visualización multimedia	Visualización de portadas y capturas de los juegos desde la ficha de producto.

3.3 Arquitectura General

El sistema está dividido en tres bloques tecnológicos:

- **Frontend:** desarrollado con React y TypeScript, siguiendo el patrón de componentes. Utiliza rutas para navegación, llamadas a la API para consumir datos, y estilos personalizados basados en un diseño mockup.
- **Backend:** desarrollado con Java y Spring Boot. Expone una API RESTful protegida mediante autenticación JWT, encargada de la gestión de usuarios, productos, claves, ventas y reservas. Utiliza JPA para persistencia en base de datos.
- **Base de Datos:** se ha diseñado un modelo relacional en MySQL que contempla las relaciones entre videojuegos, plataformas, ediciones, claves, usuarios, reservas y ventas.

Esta separación permite trabajar de forma desacoplada en cada capa del sistema, facilitando el mantenimiento y la escalabilidad.

3.4 Integración y Despliegue

Se han utilizado contenedores Docker para garantizar un entorno uniforme de ejecución tanto para desarrollo como pruebas. El fichero compose.yaml permite iniciar los servicios necesarios: base de datos MySQL, API backend y frontend React, conectados entre sí.

3.5 Ciclo de Vida del Desarrollo

El proyecto ha sido desarrollado siguiendo una metodología incremental y modular. A medida que se implementaban las funcionalidades clave, se realizaban validaciones mediante pruebas manuales y automáticas (unitarias e integración).

El control de versiones ha sido gestionado a través de Git y GitHub, permitiendo la colaboración entre miembros del equipo y facilitando el seguimiento de avances.

En conjunto, GGDeals constituye un sistema completo de comercio electrónico, cuyo diseño ha sido concebido desde cero para demostrar tanto la viabilidad técnica como la aplicabilidad de los conocimientos adquiridos en el ciclo de Desarrollo de Aplicaciones Web.

4. Tecnologías Utilizadas

El desarrollo del proyecto GGDeals ha implicado el uso coordinado de diversas tecnologías de frontend, backend, bases de datos, contenedores y pruebas. Cada una de ellas ha sido seleccionada con el objetivo de garantizar un sistema eficiente, modular, seguro y acorde con las prácticas del desarrollo web moderno.

Las tecnologías utilizadas se organizan en capas, según su función dentro de la arquitectura del sistema.

4.1 Frontend (Interfaz de Usuario)

Tecnología	Descripción Técnica
React	Librería JavaScript utilizada para construir una SPA (Single Page Application). Permite gestionar múltiples vistas en una sola página mediante el DOM virtual, mejorando el rendimiento y la experiencia del usuario.
TypeScript	Superset de JavaScript que añade tipado estático, mejorando la robustez del código y reduciendo errores en tiempo de desarrollo.
Vite	Herramienta moderna para la construcción de aplicaciones web, más rápida que Webpack y con recarga en caliente. Utilizada como bundler del frontend.
Tailwind CSS	Framework de utilidad primero (“utility-first”) que permite aplicar estilos directamente en las clases de HTML, agilizando la creación de diseños responsivos y personalizados sin escribir CSS a mano.
React Router	Biblioteca de enrutamiento para React. Permite definir rutas del frontend como /, /login, /catalogue, etc.

4.2 Backend (API y Lógica de Negocio)

Tecnología	Descripción Técnica
Java	Lenguaje principal del backend. Orientado a objetos, maduro, con amplio soporte para aplicaciones empresariales.
Spring Boot	Framework de desarrollo web que simplifica la creación de APIs RESTful. Permite la integración de múltiples dependencias bajo una configuración mínima.
Spring Security + JWT	Se ha implementado un sistema de autenticación basado en tokens JWT. Los usuarios reciben un token al iniciar sesión que se utiliza en las solicitudes autenticadas.
Thymeleaf + Bootstrap	Para el panel de administración se ha implementado una interfaz adicional basada en Thymeleaf, motor de plantillas para Java, junto con estilos visuales proporcionados por Bootstrap. Esta sección permite gestionar desde el navegador los contenidos y operaciones del sistema.
Maven	Gestor de dependencias y construcción del proyecto. Usado para compilar, testear y empaquetar la API.
Lombok	Biblioteca que reduce la escritura de código repetitivo en Java, como getters, setters y constructores.
JPA (Hibernate)	Usado como ORM (Object-Relational Mapping) para mapear entidades Java a tablas de la base de datos MySQL.

4.3 Base de Datos

Tecnología	Descripción Técnica
MySQL	Sistema de gestión de bases de datos relacional. Utilizado para persistir información de usuarios, juegos, ventas, claves, reservas, etc. Compatible con Spring Data JPA.
Modelo Normalizado	Se ha diseñado un modelo E-R con relaciones complejas: muchos-a-muchos y uno-a-muchos, reflejado en tablas auxiliares como aux_game_feature, aux_game_platform, etc.

4.4 Contenedores y Orquestación

Tecnología	Descripción Técnica
Docker	Herramienta para contenerizar la aplicación. Se han creado contenedores separados para el backend, el frontend y la base de datos.
Docker Compose	Permite definir y ejecutar múltiples contenedores en conjunto. Usado mediante el archivo compose.yaml para levantar el sistema completo en local con un solo comando.

4.5 Testing y Control de Calidad

Tecnología	Descripción Técnica
JUnit	Framework de pruebas unitarias para Java. Usado para verificar la lógica de negocio en el backend.
Testcontainers	Herramienta para pruebas de integración que permite ejecutar bases de datos reales (como MySQL) en contenedores durante los tests.
Postman (manual)	Herramienta externa usada para probar endpoints de la API durante la fase inicial del desarrollo.

4.6 Control de Versiones y Colaboración

Tecnología	Descripción Técnica
Git	Sistema de control de versiones distribuido. Usado para llevar el seguimiento de cambios en el código fuente.
GitHub	Plataforma web para alojar el repositorio del proyecto, gestionar ramas, issues y colaboraciones entre los miembros del equipo.

4.7 Entorno de Desarrollo

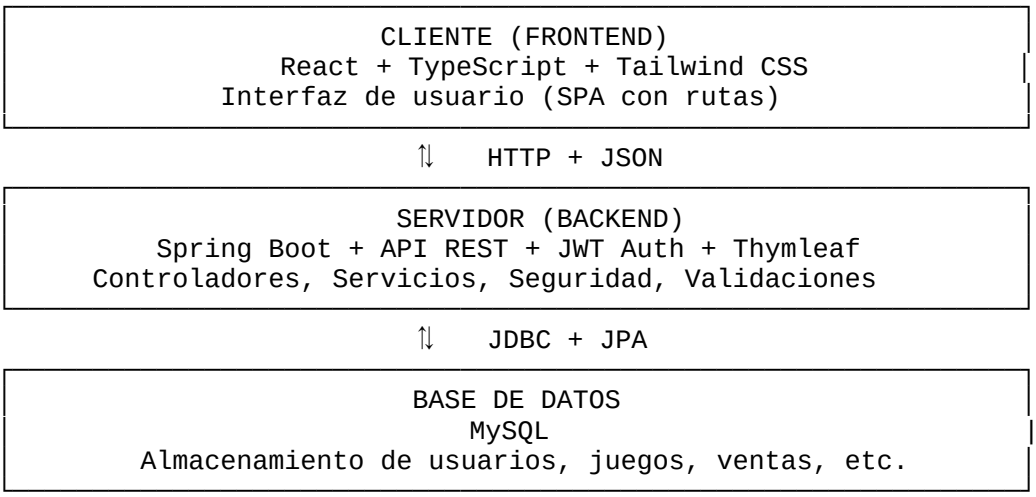
- **IDE Backend:** IntelliJ IDEA con soporte para Spring Boot y Maven.
- **IDE Frontend:** Visual Studio Code con extensiones para React, ESLint y TypeScript.
- **Sistema Operativo:** Windows y Linux (mediante WSL para ejecutar contenedores).
- **Navegadores:** Google Chrome, Firefox (pruebas de interfaz y compatibilidad).

