Diseño y desarrollo de una aplicación web para la gestión de una tienda de juegos de mesa

Paul Daniel Pop

30 de junio de 2025

Resumen– Este trabajo presenta el desarrollo de una aplicación web para la gestión de una tienda local especializada en juegos de mesa y cartas coleccionables, como Magic: The Gathering. La plataforma permite a los usuarios consultar y adquirir productos, inscribirse en eventos como torneos y mantenerse informados sobre nuevos lanzamientos. Para el personal administrador, el sistema ofrece funcionalidades para crear y modificar productos, planificar eventos, gestionar pedidos y acceder a métricas clave del negocio. Estas métricas incluyen las ganancias del mes actual, la evolución de ingresos durante los últimos tres meses mediante una gráfica comparativa, el número de usuarios activos y el estado de los pedidos pendientes. La aplicación sigue una arquitectura limpia y modular, estructurada en capas (dominio, infraestructura y aplicación), y ha sido desarrollada utilizando tecnologías modernas como Next.js, Supabase y autenticación mediante JWT. El sistema busca optimizar la operativa diaria de la tienda y mejorar la experiencia del cliente a través de una interfaz accesible, eficiente y adaptada a sus necesidades.

Palabras clave– Aplicación web, Arquitectura por capas, Gestión de tienda, Eventos y productos, Base de datos PostgreSQL, Supabase, Next.js, Prisma, JWT

Abstract– This project presents the development of a web application for the comprehensive management of a local store specializing in board games and collectible card games, such as Magic: The Gathering. The platform allows users to browse and purchase products, register for events like tournaments, and stay informed about new product releases. For store administrators, the system provides features to create and update products, schedule events, manage orders, and access key business metrics. These metrics include current monthly revenue, a comparative chart of revenue over the past three months, the number of active users, and the status of pending orders. The application follows a clean and modular architecture, structured in layers (domain, infrastructure, and application), and has been developed using modern technologies such as Next.js, Supabase, and JWT-based authentication. The system aims to optimize the store's daily operations and enhance the customer experience through an accessible, efficient, and user-focused interface.

Keywords– Web application, Layered architecture, Store management, Events and products, PostgreSQL database, Supabase, Next.js, Prisma, JWT

1 Introducción - Contexto del trabajo

Las tiendas locales especializadas en juegos de mesa y cartas coleccionables, como Magic: The Gathering, cumplen un rol clave en la creación de comunidades de jugadores. Sin embargo, muchas carecen de herramientas digitales adaptadas a sus necesidades, lo que dificulta la gestión de productos, eventos e interacción con los clientes.

• E-mail de contacto: 1607129@uab.cat — pauldaniel02@outlook.com
• Mención realizada: Ingeniería del Software
• Trabajo tutorizado por: Sergio Bachiller Rubia (Departamento de Ciencias de la Computación)
• Curso 2024/25

Actualmente, muchas de estas tiendas recurren a hojas de cálculo, redes sociales o mensajería instantánea, soluciones que resultan insuficientes a medida que crecen la actividad y la comunidad. Esto afecta tanto a la eficiencia operativa como a la experiencia del cliente, quien suele encontrarse con catálogos desactualizados o sistemas de inscripción poco estructurados.

Este trabajo propone como solución una aplicación web personalizada que digitaliza los procesos clave de una tienda física: gestión de productos, pedidos, eventos y métricas. Desarrollada con tecnologías modernas como Next.js, Supabase y autenticación JWT, la plataforma busca cubrir necesidades reales de una tienda de este estilo y sentar las bases para una solución escalable que pueda ser mejorada y aplicable a otros negocios del sector.

2 Objetivos

El objetivo principal de este Trabajo de Fin de Grado es desarrollar una aplicación web destinada a mejorar la gestión interna de una tienda de juegos de mesa, con un enfoque especial en la experiencia del usuario y la eficiencia operativa. Los objetivos específicos son los siguientes:

1. **Mejorar la experiencia de los clientes:**
    - Permitir la consulta de productos disponibles, incluyendo información sobre stock y preventas.
    - Facilitar la inscripción y seguimiento de eventos como torneos.
    - Centralizar la información importante sobre productos y eventos en un solo lugar.
2. **Optimizar la eficiencia de gestión:**
    - Sustituir métodos tradicionales como hojas de cálculo, grupos de WhatsApp, Telegram u otras redes sociales.
    - Permitir a los administradores crear y modificar productos, planificar eventos, gestionar pedidos y roles de usuarios.
    - Proporcionar un panel de métricas clave como ingresos del mes actual, evolución de ingresos, pedidos pendientes y usuarios activos.
3. **Diseñar una solución técnica escalable:**
    - Implementar una arquitectura modular y limpia basada en capas (dominio, infraestructura y aplicación).
    - Utilizar tecnologías modernas como Next.js, Supabase y JWT para autenticación.
    - Facilitar la ampliación futura de funcionalidades, como pasarelas de pago o notificaciones en tiempo real.

3 Estado del Arte

El nivel de digitalización entre tiendas de juegos de mesa varía ampliamente. Algunas con mayor capacidad, como MagicBarcelona o ManaVortex, disponen de plataformas a medida con funcionalidades avanzadas como catálogos, gestión de eventos y ecommerce. Sin embargo, la mayoría de tiendas locales dependen de herramientas genéricas como grupos de WhatsApp o formularios de Google, lo que provoca pérdida de información y procesos poco eficientes.