5. Arquitectura de la Aplicación

El sistema GGDeals está diseñado bajo una arquitectura modular de tres capas, compuesta por un cliente web (frontend), un servidor de aplicaciones (backend) y un sistema de gestión de bases de datos (DBMS). Esta división promueve la separación de responsabilidades, facilitando el mantenimiento, la escalabilidad y la incorporación futura de nuevas funcionalidades.

El patrón seguido es el **modelo cliente-servidor** con comunicación mediante **API RESTful**. Todas las capas están conectadas pero desacopladas, lo que permite modificar o sustituir una de ellas sin afectar significativamente al resto.

5.1 Descripción General de la Arquitectura



5.2 Capas del Sistema

5.2.1 Capa de Presentación (Frontend)

Desarrollada con **React y TypeScript**, esta capa gestiona toda la interacción con el usuario final. Utiliza una interfaz de página única (SPA) con rutas dinámicas para navegar entre las distintas secciones: inicio, catálogo, detalles del producto, login, carrito, etc.

Las peticiones a la API se realizan de forma asíncrona (AJAX) usando fetch o bibliotecas como axios. La autenticación se maneja mediante cookies con JWT y control de sesión en el navegador.

5.2.2 Capa de Aplicación (Backend)

Implementada con **Spring Boot (Java)**, esta capa expone un conjunto de endpoints REST. Cada endpoint representa una operación CRUD sobre una entidad (por ejemplo: /api/games, /api/users, /api/sales).

El backend está dividido en:

1. **Controladores:** reciben peticiones HTTP y devuelven respuestas.
2. **Servicios:** encapsulan la lógica de negocio.
3. **Repositorios:** gestionan el acceso a la base de datos mediante JPA.

El sistema de autenticación utiliza **JWT (JSON Web Tokens)**. Al iniciar sesión, el servidor genera un token firmado que se almacena en una cookie segura. Este token se envía automáticamente con cada petición y permite validar al usuario sin mantener sesiones activas en el servidor.

5.2.3 Capa de Persistencia (Base de Datos)

El modelo de datos está implementado en **MySQL**, con una estructura relacional normalizada. Se utilizan claves foráneas, relaciones muchos-a-muchos y uno-a-muchos para representar:

- Juegos ↔ Plataformas
- Juegos ↔ Características
- Juegos ↔ Ediciones
- Ventas ↔ Usuarios ↔ Réplicas (claves digitales)
- Reservas ↔ Juegos ↔ Usuarios

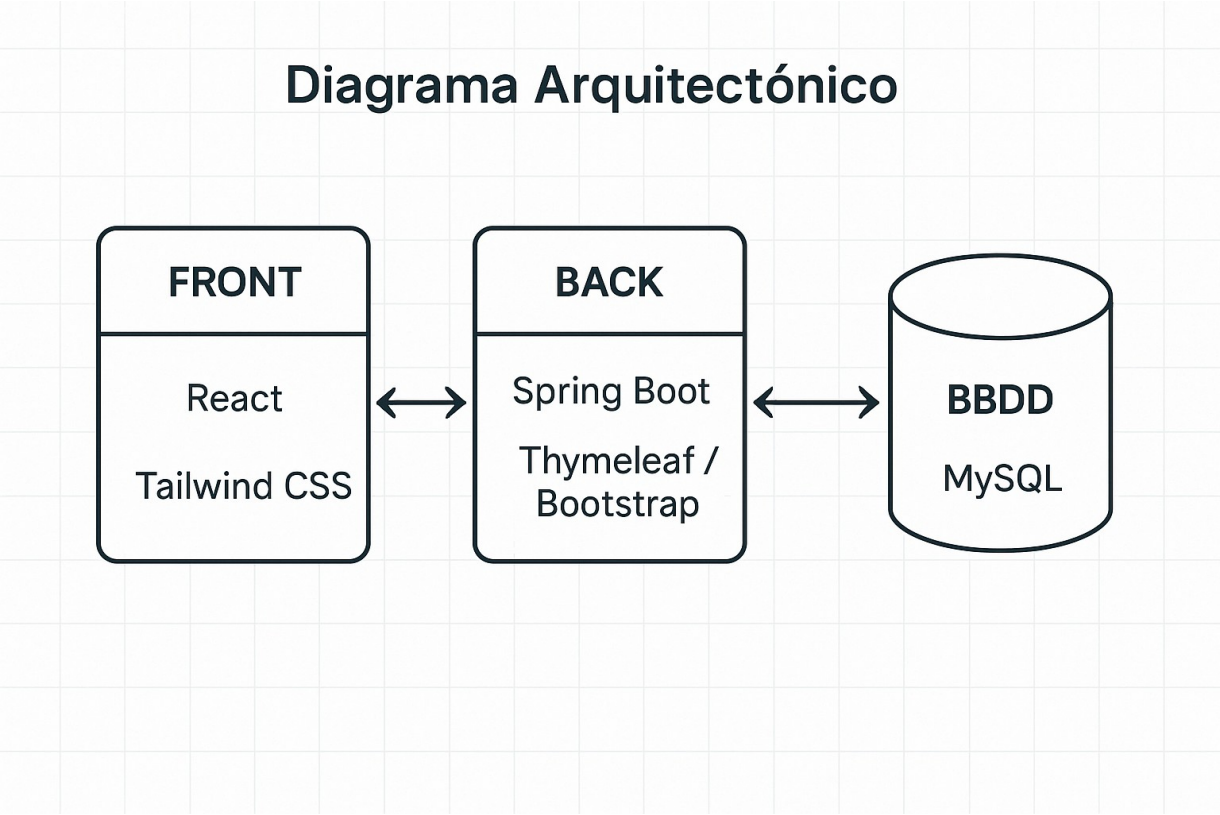
5.3 Comunicación entre Componentes

- El frontend se comunica con el backend a través de **solicitudes HTTP REST**.
- El backend interactúa con la base de datos a través de **Spring Data JPA**, que traduce automáticamente las entidades Java a sentencias SQL.
- El backend responde al frontend en formato **JSON**.
- El sistema completo está orquestado por **Docker Compose**, que lanza simultáneamente todos los servicios necesarios (React, API, MySQL).

5.4 Ventajas de esta Arquitectura

- ✔ **Escalabilidad:** Se pueden añadir más servicios o microservicios en el futuro (por ejemplo: pasarelas de pago, estadísticas, notificaciones).
- ✔ **Despliegue independiente:** Frontend y backend se pueden desplegar por separado.
- ✔ **Seguridad modular:** La capa de autenticación y autorización está encapsulada y puede adaptarse a OAuth2, LDAP, etc.
- ✔ **Mantenibilidad:** El uso de componentes desacoplados permite desarrollar, depurar o actualizar cada capa de forma independiente.

5.5 Diagrama Arquitectónico



6. Estructura del Repositorio

El proyecto GGDeals está alojado en un repositorio Git que sigue una organización por carpetas clara y modular. Esta estructura permite separar las responsabilidades de cada componente del sistema (frontend, backend, base de datos, documentación, contenedores) y facilita el mantenimiento, la colaboración entre desarrolladores y la ejecución del proyecto en distintos entornos.

A continuación se describe la jerarquía principal del repositorio, junto con la función de cada directorio o archivo relevante.

6.1 Vista General

```
css
CopiarEditar
/ (Raíz del proyecto)
│
├── docs/                          → Documentación y recursos visuales
│   ├── Diagrama de Base de Datos.pdf
│   └── GGDeals.png                → Mockup visual de la interfaz
│
├── ggdeal.api/                    → Backend (Spring Boot)
│   ├── src/main/java/             → Código fuente principal (Java)
│   │   └── com/ggdeal/
│   │       ├── configuration/     → Seguridad, JWT, políticas CORS
│   │       ├── controller/        → Endpoints REST
│   │       ├── model/             → Entidades JPA: Game, User, Sale...
│   │       ├── repository/        → Interfaces JPA
│   │       └── service/           → Lógica de negocio
│   ├── src/test/java/             → Pruebas unitarias y de integración
│   ├── resources/                 → application.properties, configuración Spring
│   └── pom.xml                    → Dependencias y build con Maven
│
├── ggdeal-front/                  → Frontend (React + TypeScript)
│   ├── app/
│   │   ├── routes/                → Vistas principales (Home, Login, Catalogue)
│   │   └── welcome/               → Página de bienvenida
│   ├── public/                    → Archivos estáticos (index.html, favicon)
│   ├── Dockerfile                 → Contenedor Docker del frontend
│   ├── vite.config.ts             → Configuración de Vite
│   └── package.json               → Dependencias y scripts
│
├── compose.yaml                   → Orquestación Docker de frontend, backend y DB
├── .gitignore                     → Exclusión de archivos temporales en Git
├── .dockerignore                  → Exclusión de archivos en imagen Docker
└── README.md                      → Documentación principal del repositorio
```

6.2 Estructura del Backend (ggdeal.api/)

La aplicación Java está estructurada en paquetes lógicos que cumplen con el patrón de capas:

- `controller`: define los endpoints REST (por ejemplo, `/api/games`, `/api/users`).
- `service`: contiene la lógica de negocio (validaciones, operaciones entre entidades).
- `repository`: interfaces para el acceso a la base de datos mediante Spring Data JPA.
- `model`: clases que representan las entidades persistentes.
- `configuration`: componentes de seguridad, autenticación JWT, CORS y configuración general de Spring Security.

6.3 Estructura del Frontend (ggdeal-front/)

El proyecto de React sigue una estructura por vistas y componentes, con enfoque modular:

- `routes/`: cada archivo representa una pantalla completa (catálogo, login, etc.).
- `welcome/`: componentes e imágenes de la página de introducción.
- `public/`: archivos visibles públicamente como index.html y logotipos.
- `Dockerfile`: define cómo construir la imagen Docker del frontend.
- `vite.config.ts`: especifica la configuración del entorno de desarrollo y producción.

6.4 Documentación (docs/)

Incluye archivos complementarios:

- Diagrama E-R de la base de datos.
- Mockup de la interfaz web.
- Recursos gráficos usados en el proyecto.

Este material es útil para validación del diseño, explicación a terceros o evaluadores y base para futuras mejoras.

6.5 Archivos de Orquestación y Configuración

- `compose.yaml`: permite levantar todos los servicios (React, API, MySQL) en contenedores Docker conectados entre sí. Facilita pruebas y despliegue local sin configurar entornos manualmente.
- `.gitignore` y `.dockerignore`: excluyen archivos temporales, credenciales locales o directorios de dependencias como `node_modules/`, `target/`, `logs/`, etc.
- `README.md`: contiene la guía de instalación y uso del sistema. Es el punto de partida para nuevos colaboradores.

6.6 Control de Versiones

El repositorio está gestionado con **Git** y alojado en **GitHub**, lo que permite:

- Trabajo colaborativo en ramas separadas.
- Seguimiento de cambios y regresión de errores.
- Documentación automatizada con commits.
- Posible integración continua futura (CI/CD).

7. Backend – API REST con Spring Boot

El backend de la plataforma GGDeals está desarrollado con el framework **Spring Boot**, una herramienta moderna y ampliamente adoptada para la creación de aplicaciones web en Java. Se ha diseñado una **API RESTful** estructurada y segura, encargada de gestionar todos los datos y operaciones necesarias para el funcionamiento del sistema, incluyendo usuarios, videojuegos, claves digitales, ventas y reservas.

7.1 Estructura General del Backend

La aplicación está organizada de forma modular en paquetes que siguen una arquitectura en capas:

```
css
CopiarEditar
ggdeal-api/
├── controller/      → Define los endpoints REST
├── service/         → Lógica de negocio
├── model/           → Entidades JPA (mapeo con base de datos)
├── repository/      → Interfaces JPA para acceso a datos
├── configuration/   → Configuración de seguridad y JWT
└── resources/       → Configuración general y properties
```

7.2 Controladores (Controllers)

Los controladores son clases anotadas con `@RestController`. Cada uno está mapeado a una URL base y contiene métodos que responden a las operaciones HTTP típicas: `GET`, `POST`, `PUT`, `DELETE`.

Ejemplos:

1. `UserController`: gestión de autenticación (login, registro).
2. `GameController`: listado y consulta de videojuegos.
3. `AdminController`: acciones restringidas a administradores.

Los métodos devuelven respuestas JSON y estados HTTP apropiados (`200 OK`, `201 Created`, `400 Bad Request`, etc.).

7.3 Entidades (Model)

Cada entidad del sistema representa una tabla en la base de datos y está anotada con `@Entity`. Las relaciones se definen explícitamente:

- **OneToMany**: un usuario puede tener muchas reservas.
- **ManyToMany**: un juego puede tener múltiples plataformas y ediciones.
- **OneToOne**: una venta se asocia con una clave (réplica) única.

Ejemplos de entidades:

`User`, `Game`, `Edition`, `Feature`, `Replica`, `Reservation`, `Sale`, `Plataform`, `GameMedia`.

7.4 Persistencia con JPA

Se utiliza **Spring Data JPA** para la abstracción del acceso a datos. Las interfaces `Repository` permiten realizar operaciones sin escribir SQL:

```
java
CopiarEditar
public interface UserRepository extends JpaRepository<User, Long> {
    User findByEmail(String email);
}
```

Estas interfaces permiten búsquedas automáticas mediante convenciones de nombres (`findByUsername`, `existsById`, etc.).

7.5 Seguridad y Autenticación con JWT

El sistema implementa un **mecanismo de autenticación basado en tokens JWT**:

1. Al iniciar sesión, el usuario recibe un token firmado que contiene su `id`, `email` y `username`.
2. Este token se almacena en una cookie segura y se envía automáticamente con cada petición.
3. En el backend, el token se valida con `JWTVerifier` antes de permitir el acceso a endpoints protegidos.

La configuración de seguridad se define en:

- `SecurityConf.java` → Permite/desactiva autenticaciones básicas y CSRF.
- `JwtProvider.java` → Generación y validación de tokens.

7.6 Validaciones y Seguridad

- Las contraseñas se almacenan de forma segura con **BCrypt**.
- Los endpoints de login y registro son públicos; el resto requieren autenticación.
- Las operaciones de administración están protegidas por el rol `ADMIN`.

7.7 Gestión de Errores

Se devuelve un mensaje claro y el código HTTP correspondiente ante cada fallo. Por ejemplo:

- ✗ `401 Unauthorized` si el token es inválido o no se proporciona.
- ✗ `404 Not Found` si un recurso no existe.
- ✗ `400 Bad Request` para datos mal estructurados.

7.8 Pruebas del Backend

Se ha implementado un entorno de pruebas con:

- **JUnit** para pruebas unitarias de servicios.
- **Testcontainers** para pruebas de integración con bases de datos reales en contenedores.

7.9 Ventajas del uso de Spring Boot

- ✔ Reduce la configuración mediante convenciones predeterminadas.
- ✔ Se integra fácilmente con MySQL, JWT, JPA y otras bibliotecas.
- ✔ Ofrece robustez, modularidad y gran comunidad de soporte.
- ✔ Escalable y adecuado para microservicios.

8. Frontend – Interfaz con React y TypeScript

La capa de presentación del sistema GGDeals ha sido desarrollada utilizando **React** junto con **TypeScript**. Esta combinación permite construir una interfaz moderna, rápida y mantenible, con componentes reutilizables y tipado estático para mejorar la robustez del desarrollo.

La aplicación funciona como una **SPA (Single Page Application)**, lo que significa que el contenido se actualiza dinámicamente sin necesidad de recargar la página completa, proporcionando una experiencia de usuario fluida y moderna.

8.1 Estructura del Proyecto Frontend

El código fuente del frontend se encuentra en la carpeta `ggdeal-front/` del repositorio y está organizado de la siguiente manera:

```
less
CopiarEditar
ggdeal-front/
├── app/
│   ├── routes/          → Rutas principales de navegación (ej: Home, Login,
Catalogue)
│   └── welcome/         → Página de bienvenida con presentación del proyecto
├── public/              → Archivos estáticos (favicon, index.html)
├── vite.config.ts       → Configuración de Vite para desarrollo y build
├── package.json         → Gestión de dependencias y scripts
└── Dockerfile           → Contenedor Docker para despliegue del frontend
```

8.2 Tecnología Utilizada

1. **React**: permite crear una interfaz declarativa basada en componentes.
2. **TypeScript**: añade tipado estático y mejora la calidad del código.
3. **Vite**: herramienta de build moderna, rápida y ligera para proyectos con React.
4. **React Router**: sistema de rutas para navegación entre vistas.
5. **Fetch/Axios**: consumo de la API REST desde el frontend.
6. **CSS personalizado**: siguiendo el mockup de diseño incluido (`GGDeals.png`).