Frente a esto, el desarrollo web moderno ofrece soluciones específicas mediante arquitecturas modulares y tecnologías como Next.js, Supabase o JWT. Estas permiten crear aplicaciones robustas, escalables y adaptadas al sector. Además, el uso de almacenamiento en la nube facilita la gestión de contenido multimedia sin necesidad de infraestructura adicional.

Este proyecto toma como referencia las carencias detectadas en tiendas locales y las buenas prácticas observadas en modelos más avanzados, proponiendo una plataforma flexible y orientada a resolver las necesidades concretas del sector.

4 Metodología

Para el correcto desarrollo de este proyecto se ha seguido una metodología ágil híbrida entre scrum y estrategia incremental, que ha permitido iterar progresivamente sobre las funcionalidades clave del sistema. Este enfoque ha resultado especialmente útil para el desarrollo de un proyecto individual como es el caso, para adaptar el desarrollo a los requisitos cambiantes y facilitar una validación temprana de los módulos implementados.

El trabajo se ha estructurado en sprints de dos semanas, cada uno centrado en la entrega de funcionalidades completas. Al finalizar cada sprint se evaluaron los avances y se ajustaron las tareas del siguiente ciclo en función de las prioridades técnicas y mejoras detectadas en las funcionalidades. Este modelo iterativo ha favorecido la entrega continua de funcionalidades y ayudado en la organización y seguimiento del desarrollo.

La organización y seguimiento de las tareas se ha llevado a cabo mediante la herramienta **Trello**, donde se creó un tablero dividido en columnas correspondientes al flujo de trabajo: “Backlog”, “In Progress” y “Done”. Este sistema ha permitido tener una representación visual clara del estado de cada funcionalidad y mantener una gestión eficiente del tiempo y los objetivos durante cada etapa.

5 Planificación

La planificación del proyecto se ha estructurado en varias fases, cada una asociada a uno o más sprints, distribuidos a lo largo del cuatrimestre. A continuación se detallan las principales etapas del desarrollo:

- **Fase 1: Análisis de requisitos y diseño preliminar (Sprint 1)**: En esta etapa se realizó un estudio de las

necesidades del negocio y se definieron las funcionalidades principales que debía cubrir la plataforma. Se diseñó la estructura inicial de la aplicación mediante diagramas UML, la arquitectura, frameworks y herramientas que se utilizarían en el desarrollo y su correspondiente configuración.

- **Fase 2: Sistema de autenticación y roles (Sprint 2):** Se desarrolló el sistema de registro, inicio de sesión y recuperación de contraseña para los usuarios mediante tecnologías como JWT.
- **Fase 3: Módulo de tienda para clientes (Sprint 3 y 4):** Se desarrollaron las páginas y funcionalidades que permitan a los clientes navegar a través del catálogo de productos aplicando filtros, gestionar el contenido de su carrito de compras, hacer pedidos y consultar su estado en un historial de pedidos.
- **Fase 4: Módulo de Eventos para clientes (Sprint 5):** Se ha desarrollado el calendario donde los clientes pueden consultar la información e inscribirse a los eventos disponibles. También se habilitó un apartado para consultar el historial de eventos en los que ha participado el cliente.
- **Fase 5: Sistema de gestión de productos y pedidos para administradores (Sprint 6 y 7):** Se implementó la funcionalidad para que los administradores pudieran crear, modificar y visualizar productos, además de poder gestionar los pedidos realizados por los clientes.
- **Fase 6: Sistema de gestión de eventos y torneos (Sprint 8):** Se implementó la funcionalidad para que los administradores pudieran crear y modificar eventos, además de poder consultar los participantes registrados hasta el momento.
- **Fase 7: Sistema de métricas y gestión de usuarios (Sprint 8):** Se habilitó un panel de métricas para visualizar datos comerciales de interés y la posibilidad de dar o quitar el rol de administrador a usuarios.

6 Diseño de la aplicación

Antes de empezar con el desarrollo se hizo un estudio y diseño de la arquitectura del sistema, los patrones de diseño y los diagramas de UML preliminares que respaldan las decisiones técnicas tomadas para este proyecto.

6.1. Visión general

La plataforma se apoya en una **arquitectura en capas (N-Tier / Layered)** [1] [2] aplicada tanto al frontend como al backend. Cada capa posee responsabilidades claramente delimitadas y comunica solo con la capa adyacente, lo que implementa el principio Single-Responsibility a nivel de sistema y favorece el desacoplamiento, mantenibilidad y escalabilidad del sistema [3].

6.2. Capas del Frontend

En el frontend, la aplicación se organiza en las siguientes capas:

- **Capa de Presentación (Interfaz de Usuario):** Se encarga de mostrar la información y gestionar la interacción con el usuario. Aquí se implementan los elementos visuales, formularios y controles de navegación.
- **Capa de Aplicación (Casos de Uso):** Esta capa se encarga de manejar la lógica de negocio que no está directamente relacionada con la presentación de la interfaz de usuario, pero que sí son importantes en el comportamiento y flujo de la aplicación. En esta capa, se gestionan aspectos como estados globales de la aplicación, manejo de localStorage o interacciones con lógica de negocio.
- **Capa de Dominio (Modelo):** Define las entidades fundamentales y sus relaciones (por ejemplo, Usuario, Producto, Evento). Estas entidades representan la lógica central del negocio, sin depender de detalles de presentación o de la base de datos.
- **Capa de Infraestructura (Integración con APIs):** Se encarga de la comunicación con el backend y la definición e implementación de servicios externos como por ejemplo Supabase Storage o servicios propios para manejar llamadas http.

6.3. Capas del Backend

El Backend sigue una estructura similar, con las siguientes capas:

- **Capa de Presentación (Controladores/Endpoints de APIs):** Los controladores reciben las solicitudes HTTP del frontend, realizan tareas de validación de roles y envían la petición a la capa de aplicación para su procesamiento, devolviendo las respuestas apropiadas y manejando excepciones.
- **Capa de Aplicación (Casos de Uso):** Aquí está presente la lógica del sistema, que procesa las operaciones como el registro de usuarios, gestión de productos y torneos, y maneja la interacción con la base de datos.
- **Capa de Dominio (Modelo):** Contiene las entidades y las reglas que rigen su comportamiento. Estas entidades son independientes de los detalles técnicos, lo que permite que la lógica de negocio sea reutilizable y fácilmente testeable.
- **Capa de Infraestructura (Servicios externos):** Se encarga de la implementación y conexión de la base de datos y otros servicios externos, por ejemplo servicios de encriptación de contraseñas, generación de tokens JWT o servicios de envío de emails.

6.4. Patrones de diseño adicionales

- **Repository:** [4] Permite abstraer y centralizar el acceso a los datos desde la lógica de negocio. Este patrón actúa como un intermediario entre la capa de dominio y la capa de infraestructura, proporcionando una interfaz clara y consistente para realizar operaciones sobre entidades como usuarios, productos o eventos, sin depender de detalles específicos como la tecnología usada para la base de datos. De esta manera

se consigue desacoplar la lógica del dominio de los detalles técnicos de acceso de datos, facilitando el mantenimiento del sistema y la posibilidad de cambiar las tecnologías de persistencia sin afectar otras partes del sistema.

Aplicado a este proyecto, se define una interfaz de repositorio en la capa de dominio que especifica las operaciones básicas (como findById, findAll, save o delete) y los datos de entrada y salida esperados. Esta interfaz es implementada posteriormente en la capa de infraestructura, donde se utilizan las tecnologías concretas (por ejemplo, PostgreSQL) para ejecutar las operaciones de acceso a la base de datos.

- **Inyección de Dependencias (DI):** [5] Permite proporcionar a una clase los objetos que necesita para funcionar (sus dependencias), en lugar de que los cree por sí misma. De este modo, se consigue un menor acoplamiento entre componentes y una mayor flexibilidad a la hora de sustituir implementaciones.

  En el contexto de esta aplicación, la inyección de dependencias se aplica especialmente al trabajar con el patrón **Repository** mencionado anteriormente. La implementación concreta de la interfaz repository es inyectada en los casos de uso a través del constructor de la clase.

  Este patrón proporciona los siguientes beneficios:

  1. **Desacoplamiento:** Las clases no dependen de implementaciones concretas, lo que facilita cambios tecnológicos sin afectar al dominio.
  2. **Facilidad de pruebas:** Se pueden inyectar implementaciones simuladas que permiten realizar pruebas unitarias sin acceso a servicios externos.
  3. **Modularidad:** El sistema es más flexible y fácil de extender o mantener, especialmente al añadir nuevas entidades o cambiar la lógica de persistencia.

6.5. Diagramas UML

Durante la fase de diseño se elaboraron varios diagramas UML para planificar la estructura y comportamiento del sistema antes del desarrollo. Concretamente:

- **Clases:** para definir entidades del dominio, atributos y relaciones.
- **Casos de uso:** para identificar actores y funcionalidades clave.
- **Secuencia:** para representar interacciones entre componentes en operaciones representativas.

Estos diagramas sirvieron como referencia inicial, aunque no se generaron para todas las operaciones, evitando redundancias y optimizando el tiempo disponible. Todos los diagramas se incluyen en el apéndice para su consulta detallada18.

7 Requisitos del Sistema

Se han definido los requisitos del sistema en dos categorías principales:

- **Funcionales:** describen las funcionalidades que debe implementar el sistema.
- **No funcionales:** establecen criterios a cumplir como: rendimiento, usabilidad, escalabilidad y seguridad.

Ambos grupos se recogen de forma estructurada en tablas disponibles en el apéndice 1.

8 Desarrollo

En esta sección se describe en detalle la implementación y decisiones técnicas tomadas para ciertas partes de este trabajo de fin de grado. Todo ha sido estructurado según la arquitectura por capas y patrones de diseño explicados anteriormente.

8.1. Frontend

El frontend se ha construido con Next.js y Typescript, aprovechando el enrutado automático, el renderizado híbrido para optimizar tiempos de carga, y por otro lado aprovechando la inferencia de tipo de typescript para poder detectar errores más facilmente durante el desarrollo.

8.1.1. Interfaz de Usuario

Para garantizar un buen diseño y experiencia de usuario se han usado componentes visuales altamente personalizables proporcionados por Shadcn, juntamente con iconos de Lucide-React y notificaciones emergentes (toast) para poder dar feedback en todo momento a los usuarios de la aplicación.

8.1.2. Contextos y hooks personalizados

Para separar la lógica de estado y las llamadas a la API, se han definido *React Contexts* y *Custom Hooks*. Por ejemplo, **ProductContext** expone un hook **useCart** que encapsula operaciones como **fetchWithFilters, createProduct, updateProduct, deleteProduct** y gestiona estados como *loading*, *error*, *pagination* o *filters*.

Este hook puede ser utilizado en varias partes de la aplicación, como por ejemplo en la sección de tienda orientada a los clientes o en el panel para administradores. De esta manera se favorece la reutilización y separación de responsabilidades en el código, además de desacoplar la UI del manejo de estados y operaciones.