8.3 Principales Componentes

El frontend está dividido en **componentes reutilizables**, cada uno de ellos responsable de una parte de la interfaz. Algunos de los más relevantes son:

Componente	Función Principal
`Header`	Barra superior con navegación, enlaces y estado del usuario
`GameCard`	Muestra la información de un juego en formato de tarjeta
`GameDetail`	Página con detalles ampliados del juego, portadas y ediciones
`Login/Register`	Formularios de autenticación con validaciones
`Cart`	Carrito de compra dinámico con sumatorio de precios
`AdminPanel`	Sección restringida con funciones CRUD para administradores

8.4 Enrutamiento

La navegación está controlada mediante **React Router**, lo que permite renderizar diferentes vistas según la URL sin recargar la página.

Ejemplos de rutas:

Ruta	Descripción
`/`	Página de inicio (bienvenida)
`/login`	Formulario de acceso
`/register`	Registro de nuevos usuarios
`/catalogue`	Lista completa de videojuegos
`/game/:id`	Vista detallada del videojuego
`/cart`	Carrito de compra del usuario
`/admin`	Panel de administración (solo si `admin`)

8.5 Interacción con la API

El frontend se comunica con el backend a través de **peticiones HTTP** a los endpoints definidos en la API REST. Las llamadas se realizan mediante `fetch` o `axios`, y se gestionan los estados de carga, éxito y error.

La autenticación se realiza automáticamente mediante cookies que almacenan el token JWT generado por el backend. Si el token es válido, el usuario puede acceder a rutas protegidas.

8.6 Estilo Visual

El diseño del frontend ha sido creado siguiendo un **mockup visual personalizado** con referencias a la estética de plataformas como GG.deals. Se han aplicado estilos en CSS sin el uso de frameworks de diseño como Bootstrap o Tailwind, permitiendo mayor control y personalización.

El diseño se centra en:

- **Contraste visual** entre texto e imágenes.
- **Jerarquía visual clara** mediante tipografías y espaciado.
- **Diseño responsive** para distintos tamaños de pantalla (en proceso de mejora).

8.7 Consideraciones de Accesibilidad

Aunque no se ha hecho una implementación completa de criterios WCAG, se ha procurado:

- Usar etiquetas semánticas (`<main>`, `<nav>`, `<section>`).
- Incluir textos alternativos en imágenes y botones.
- Evitar colores con bajo contraste.

8.8 Limitaciones y Mejoras Futuras

- ✗ Implementar **mejoras en la validación de formularios**.
- ✗ Agregar **test de componentes** con bibliotecas como Jest o React Testing Library.
- ✗ Optimizar el rendimiento en dispositivos móviles.
- ✗ Incluir animaciones suaves para transiciones y modales.

9. Base de Datos – Modelo Relacional

El modelo de datos de GGDeals ha sido diseñado bajo un enfoque relacional, utilizando **MySQL** como sistema gestor de base de datos. Este modelo refleja las entidades fundamentales del sistema, sus atributos y las relaciones entre ellas, manteniendo una estructura **normalizada** para asegurar integridad, consistencia y eficiencia en las consultas.

La base de datos está preparada para soportar múltiples operaciones simultáneas, incluyendo lectura masiva (para catálogo), escritura (ventas, reservas), y validaciones de integridad referencial.

9.1 Objetivos del Modelo de Datos

1. Representar de forma clara las entidades del negocio (usuarios, videojuegos, ventas, etc.).
2. Permitir una gestión eficiente del inventario y las claves digitales.
3. Facilitar la implementación de funcionalidades como filtrado, búsqueda, y reservas.
4. Asegurar integridad referencial entre entidades mediante claves foráneas.
5. Escalar fácilmente en caso de ampliación del catálogo o usuarios.

9.2 Entidades Principales

A continuación se describen las tablas más relevantes:

1. `users`

- `id` (PK)
- `username`
- `email`
- `password_hash`
- `role` (user / admin)
- `created_at`

Almacena la información de autenticación y permisos de cada usuario.

2. `games`

- `id` (PK)
- `title`
- `description`
- `release_date`
- `developer`
- `publisher`

Contiene los datos generales de los videojuegos. No distingue entre ediciones o plataformas, lo cual se modela en entidades auxiliares.

3. `editions`

- `id` (PK)
- `game_id` (FK)
- `name` (Standard, Deluxe, etc.)
- `price`
- `stock`

Cada juego puede tener múltiples ediciones. Se relacionan directamente con claves y ventas.

4. `platforms`

- `id` (PK)
- `name` (PC, PS5, Xbox Series, etc.)

Listado de plataformas disponibles.

5. `aux_game_platform`

Tabla intermedia **(many-to-many)** entre `games` y `platforms`.

6. `features`

- `id` (PK)
- `name` (Multiplayer, Coop, Online, etc.)

Características que se pueden asociar a los juegos.

7. `aux_game_feature`

Tabla intermedia **(many-to-many)** entre `games` y `features`.

8. `replicas`

- `id` (PK)
- `edition_id` (FK)
- `platform_id` (FK)
- `code` (clave de activación digital)
- `available` (booleano)

Cada réplica representa una clave digital única asociada a una edición de juego y plataforma.

9. `sales`

- `id` (PK)
- `user_id` (FK)
- `replica_id` (FK)
- `sale_date`

Guarda el registro de cada venta confirmada. Una clave no puede venderse dos veces.

10. `reservations`

- `id` (PK)
- `user_id` (FK)
- `edition_id` (FK)
- `platform_id` (FK)
- `status` (pendiente / completada)
- `reservation_date`

Gestión de reservas de juegos sin stock. Se activa cuando haya claves disponibles.

11. `game_media`

- `id` (PK)
- `game_id` (FK)
- `type` (cover / screenshot)
- `url` (ruta al recurso multimedia)

Permite asociar múltiples imágenes a cada juego para su visualización en frontend.

9.3 Relación entre Tablas

El modelo utiliza relaciones:

1) **Uno a Muchos**:
    a) Un usuario puede tener muchas ventas y reservas.
    b) Un juego puede tener muchas ediciones.
2) **Muchos a Muchos** (con tablas auxiliares):
    a) Un juego puede estar en muchas plataformas.
    b) Un juego puede tener muchas características.
3) **Uno a Uno**:
    a) Cada venta se asocia a una única réplica digital.

Estas relaciones permiten consultar, filtrar y gestionar los datos de manera eficiente.

9.4 Normalización del Modelo

El modelo se encuentra **en tercera forma normal (3FN)**:

- No hay redundancia innecesaria.
- Las claves foráneas aseguran la integridad referencial.
- Las relaciones muchos-a-muchos están correctamente gestionadas con tablas intermedias.
- Se evitan atributos multivaluados.

9.5 Integridad y Seguridad

- Todas las relaciones están protegidas con **claves foráneas con restricciones ON DELETE y ON UPDATE**.
- Se aplica control de stock a nivel de edición y réplica.
- Las claves digitales solo pueden ser vendidas o reservadas una vez.
- Las contraseñas no se almacenan en texto plano.

10. Seguridad y Autenticación

La seguridad en aplicaciones web es un componente esencial, especialmente cuando se gestionan datos sensibles como credenciales de usuarios y transacciones comerciales. En GGDeals se ha implementado un sistema robusto basado en **Spring Security** y **JSON Web Tokens (JWT)**, que garantiza la autenticación segura, la autorización basada en roles y la protección frente a accesos no autorizados.

10.1 Objetivos del Módulo de Seguridad

1. **Proteger los endpoints sensibles** del backend.
2. **Evitar accesos no autorizados** a recursos de usuarios o funciones administrativas.
3. **Autenticar usuarios de forma segura** mediante tokens firmados digitalmente.
4. **Asignar permisos por rol** (usuario común o administrador).
5. **Prevenir vulnerabilidades básicas**, como exposición de contraseñas o sesiones inseguras.