8.1.3. Servicio HTTP global con Axios

Se ha implementado un servicio HTTP utilizando Axios configurado globalmente con:

- Timeout de 5000ms para evitar peticiones colgadas.
- Cabecera ”withCredentials: true”para enviar cookies en cada petición.

- Cabeceras predeterminadas como Çontent-Type: application/json".
- Interceptores de respuesta para transformar errores nativos de Axios a mensajes de error claros y más fácilmente interpretables.
- Estructura uniforme de respuesta: todas las peticiones devuelven una estructura consistente, lo que facilita el manejo de los datos y errores producidos en la petición.

La implementación de este servicio ha aportado significantes beneficios como:

- **Configuración centralizada:** cualquier cambio sobre las cabeceras de la petición o en la estructura de la respuesta se hace en un único punto sin afectar otras partes del sistema.
- **Consistencia:** todos los componentes utilizan una instancia de este servicio, garantizando un comportamiento homogéneo.
- **Mantenibilidad:** al abstraer el comportamiento del servicio a través de la interfaz `IHttpService`, simplifica el cambio a otra librería si fuese necesario y permite inyectar este servicio a los casos de uso necesarios.

8.1.4. Carrito de compra

Para esta funcionalidad se planteó un carrito que fuese persistente entre sesiones, con operaciones de añadir, eliminar y actualizar cantidad de producto muy rápidas e inmediatas, y que fuese simple de implementar. Por lo tanto, se tomó la decisión de implementar el carrito usando el almacenamiento local del navegador.

El flujo de datos de la implementación final es el siguiente:

1. Se utiliza un contexto global para el carrito, que al montarse en react, se instancia el repositorio de LocalStorage automáticamente y se crea el estado del carrito a partir de los datos previamente almacenados (si existen). Esto garantiza la persistencia entre sesiones, permitiendo al usuario recuperar su carrito incluso tras cerrar y reabrir el navegador.
2. Cada vez que se ejecuta una acción como añadir, eliminar o modificar la cantidad de un producto en el carrito, el repositorio actualiza automáticamente los datos guardados en LocalStorage y devuelve el carrito actualizado.
3. La función `calculateCart()` recalcula el subtotal, descuentos, costes de envío y total, sincronizando el estado interno del carrito con la información del repositorio.
4. Cuando el usuario decide completar la compra, la información del carrito se envía al backend mediante el servicio HTTP, donde se comprueba el stock disponible y se crea el pedido.

Esta solución, al no depender de un backend para el carrito, reduce la complejidad y simplifica el desarrollo. Además las operaciones de lectura/escritura en LocalStorage son rápidas y cumple con los criterios de velocidad de respuesta.

8.2. Backend

El backend se expone mediante **endpoints** siguiendo el principio REST. Las rutas se implementan con los handlers del App Router de Next.js que permiten definir rutas rápidamente usando la estructura de carpetas del directorio *app/api*. Por ejemplo:

- Para la ruta */api/products*, se crea una carpeta *app/api/users/* que contiene un archivo *route.ts* donde se definen los métodos HTTP necesarios (GET, POST, etc.).
- Para rutas dinámicas como */api/order/[id]*, se crea una carpeta *app/api/orders/[id]/* y dentro se define el archivo *route.ts*. El nombre entre corchetes permite capturar parámetros de ruta (id en este caso).
- También se pueden definir subrutas como */api/profile/avatar*, se estructura como */api/profile/avatar/route.ts*.

8.2.1. Listado de endpoints

Estos son todos los endpoints disponibles:

- **POST /auth/login:** usado para iniciar sesión y generar un JWT [6] y guardarlo en una cookie HttpOnly, accesible solamente desde el servidor [7].
- **POST /auth/register:** usado para registrarse en el sistema.
- **POST /auth/request-password-reset:** usado para solicitar un enlace, enviado al email, para restablecer la contraseña
- **POST /auth/reset-password:** usado para restablecer la contraseña, se debe tener un token válido.
- **POST /auth/sign-out:** usado para terminar la sesión de un usuario, borrando la cookie *authToken*.
- **GET /events:** recuperar todos los eventos o recuperar los eventos asociados a un usuario según su estado (próximos o finalizados).
- **POST /events:** usado para crear un nuevo evento.
- **GET /events/status:** recuperar todos los eventos pero filtrado por estado.
- **PUT /events/[id]:** actualizar los detalles de un evento.
- **DELETE /events/[id]:** eliminar un evento en concreto.
- **POST /events/[id]/available:** consultar si un evento tiene plazas libres.
- **POST /events/[id]/cancel:** cancelar la asistencia a un evento.

- **POST /events/[id]/is-registered:** comprobar si un usuarios está registrado a un evento en concreto.
- **POST /events/[id]/register:** registrar un usuario a un evento en concreto.
- **GET /metrics:** obtener las métricas de negocio visibles desde el panel de administrador.
- **GET /order:** recuperar todos los pedidos o recuperar los pedidos filtrados por su estado (pendientes, cancelados o completados)
- **POST /order/pay:** usado para realizar una compra.
- **GET /order/history:** obtener los pedidos realizados por un usuario y filtrado según su estado.
- **GET /order/metrics:** obtener métricas de negocio relacionadas solamente a pedidos.
- **POST /order/[id]/cancel:** usado para que un usuario pueda cancelar un pedido.
- **PUT /order/[id]:** usado para cambiar el estado de un pedido.
- **GET /products:** obtener productos filtrados según diversos factores (precio, categoría, con descuento, etc.).
- **POST /products:** usado para crear un nuevo producto.
- **GET /products/[id]:** obtener los detalles de un producto en concreto.
- **PUT /products/[id]:** modificar los detalles de un producto en concreto.
- **DELETE /products/[id]:** eliminar un producto en concreto.
- **GET /products/all:** recuperar todos los productos.
- **GET /profile:** obtener información de usuario como nombre, correo electrónico e imágen de perfil.
- **PUT /profile/avatar:** usado para modificar la imágen de perfil.
- **GET /users:** recuperar el nombre y correo electrónico de todos los usuarios de la aplicación.
- **PUT /users:** usado para dar o quitar el rol de administrador a un usuario.

Todos los endpoints usados para crear, eliminar o modificar algún tipo de información o de consultar información sensible, por ejemplo ver todos los usuarios registrados, están protegidos y son accesibles exclusivamente por usuarios administradores.

8.2.2. Middleware de autenticación

Se ha implementado un middleware que comprueba en cada petición la posesión y validez de la cookie de sesión que alberga el JWT. Se utiliza la librería **JOSE** para verificar la firma y extraer el id y el rol del usuario para posteriormente inyectarlos en la petición a través de cabeceras personalizadas visibles solamente en el backend. De esta manera el backend siempre conoce el id y el rol del usuario.

8.2.3. Cambio de contraseña

Para implementar el proceso de recuperación de contraseña, se ha creado una tabla específica en la base de datos llamada **PasswordResetToken**, cuya finalidad es almacenar los tokens generados para este propósito. Los campos de esta tabla son los siguientes:

- **id:** identificador autogenerado.
- **token:** valor del token generado.
- **valid:** bandera para indicar validez.
- **user:** relación con el usuario.
- **expiresAt:** fecha de expiración del token.
- **createdAt:** fecha de creación del token.

El flujo completo del proceso de cambio de contraseña es el siguiente:

1. El usuario solicita recuperar su contraseña desde la interfaz.
2. Se genera un token único y se almacena en la tabla **PasswordResetToken**, con su fecha de expiración y un flag de validez.
3. Se envía un email al usuario con un enlace personalizado que contiene el token como parámetro (por ejemplo: https://mtgshop.com/reset-password?token=abc123).
4. El usuario accede a la página **reset-password** y establece su nueva contraseña.
5. El backend valida el token recibido (verifica que existe, que no ha expirado y que sigue siendo válido).
6. Si el token es válido, se actualiza la contraseña del usuario, se invalida el token, y se completa el proceso.

8.2.4. Servicio de envío de correos electrónicos

El sistema de envío de correos electrónicos se ha implementado usando la librería **nodemailer**, junto con credenciales SMTP de una cuenta de Gmail personal.

El servicio expone dos funciones principales:

- **sendWelcomeEmail():** se usa tras el registro de un usuario nuevo. Envía un correo de bienvenida personalizado con el nombre del usuario.
- **sendResetPasswordEmail():** se utiliza cuando el usuario solicita restablecer su contraseña. Esta función genera un enlace con un token y lo envía al correo del usuario.

8.2.5. Implementación de otras funcionalidades

Siguiendo la arquitectura por capas, cada operación de negocio se implementa como un *Use Case* (por ejemplo, `CreateOrderUseCase`, `UpdateUserRoleUseCase`). Estos casos de uso definen la lógica de aplicación y orquestan la interacción entre las entidades del dominio y los repositorios.

Las entidades del dominio representan los conceptos centrales del negocio (como `Order`, `User`, `Product`) y encapsulan sus reglas internas. Por ejemplo, el caso de uso para crear un pedido no se encarga directamente de insertar en la base de datos, sino de construir correctamente la entidad `Order` y delegar la persistencia a través de un repositorio.

Cada caso de uso depende únicamente de interfaces declaradas en el dominio (por ejemplo, `IOrderRepository`, `IUserRepository`), que se inyectan en su constructor siguiendo el principio de inversión de dependencias explicado en secciones anteriores de este mismo documento. En la capa de infraestructura se hacen las implementaciones concretas de estas interfaces utilizando Prisma ORM, lo que permite mantener una separación clara entre la lógica de negocio y la tecnología empleada para acceder y manipular los datos.

8.3. Infraestructura y despliegue

8.3.1. Entorno local con Docker

Durante el desarrollo local se ha utilizado **Docker** para levantar un contenedor con PostgreSQL mediante un archivo `docker-compose.yml`.

8.3.2. Supabase y Vercel en producción

En producción, la aplicación se apoya en **Supabase** para la base de datos y almacenamiento de imágenes. La base de datos PostgreSQL se aloja en Supabase Database, mientras que las imágenes (productos, perfiles, etc.) se almacenan en Supabase Storage y se accede a ellas mediante URLs públicas guardadas en la propia base de datos.

El frontend y el backend se despliegan en **Vercel**, aprovechando su integración continua con GitHub. Permitiendo que cada vez que se hace un push a la rama principal, Vercel compila y publica automáticamente una nueva versión.

Durante el despliegue surgieron algunos problemas técnicos:

- **Conectividad entre Vercel y Supabase Database**: por defecto, Vercel usa IPv4 para sus funciones serverless, mientras que Supabase Database ofrece conexiones directas solo por IPv6. Esto provocó errores de conexión desde la aplicación desplegada.
- **Solución**: se modificó la estrategia de conexión usando el mecanismo de **connection pooling** de Supabase (basado en PgBouncer), que permite conexiones IPv4 y está pensado para entornos serverless como Vercel. Fue necesario modificar la cadena de conexión y los parámetros de Prisma en producción.

8.4. Pruebas y seguridad de datos

8.4.1. Validación con Zod

En los formularios del frontend se emplea **Zod** para validar los datos de entrada. Esta validación permite detectar errores de forma temprana, mostrar mensajes descriptivos al usuario y evitar enviar datos erróneos al backend.