10.2 Sistema de Autenticación con JWT

¿Qué es JWT?

JWT (JSON Web Token) es un estándar abierto (RFC 7519) que permite transmitir información entre partes de forma segura como un objeto JSON firmado. Es ideal para sistemas sin estado (**stateless**) como APIs REST.

Flujo de autenticación:

1. El usuario introduce su correo y contraseña.
2. El backend verifica las credenciales.
3. Si son válidas, se genera un **token JWT** que incluye:
    - ID del usuario.
    - Email.
    - Rol.
    - Tiempo de expiración.
4. El token se devuelve al navegador y se almacena en una **cookie HTTP-only**.
5. Cada petición posterior incluye automáticamente el token.
6. El backend valida el token en cada endpoint protegido.

10.3 Componentes del Módulo de Seguridad

Archivo / Clase	Función
`SecurityConf.java`	Configura Spring Security: rutas públicas, protegidas, CORS, CSRF.
`JwtProvider.java`	Genera y verifica los tokens JWT usando una clave secreta.
`JwtFilter.java`	Filtro que intercepta cada solicitud HTTP y valida el token.
`AuthenticationController`	Endpoints públicos de login (`/login`) y registro (`/register`).
`UserDetailsServiceImpl`	Carga los datos del usuario desde la base de datos al iniciar sesión.

10.4 Roles y Permisos

1. El sistema reconoce al menos dos tipos de usuarios:
   - **Usuario estándar (`ROLE_USER`)**: puede acceder al catálogo, gestionar su cuenta, realizar compras y reservas.
   - **Administrador (`ROLE_ADMIN`)**: acceso completo al panel de gestión (CRUD de juegos, claves, usuarios, etc.).
2. La protección se aplica a nivel de método o endpoint mediante anotaciones como:

```
java
CopiarEditar
@PreAuthorize("hasRole('ADMIN')")
```

3. Y en el archivo de configuración de seguridad:

```
java
CopiarEditar
http.authorizeRequests()
    .antMatchers("/api/admin/**").hasRole("ADMIN")
    .antMatchers("/api/user/**").hasAnyRole("USER", "ADMIN")
    .anyRequest().permitAll();
```

10.5 Hashing de Contraseñas

Las contraseñas nunca se almacenan en texto plano. Se utiliza el algoritmo **BCrypt** para cifrarlas antes de guardar en la base de datos.

1. Esto se realiza automáticamente al registrar un usuario:

```
java
CopiarEditar
String encodedPassword = passwordEncoder.encode(rawPassword);
```

2. Y se compara durante el login:

```
java
CopiarEditar
passwordEncoder.matches(rawInput, storedHash);
```

10.6 Cookies Seguras y Protección CSRF

- El token JWT se entrega en una cookie **`HttpOnly`** que no puede ser accedida desde JavaScript.
- Se habilita la protección CORS solo para los dominios autorizados.
- CSRF está deshabilitado por diseño ya que JWT no depende de sesiones y el token es obligatorio para cada petición.

10.7 Rutas Públicas y Privadas

Ruta	Acceso	Protección
`/login`, **`/register`**	Público	No requieren autenticación
`/api/games`, **`/catalogue`**	Público	Lectura abierta
`/api/user/profile`	Privado	Requiere rol USER o ADMIN
`/api/admin/**`	Privado	Exclusivo para rol ADMIN
`/api/sales`, **`/api/cart`**	Privado	Solo usuario autenticado

10.8 Buenas Prácticas de Seguridad Aplicadas

- ✔ Uso de HTTPS (en entorno de producción).
- ✔ Tokens con fecha de expiración y firma única.
- ✔ Filtrado de datos de entrada en formularios.
- ✔ Separación de lógica de seguridad del resto del código.
- ✔ Acceso mínimo necesario para cada tipo de usuario (principio de menor privilegio).

11. Casos de Uso Principales

Los casos de uso representan las acciones más relevantes que los distintos tipos de usuarios pueden realizar dentro del sistema GGDeals. Cada uno de ellos está diseñado para cumplir una necesidad concreta y se implementa mediante la colaboración entre la interfaz de usuario (frontend), la lógica de negocio (backend) y la base de datos.

En GGDeals se identifican dos roles principales de usuario: **Usuario Registrado** y **Administrador**. A continuación se detallan los casos de uso más significativos, clasificados por tipo de usuario.

11.1 Casos de Uso para Usuarios Registrados

CU-01: Registro de Usuario

- **Descripción**: Permite a un nuevo usuario crear una cuenta en la plataforma.
- **Entrada**: Nombre de usuario, email, contraseña.
- **Validaciones**:
    - El correo debe ser único.
    - La contraseña debe tener una longitud mínima.
- **Resultado esperado**: El usuario queda registrado y puede iniciar sesión.
- **Seguridad**: Contraseña almacenada con hash (BCrypt).

CU-02: Inicio de Sesión

- **Descripción**: Autenticación mediante email y contraseña.
- **Proceso**:
    1. Envío de credenciales al backend.
    2. Verificación de credenciales.
    3. Respuesta con token JWT.
- **Resultado esperado**: El usuario accede a funcionalidades privadas (carrito, perfil, etc.).

CU-03: Navegación por el Catálogo de Juegos

- **Descripción**: Explorar el catálogo completo de videojuegos con filtros avanzados.
- **Filtros disponibles**:
    - Plataforma (PS5, Xbox, PC, etc.).
    - Precio.
    - Edición.
    - Características (online, multijugador, etc.).
- **Resultado esperado**: Se muestran juegos que coinciden con los criterios seleccionados.

CU-04: Visualización Detallada de un Juego

- **Descripción**: Ver la información completa de un videojuego.
- **Incluye**:
    - Descripción.
    - Imágenes (carátula, capturas).
    - Ediciones disponibles.
    - Stock por plataforma.
- **Resultado esperado**: Página detallada del producto.

CU-05: Añadir Productos al Carrito

- **Descripción**: Permite seleccionar uno o varios juegos y agregarlos al carrito.
- **Restricciones**:
    - Solo productos disponibles (stock > 0).
- **Resultado esperado**: Carrito actualizado con listado y total.

CU-06: Compra de Videojuego

- **Descripción**: Finalizar la compra de los productos seleccionados.
- **Proceso**:
    1. Selección de edición y plataforma.
    2. Confirmación de compra.
    3. Asignación de réplica digital (clave).
- **Resultado esperado**: Generación de venta, descuento de stock, clave visible para el usuario.

CU-07: Realizar Reserva

- **Descripción**: Permite reservar un juego sin stock.
- **Proceso**:
    - Selección de producto.
    - Registro de la reserva (estado: pendiente).
- **Resultado esperado**: El usuario será notificado cuando haya stock disponible.

11.2 Casos de Uso para Administradores

CU-08: Acceso al Panel de Administración

- **Requiere**: Rol ADMIN y autenticación válida.
- **Acciones disponibles**:
    - Gestión de juegos, ediciones y claves.
    - Visualización de ventas y reservas.
    - Alta/baja de usuarios.

CU-09: Alta de Videojuegos

- **Descripción**: Añadir nuevos juegos al catálogo.
- **Datos requeridos**:
    - Título, descripción, desarrollador, editor, plataformas, características.
- **Resultado esperado**: Juego visible en el catálogo para todos los usuarios.

CU-10: Gestión de Ediciones y Claves

- **Descripción**: Crear ediciones de juego (Standard, Deluxe…) y asociar claves digitales.
- **Resultado esperado**:
    - Las ediciones aparecen en la ficha del juego.
    - Las claves se asignan automáticamente al completar una compra.

CU-11: Gestión de Reservas

- **Descripción**: Confirmar una reserva cuando haya stock disponible.
- **Proceso**:
    1. Revisión de reservas pendientes.
    2. Liberación de una clave asociada.
    3. Notificación al usuario.
- **Resultado esperado**: Reserva marcada como completada.

11.3 Casos de Uso Generales

CU-12: Cierre de Sesión

- **Descripción**: Elimina la cookie del token JWT y revoca acceso a funciones privadas.
- **Resultado esperado**: Redirección a la página de bienvenida.