8.4.2. Pruebas manuales con Postman y pruebas exploratorias

Durante el desarrollo de cada funcionalidad, se han utilizado colecciones de **Postman** para probar manualmente los distintos flujos de los endpoints: casos de éxito, errores esperados, tokens inválidos, inputs mal formateados, etc. Adicionalmente, se ha seguido un enfoque de *exploratory testing* para interactuar con la aplicación como lo haría un usuario final, especialmente tras introducir nuevas funcionalidades, asegurando que las funcionalidades anteriores no se vean afectadas.

8.4.3. Manejo de errores con try/catch

La gestión de errores se ha realizado principalmente mediante bloques `try/catch` en las capas más próximas a la interacción con el exterior:

- En la **capa de infraestructura**, los bloques `try/catch` capturan errores de servicios externos como la base de datos (Prisma). Esto ha permitido envolver errores en estructuras controladas y lanzar excepciones con mensajes claros.
- En la **capa de presentación del backend** (API endpoints), los `try/catch` se utilizan para envolver la ejecución de los casos de uso, capturar cualquier fallo y devolver respuestas HTTP adecuadas al cliente (por ejemplo, códigos 400 o 500 y mensajes descriptivos).
- En la **capa de presentación del frontend**, los bloques `try/catch` se utilizan para capturar errores en las peticiones del cliente y proporcionar *feedback* inmediato al usuario mediante notificaciones emergentes (toasts), como por ejemplo en el proceso de compra o eliminación de productos del carrito.

De esta manera se asegura que los errores se gestionan cerca de donde ocurren y se presentan al usuario de forma clara.

9 RESULTADOS

Pantalla de Inicio de Sesión y Registro: Permite a los usuarios registrarse o iniciar sesión en la aplicación mediante el sistema de autenticación implementado con JWT. Al registrar un usuario el sistema envía automáticamente un email dándole la bienvenida al nuevo cliente.

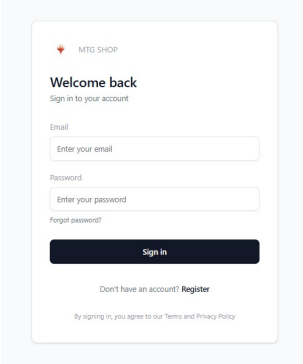

Fig. 1: Login

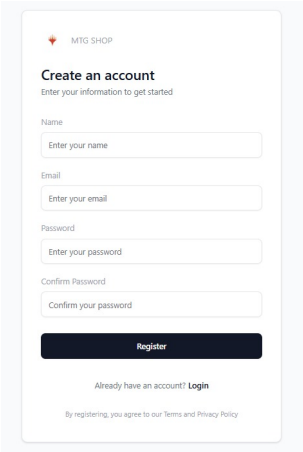

Fig. 2: Register

Pantallas para restablecer contraseña: Si el usuario olvida la contraseña puede clicar al link ”Has olvidado la contaseña?”, se le redirijirá a un formulario donde deberá introducir su email. El usuario recibirá en el correo un link autogenerado para establecer una nueva contraseña.

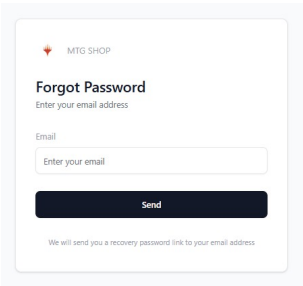

Fig. 3: Formulario Solicitar Nueva Contraseña

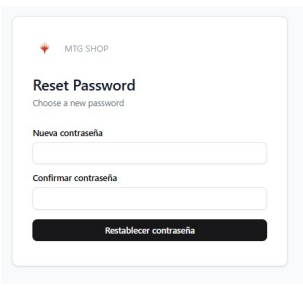

Fig. 4: Reestablecer Contraseña

Pantalla principal de la tienda: Aquí se muestran solamente algunos productos destacados o en preventa. El cliente puede añadir productos al carrito directamente desde esta página.

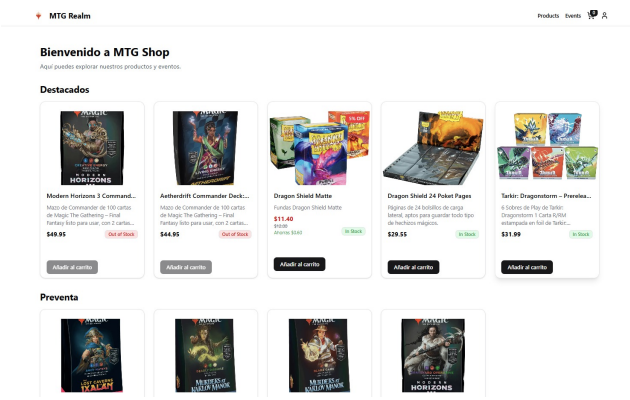

Fig. 5: Pantalla Principal de la Tienda

Pantalla de catálogo de productos: En esta pantalla se muestran todos los productos y el cliente podrá buscar productos mediante la barra de búsqueda o aplicar filtros según el tipo de porducto, rango de precio y criterios adicionales como solo productos en stock, productos en pre-venta y/o productos con descuentos.