CU-13: Búsqueda por Nombre

- **Descripción**: Permite buscar un videojuego escribiendo parte del título.
- **Resultado esperado**: Muestra coincidencias en tiempo real.

12. Diseño UI/UX de la Plataforma

El diseño de la interfaz de usuario (UI) y la experiencia de usuario (UX) son componentes esenciales del proyecto GGDeals. Una buena experiencia de usuario no solo hace la aplicación más atractiva, sino que también mejora la comprensión, la accesibilidad y la eficiencia en la navegación por parte del usuario final.

La plataforma ha sido diseñada siguiendo principios de diseño centrado en el usuario, con un estilo visual inspirado en plataformas reales del sector como *GG.deals* o *InstantGaming*, pero adaptado a un enfoque propio mediante un mockup elaborado específicamente para este proyecto.

12.1 Objetivos del Diseño UI/UX

1. Crear una **interfaz clara, limpia y jerárquica** que guíe la atención del usuario.
2. Utilizar **colores contrastados y tipografías legibles** para mejorar la accesibilidad.
3. Asegurar una **navegación intuitiva** con rutas bien definidas y coherencia visual.
4. Diseñar una estructura adaptable a **distintos tamaños de pantalla** (responsive).
5. Priorizar la usabilidad en funcionalidades clave como búsqueda, carrito y detalles de producto.

12.2 Estructura de la Interfaz

El diseño está basado en una disposición **clásica de tres bloques**:

Sección	Descripción
`Header`	Incluye el logotipo, menú de navegación, icono de carrito y login/registro.
Cuerpo principal	Área central donde se renderizan las distintas vistas según la ruta.
`Footer`	Información complementaria, enlaces legales o contacto (en desarrollo).

12.3 Principales Vistas del Sistema

Página de Inicio (Welcome)

- Presenta la plataforma al usuario.
- Contiene mensaje de bienvenida, imagen destacada y llamada a la acción (CTA).
- Enlace al catálogo y botones de registro/inicio de sesión.

Catálogo de Videojuegos

- Lista de tarjetas (`GameCard`) con portada, título y precio.
- Filtros a la izquierda por:
    - Plataforma.
    - Rango de precio.
    - Características.
- Barra de búsqueda por nombre de juego.

Detalle de Producto

- Imagen principal (portada del juego).
- Capturas de pantalla en carrusel.
- Descripción completa del videojuego.
- Selección de edición y plataforma.
- Botón de compra o reserva según disponibilidad.

Carrito de Compra

- Lista de productos añadidos.
- Visualización de totales.
- Botón de pago (simulado).
- Mensajes dinámicos de éxito/error.

Panel de Usuario / Perfil

- Información básica del usuario.
- Historial de compras y claves obtenidas.
- Estado de reservas pendientes o completadas.

Panel de Administración

- Menú lateral con acceso a secciones de gestión.
- Formulario para añadir videojuegos y claves.
- Tabla editable para administrar usuarios, ventas y stock.

12.4 Mockup Visual

El diseño de referencia ha sido creado en formato `.png` y está incluido en la carpeta `docs/` como `GGDeals.png`. Este mockup ha servido como guía visual durante el desarrollo del frontend, marcando:

- Distribución de elementos.
- Estilo de botones.
- Paleta de colores predominante (fondo claro, acentos en azul y naranja).
- Composición de tarjetas de productos.

12.5 Principios de Diseño Aplicados

- **Consistencia visual**: todos los elementos comparten estilos comunes (bordes, sombras, fuentes).
- **Retroalimentación inmediata**: cambios visuales tras acciones (hover, clic, mensajes de estado).
- **Minimización del esfuerzo cognitivo**: iconos y etiquetas autoexplicativas.
- **Diseño adaptativo**: visualización aceptable en pantallas medianas (ordenadores portátiles y tabletas).

12.6 Limitaciones Actuales

- ✗ Aún no se ha optimizado totalmente para móviles pequeños (pendiente diseño mobile-first).
- ✗ Falta implementación de accesibilidad avanzada (lectores de pantalla, navegación por teclado).
- ✗ No se han aplicado animaciones más allá de las básicas proporcionadas por CSS.

12.7 Futuras Mejoras

- ✗ Incorporar modo oscuro (Dark Mode).
- ✗ Integrar librerías de diseño accesible como Material UI o Chakra UI.
- ✗ Validar con usuarios reales mediante tests de usabilidad.
- ✗ Diseñar vistas específicas para dispositivos móviles con breakpoints adicionales.

13. Contenedores y Orquestación con Docker

El uso de contenedores es una práctica ampliamente adoptada en el desarrollo y despliegue de aplicaciones modernas. Permite garantizar que el sistema se ejecuta de forma idéntica en distintos entornos (desarrollo, pruebas, producción), independientemente del sistema operativo o configuración local.

En el proyecto GGDeals se ha utilizado **Docker** para contenerizar tanto el **frontend**, como el **backend** y la **base de datos**. A su vez, se ha implementado **Docker Compose** para orquestar los servicios, facilitando su ejecución conjunta con una única instrucción.

13.1 Objetivos de la Contenerización

1. Asegurar **portabilidad** del sistema completo.
2. Evitar problemas de compatibilidad entre entornos.
3. Simplificar el **despliegue local** o en servidores.
4. Permitir **reinicios y recreación del entorno** de forma controlada.
5. Facilitar la integración de pruebas automáticas.

13.2 Servicios Contenidos

Servicio	Tecnología	Imagen Base	Exposición	Función Principal
Frontend	React + Vite	`node:20-alpine`	`5173`	Interfaz web accesible vía navegador.
Backend	Spring Boot (Java)	`openjdk:21-jdk-slim`	`8080`	API RESTful y lógica de negocio.
Base de Datos	MySQL 8	`mysql:8`	`3306`	Persistencia de datos relacional.

13.3 Dockerfile del Frontend

Ubicado en `ggdeal-front/Dockerfile`. Este archivo construye el proyecto de React y lo sirve con un servidor estático.

```
Dockerfile
CopiarEditar
FROM node:20-alpine
WORKDIR /app
COPY . .
RUN npm install && npm run build
RUN npm install -g serve
CMD ["serve", "-s", "dist"]
EXPOSE 5173
```

13.4 Dockerfile del Backend

Ubicado en `ggdeal.api/Dockerfile`. Empaqueta la aplicación Spring Boot como un `.jar` ejecutable.

```
Dockerfile
CopiarEditar
FROM openjdk:21-jdk-slim
COPY target/ggdeal-api.jar app.jar
ENTRYPOINT ["java", "-jar", "/app.jar"]
EXPOSE 8080
```

13.5 Archivo docker-compose.yaml

Archivo clave para lanzar todos los servicios de forma orquestada:

```
yaml
CopiarEditar
version: '3.8'

services:
  database:
    image: mysql:8
    container_name: ggdeal-db
    restart: always
    environment:
      MYSQL_ROOT_PASSWORD: root
      MYSQL_DATABASE: ggdeal
    ports:
      - "3306:3306"
    volumes:
      - db_data:/var/lib/mysql

  backend:
    build: ./ggdeal.api
    container_name: ggdeal-api
    ports:
      - "8080:8080"
    depends_on:
      - database
    environment:
      SPRING_DATASOURCE_URL: jdbc:mysql://database:3306/ggdeal
      SPRING_DATASOURCE_USERNAME: root
      SPRING_DATASOURCE_PASSWORD: root

  frontend:
    build: ./ggdeal-front
    container_name: ggdeal-front
    ports:
      - "5173:5173"
    depends_on:
      - backend

volumes:
  db_data:
```

13.6 Ejecución del Proyecto con Docker Compose

Con todos los archivos correctamente configurados, el sistema puede levantarse ejecutando desde la raíz del repositorio:

```
bash
CopiarEditar
docker-compose up --build
```

Esto:

- Crea las imágenes necesarias.
- Inicia los contenedores interconectados.
- Permite acceder a:
    - **Frontend**: http://localhost:5173
    - **Backend**: http://localhost:8080/api
    - **MySQL**: accesible por herramientas como DBeaver o PhpMyAdmin.

13.7 Ventajas Obtenidas

- ✔ **Despliegue reproducible** en cualquier equipo con Docker.
- ✔ Aislamiento de dependencias.
- ✔ Independencia del sistema operativo.
- ✔ Posibilidad de integración con pipelines de CI/CD.

13.8 Limitaciones y Futuras Mejoras

- ✗ Actualmente no se ha integrado un sistema de logs compartido.
- ✗ No se ha configurado un volumen para respaldos automáticos de la base de datos.
- ✗ Podría incorporarse **Nginx** como proxy inverso para servir frontend y backend desde el mismo dominio.
- ✗ En producción sería recomendable usar **Docker Swarm** o **Kubernetes**.

14. Pruebas y Testing del Sistema

El proceso de pruebas es una parte esencial del desarrollo de software, especialmente en sistemas con múltiples capas como GGDeals. El objetivo principal del testing es **verificar que todas las funcionalidades del sistema funcionan correctamente** y que el sistema responde de forma adecuada ante entradas válidas e inválidas.

En GGDeals se han implementado **pruebas unitarias y de integración en el backend**, y se han planificado pruebas manuales para el frontend, con posibilidad de automatización futura.

14.1 Objetivos del Testing

1. Validar la **lógica de negocio** implementada en el backend.
2. Verificar el **funcionamiento correcto de los endpoints REST**.
3. Comprobar la integridad de las operaciones sobre la base de datos.
4. Detectar **errores en fases tempranas del desarrollo**.
5. Garantizar que nuevas funcionalidades **no rompan el sistema existente** (regresión).

14.2 Testing en el Backend

El backend ha sido desarrollado con **Spring Boot**, lo que permite integrar fácilmente bibliotecas de prueba como:

- **JUnit 5**: para pruebas unitarias.
- **Spring Boot Test**: para pruebas de integración con contexto completo.
- **Testcontainers**: para lanzar bases de datos reales (como MySQL) en contenedores durante la prueba.

Pruebas Unitarias

- Se aplican principalmente a los **servicios (`Service`)** y componentes con lógica de negocio aislada.
- Utilizan objetos simulados (`Mock`) para evitar dependencias con base de datos o red.

Ejemplo:

```
java
CopiarEditar
@Test
void testCalculateTotalPrice() {
    BigDecimal price = gameService.calculateTotal(2, new BigDecimal("29.99"));
    assertEquals(new BigDecimal("59.98"), price);
}
```

Pruebas de Integración

- Ejecutan escenarios completos que implican acceso a base de datos y API.
- Se utiliza la anotación `@SpringBootTest` junto a **Testcontainers** para levantar una base de datos real en un contenedor temporal.

Ejemplo:

```
java
CopiarEditar
@Test
void testCreateAndFetchUser() {
    User user = new User("test@example.com", "password");
    userRepository.save(user);
    Optional<User> found = userRepository.findByEmail("test@example.com");
    assertTrue(found.isPresent());
}
```

14.3 Estructura del Código de Pruebas

Ubicado en:

```
css
CopiarEditar
ggdeal.api/
└── src/test/java/com/ggdeal/
    ├── controller/   → Pruebas a endpoints REST
    ├── service/      → Pruebas unitarias a lógica de negocio
    ├── repository/   → Pruebas sobre JPA y queries personalizadas
```

14.4 Cobertura de Pruebas

Actualmente se han probado las siguientes funcionalidades del backend:

- Registro e inicio de sesión de usuarios.
- Validación de tokens JWT.
- Creación y consulta de videojuegos.
- Consulta de ediciones y plataformas.
- Gestión de claves y ventas.

Las funcionalidades más recientes como reservas están pendientes de ser cubiertas.

14.5 Testing del Frontend

Pruebas Manuales

Hasta el momento, el frontend se ha testado de forma manual, verificando:

- Navegación entre rutas.
- Comprobación visual de elementos.
- Validaciones en formularios.
- Comportamiento del carrito.
- Comunicación con la API.

Testing Automatizado (Futuro)

Está prevista la integración de:

Herramienta	Propósito
Jest	Framework de pruebas para React y JS.
React Testing Library	Permite testear componentes desde el punto de vista del usuario.
Cypress	Testing de extremo a extremo (E2E).

14.6 Automatización de Pruebas

Se prevé en el futuro incluir un pipeline de CI (Integración Continua) con herramientas como GitHub Actions, que ejecute las pruebas automáticamente ante cada cambio en el repositorio.

14.7 Resultados y Conclusión

El testing ha permitido detectar y corregir errores en fases tempranas del desarrollo, reducir el riesgo de regresiones y mejorar la confianza en el funcionamiento del sistema. La integración de Testcontainers ha sido especialmente útil para simular entornos reales de base de datos en los tests.

A medida que se incorporen nuevas funcionalidades y componentes en el frontend, se ampliará la cobertura con pruebas automatizadas también del lado del cliente.

15. Retos Técnicos y Soluciones

Durante el desarrollo del proyecto GGDeals, se presentaron diversos desafíos técnicos en las distintas capas del sistema: frontend, backend, base de datos, despliegue y pruebas. Esta sección documenta los principales problemas encontrados y las soluciones adoptadas, destacando el enfoque de aprendizaje práctico y adaptación a entornos reales que ha guiado el trabajo.

15.1 Integración Frontend–Backend

- Problema: La primera integración entre React (frontend) y Spring Boot (backend) presentó errores de CORS (Cross-Origin Resource Sharing), impidiendo que las peticiones HTTP funcionaran correctamente entre ambos servicios.

- Solución:

- Se habilitó la configuración CORS global en Spring Security mediante la clase `SecurityConf.java`, permitiendo solicitudes desde `http://localhost:5173`.
- Se ajustaron las cabeceras y métodos permitidos en el filtro de seguridad.

```
java
CopiarEditar
http.cors().configurationSource(request -> {
    CorsConfiguration config = new CorsConfiguration();
    config.setAllowedOrigins(List.of("http://localhost:5173"));
    config.setAllowedMethods(List.of("GET", "POST", "PUT", "DELETE"));
    config.setAllowCredentials(true);
    config.setAllowedHeaders(List.of("*"));
    return config;
});
```

15.2 Configuración de JWT en Cookies

- Problema: Inicialmente, el token JWT se almacenaba en `localStorage`, lo que generaba problemas de seguridad y dificultad para mantener la sesión de usuario entre pestañas.

- Solución:

- Se migró el sistema para que el backend devuelva el JWT en una **cookie segura (`HttpOnly`)**, inaccesible desde JavaScript.
- Se modificó el cliente React para que las peticiones incluyan automáticamente las cookies (`credentials: 'include'`).

15.3 Gestión de Claves Digitales

- Problema: Era necesario asociar múltiples claves digitales únicas por edición y plataforma, evitando ventas duplicadas y permitiendo reservas si no hay stock.

- Solución:

- Se creó la tabla `replicas` con campos `edition_id`, `platform_id`, `code` y `available`.
- Al confirmar una venta, el backend selecciona una réplica libre y la marca como `disponible = false`.
- Si no hay stock, se ofrece al usuario la opción de reserva, que queda pendiente.

15.4 Contenerización con Docker

- Problema: Al intentar ejecutar backend y frontend en contenedores separados, surgieron errores de red entre servicios y sincronización al iniciar la base de datos.

- Solución:

- Se configuró `docker-compose.yaml` con dependencias (`depends_on`) y tiempos de espera controlados.
- Se ajustaron las variables de entorno del backend para apuntar al nombre del servicio de base de datos, no a `localhost`.

```
yaml
CopiarEditar
SPRING_DATASOURCE_URL: jdbc:mysql://database:3306/ggdeal
```

15.5 Estructura del Modelo de Datos

- Problema: Algunos casos de uso requerían relaciones complejas no contempladas en el modelo inicial, como juegos con múltiples características o plataformas.

- Solución:

- Se incorporaron **tablas auxiliares muchos-a-muchos**: `aux_game_platform` y `aux_game_feature`.
- Se adaptaron los repositorios y servicios a las nuevas relaciones.

15.6 Validación y Seguridad

- Problema: Algunos formularios en el frontend no validaban adecuadamente los campos antes de enviarlos, lo que generaba errores silenciosos en el backend.

- Solución:

- Se añadió validación HTML5 y validación con TypeScript en los formularios.
- Se utilizaron mensajes visuales de error y confirmación en la interfaz.