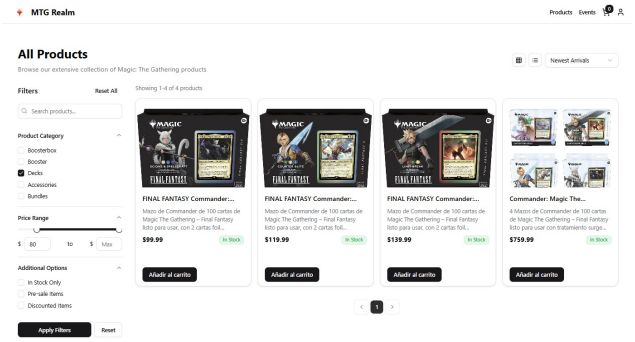

Fig. 6: Página de Productos

Pantalla detallada de producto: Al clicar la tarjeta de un producto el sistema redirige a la página detallada del producto, donde el cliente podrá ver toda la información completa del producto.

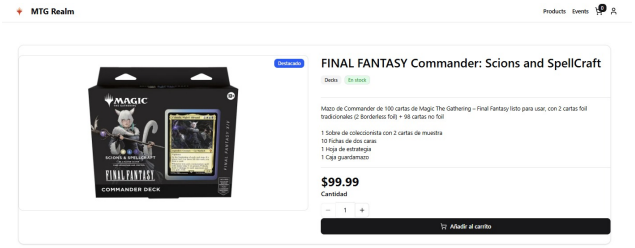

Fig. 7: Página detallada de Producto

Pantalla de carrito de la compra: Al añadir productos al carrito todos aparecerán en la página de carrito. En esta página el cliente puede modificar las cantidades de cada producto, eliminar todos los productos del carrito, ver el resumen del pedido y proceder al pago del pedido. Al clicar el botón de pagar por simplicidad se realiza la compra directamente, se guarda en la base de datos y se actualizan los stocks.

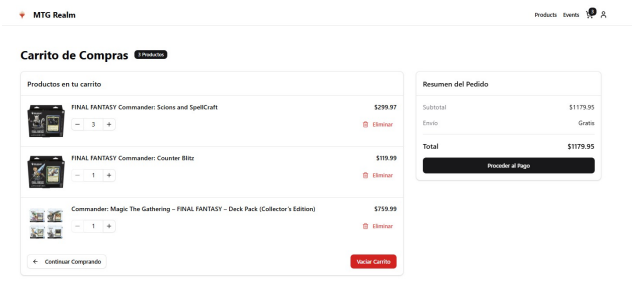

Fig. 8: Página de Carrito

Pantalla de calendario de eventos: En esta página el cliente puede consultar los eventos disponibles en el calendario, ver detalles del evento (fecha, aforo, reglas, premios...) e inscribirse.

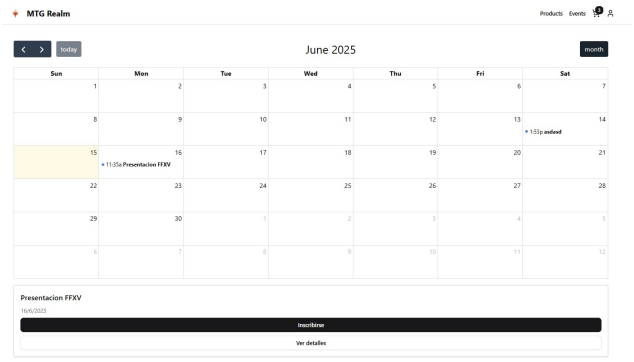

Fig. 9: Página Calendario de Eventos

Pantalla de perfil de usuario: En esta página el cliente puede ver sus datos personales, su historial de pedidos y su historial de eventos. Además, desde esta pantalla el usuario puede modificar su contraseña, cancelar pedidos o eventos que aún no se hayan completado.

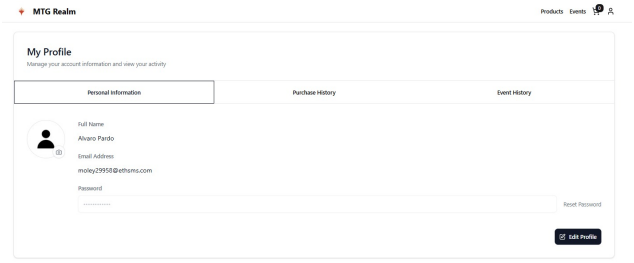

Fig. 10: Página de perfil

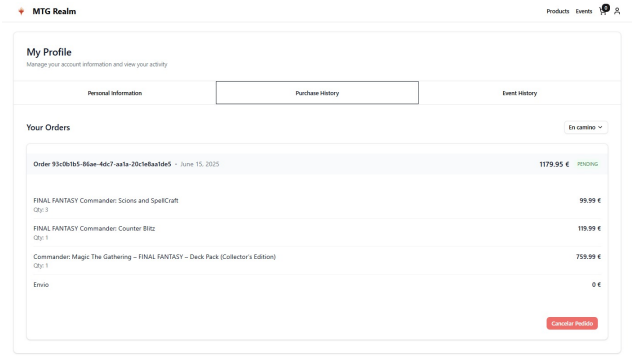

Fig. 11: Historial de Compras

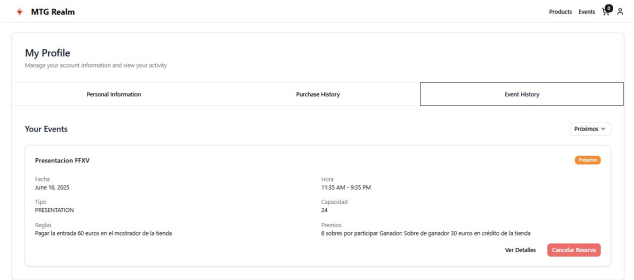

Fig. 12: Historial de Eventos

Pantalla principal del panel admin: Esta pantalla solamente es accesible por administradores de la tienda. Aquí pueden ver algunas métricas de negocio como ganancias este mes, evolución de ganancias en los últimos 3 meses, además de cantidad de pedidos, cantidad de registros a eventos y cantidad de clientes activos (que hayan hecho una compra o participado a un evento) en los últimos 30 días.