15.7 Testing con Base de Datos Real

- Problema: Las pruebas de integración en el backend requerían conexión con una base de datos real, pero no era práctico utilizar la misma base de desarrollo.

- Solución:

- Se integró **Testcontainers** para lanzar contenedores MySQL temporales durante las pruebas.
- Esto permitió simular un entorno real sin comprometer datos de desarrollo.

15.8 Coordinación entre Rutas y Autenticación

- Problema: El acceso a rutas protegidas en el frontend (como el panel de administración) debía limitarse al rol correspondiente.

- Solución:

- Se implementó un sistema de control de acceso basado en el token JWT decodificado en frontend.
- Se protegieron rutas sensibles con redirecciones automáticas si el rol no es adecuado.

15.9 Recursos Limitados

- Problema: Algunas decisiones de implementación (por ejemplo, no usar librerías externas para el diseño visual) se debieron a restricciones de tiempo y aprendizaje autodidacta.

- Solución:

- Se priorizó el desarrollo con herramientas esenciales.
- El diseño se elaboró desde cero con CSS personalizado, adaptado al mockup visual.

15.10 Gestión del Tiempo y Alcance

- Problema: Se hizo evidente que la implementación completa de todas las funcionalidades (por ejemplo, validación por email) no era viable en el tiempo disponible.

- Solución:

- Se definió un **MVP (Producto Mínimo Viable)** claro.
- Se documentaron las funcionalidades pendientes como mejoras futuras (ver sección 16).

16. Conclusiones

El desarrollo de la plataforma GGDeals como proyecto final del ciclo formativo de **Desarrollo de Aplicaciones Web** ha supuesto una experiencia integral que abarca todos los aspectos clave del desarrollo web moderno. Desde la planificación y diseño de interfaces hasta la implementación técnica de una arquitectura cliente-servidor completa, el proyecto ha permitido aplicar y reforzar los conocimientos adquiridos durante el ciclo, además de explorar nuevas tecnologías por cuenta propia.

16.1 Logros Técnicos Alcanzados

1. Se ha implementado un sistema web funcional con separación de capas: **frontend**, **backend** y **base de datos**, orquestadas mediante **Docker**.
2. El **frontend** se ha desarrollado con **React** y **TypeScript**, siguiendo una estructura modular basada en rutas, componentes y lógica desacoplada.
3. El **backend** se ha construido con **Spring Boot**, implementando una **API RESTful** robusta y segura, con control de roles mediante **JWT**.
4. Se ha diseñado un modelo de datos **relacional y normalizado** en **MySQL**, capaz de gestionar la complejidad de una tienda online de videojuegos.
5. La aplicación ha sido contenida en **Docker**, permitiendo su despliegue y ejecución de forma reproducible.
6. Se han implementado **pruebas unitarias y de integración** en el backend con **JUnit** y **Testcontainers**.

16.2 Conocimientos Aplicados

Durante el desarrollo se han puesto en práctica competencias esenciales del desarrollo web profesional:

- **Programación orientada a objetos (Java y TypeScript)**.
- **Diseño y uso de APIs REST**.
- **Uso de bases de datos relacionales** con diseño en 3FN.
- **Gestión de control de versiones** con Git y GitHub.
- **Contenerización y orquestación** de servicios con Docker y Docker Compose.
- **Diseño centrado en el usuario**, accesibilidad y usabilidad en la interfaz.

Además, se han incorporado tecnologías y prácticas que no formaban parte del currículo oficial pero son esenciales en el mundo laboral actual, como JWT, testing con Testcontainers o despliegue en contenedores.

16.3 Dificultades Superadas

El proyecto ha supuesto un reto realista con dificultades técnicas significativas. Algunos de los desafíos más relevantes fueron:

- La integración de frontend y backend con autenticación segura.
- La modelización de relaciones complejas en la base de datos.
- La gestión de claves digitales únicas por edición y plataforma.
- La contenerización completa de todos los servicios y su comunicación.

Estas dificultades se abordaron mediante análisis, consulta de documentación oficial, foros especializados y aprendizaje autónomo.

16.4 Impacto Formativo

A nivel personal y académico, este proyecto ha sido una **experiencia altamente formativa**, donde se ha ejercitado no solo la capacidad técnica, sino también la **autonomía, organización, resolución de problemas y toma de decisiones tecnológicas**.

Este enfoque práctico ha reforzado la preparación para acceder al mercado laboral con una base sólida y actualizada.

16.5 Líneas de Mejora Futura

Aunque el sistema cumple con su propósito como prototipo funcional, existen áreas donde se podrían aplicar mejoras en el futuro:

- ✗ Añadir **pasarelas de pago reales** (PayPal, Stripe).
- ✗ Incorporar un **modo responsive completo** para dispositivos móviles.
- ✗ Implementar **notificaciones por correo electrónico** ante reservas completadas.
- ✗ Mejorar la **interfaz administrativa** con dashboards gráficos.
- ✗ Añadir pruebas automatizadas en el frontend con **Jest o Cypress**.
- ✗ Crear un **sistema de valoraciones y reseñas de juegos**.
- ✗ Publicar la aplicación en un entorno de producción (VPS o PaaS).

La conclusión general es que el desarrollo de GGDeals ha sido una experiencia exitosa, que cumple con los objetivos técnicos, funcionales y formativos del módulo FCT, y que representa un proyecto web completo, profesional y escalable.

17. Bibliografía y Recursos Consultados

A lo largo del desarrollo de GGDeals se ha recurrido a múltiples fuentes de documentación técnica, tutoriales, documentación oficial y comunidades en línea. Esta sección recoge todas las referencias que han servido de apoyo, tanto para implementar funcionalidades como para resolver problemas técnicos y adquirir nuevos conocimientos.

17.1 Documentación Oficial

- **Spring Boot:** https://spring.io/projects/spring-boot
- **Spring Security:** https://docs.spring.io/spring-security/reference/index.html
- **Spring Data JPA:** https://docs.spring.io/spring-data/jpa/docs/current/reference/html/
- **JSON Web Tokens (JWT):** https://jwt.io/introduction
- **ReactJS:** https://reactjs.org/
- **TypeScript:** https://www.typescriptlang.org/docs/
- **Vite (Build Tool):** https://vitejs.dev/
- **Docker:** https://docs.docker.com/
- **Docker Compose:** https://docs.docker.com/compose/
- **MySQL:** https://dev.mysql.com/doc/
- **JUnit 5:** https://junit.org/junit5/docs/current/user-guide/
- **Testcontainers:** https://www.testcontainers.org/
- **Git:** https://git-scm.com/doc
- **GitHub:** https://docs.github.com/

17.2 Artículos y Tutoriales

- **Baeldung (Java & Spring):** https://www.baeldung.com/
- **FreeCodeCamp (React y Web Dev):** https://www.freecodecamp.org/news/tag/react/
- **DigitalOcean Community:** https://www.digitalocean.com/community/tutorials
- **Stack Overflow (consultas técnicas puntuales):** https://stackoverflow.com/
- **Java Guides:** https://www.javaguides.net/
- **Dev.to – JWT con Spring:** https://dev.to/devtony101/spring-boot-jwt-setup-2om

17.3 Recursos Educativos y Visuales

- **GGDeals (referencia de diseño):** https://gg.deals/
- **Figma (para análisis de mockups):** https://www.figma.com/
- **Canva (referencia para estructura visual):** https://www.canva.com/
- **FontAwesome (iconos):** https://fontawesome.com/

17.4 Bibliotecas y Frameworks Usados

- **React Router:** https://reactrouter.com/
- **Axios (HTTP):** https://axios-http.com/
- **BCrypt para Java:** https://github.com/patrickfav/bcrypt
- **Serve (frontend estático):** https://www.npmjs.com/package/serve
- **Vite Plugin React:** https://github.com/vitejs/vite-plugin-react

17.5 Herramientas de Desarrollo

- **Visual Studio Code:** https://code.visualstudio.com/
- **IntelliJ IDEA:** https://www.jetbrains.com/idea/
- **Postman (testing manual):** https://www.postman.com/
- **DBeaver (gestor de BBDD):** https://dbeaver.io/
- **Docker Desktop:** https://www.docker.com/products/docker-desktop