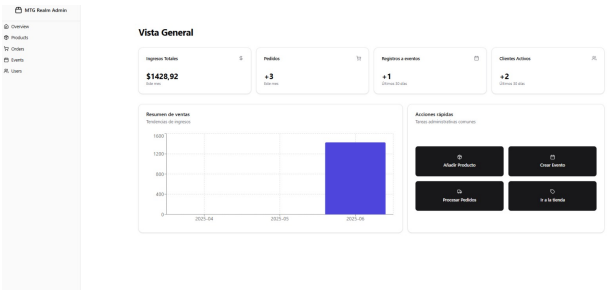

Fig. 13: Página Principal Panel Admin

Página de gestión de productos: En esta página los administradores pueden añadir, modificar o eliminar productos. Al crear o modificar un producto se puede especificar también la imágen que se mostrará. Esta imágen se guarda en Supabase Storage, se genera una url pública que se guarda en la base de datos. Las imágenes se muestran en la interfaz gracias a esta url pública.

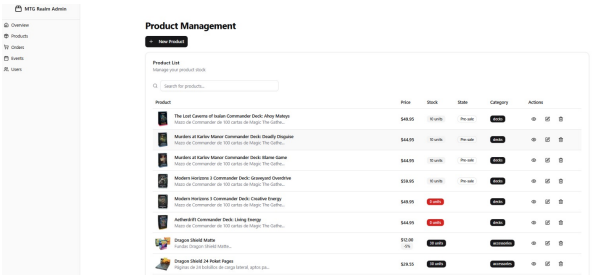

Fig. 14: Pantalla de Gestión de Productos

Pantalla de gestión de pedidos: En esta pantalla los administradores pueden consultar los pedidos del mes según su estado (completado, pendiente, cancelado o todos) y visualizar los detalles, es decir, datos como el comprador, productos, precio final, etc. Además, se muestra información adicional útil como cantidad de pedidos pendientes, completados y ganancias ese mes.

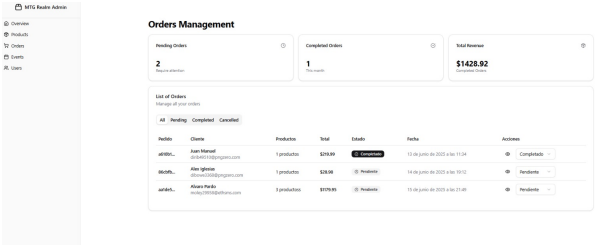

Fig. 15: Pantalla de Gestión de Pedidos

Pantalla de gestión de eventos: En esta pantalla los administradores pueden crear, modificar o eliminar eventos, además de poder consultar los participantes de cada evento.

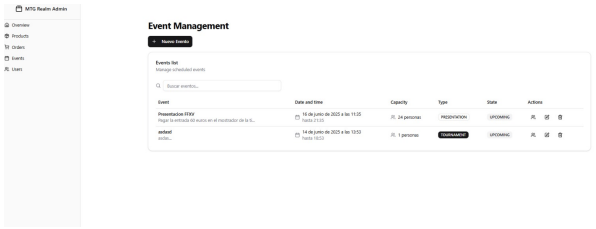

Fig. 16: Pantalla de Gestión de Eventos

Pantalla de gestión de usuarios: En esta pantalla los administradores pueden buscar usuarios y dar o quitar el rol de administrador según vean conveniente.

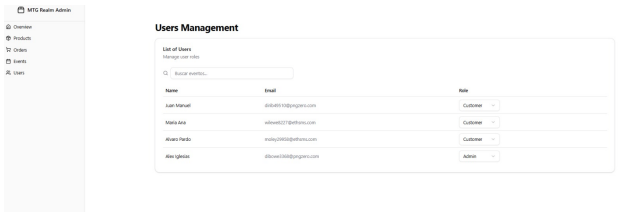

Fig. 17: Pantalla de Gestión de Usuarios

10 Conclusiones

El proyecto ha cumplido su objetivo principal: entregar una versión funcional de una aplicación web para la gestión de una tienda de juegos de mesa. La plataforma cubre los requisitos más relevantes definidos al inicio, dentro del plazo previsto.

Durante el desarrollo se aplicaron conocimientos adquiridos en la carrera como arquitectura por capas, patrones de diseño, bases de datos relacionales y despliegue continuo. El resultado es un sistema estructurado, mantenible y escalable.

Quedan algunas mejoras por implementar si se continúa el proyecto:

- **Integrar una pasarela de pago** para completar el flujo de compra.
- **Mejorar la gestión de cancelaciones** de eventos y pedidos mediante aprobación administrativa.
- **Añadir tests automáticos** para detectar errores de forma temprana.
- **Refinar la interfaz** con una identidad visual propia y estilo más consistente.

A nivel personal, ha sido una experiencia valiosa para consolidar habilidades técnicas y de gestión del tiempo, trabajando con tecnologías modernas como TypeScript, Next.js, Prisma, Supabase o Vercel.

11 Agradecimientos

Antes de finalizar este documento, me gustaría agradecer a mi tutor, Sergio Bachiller, por su orientación a lo largo del desarrollo del Trabajo de Fin de Grado. Gracias por resolver mis dudas, transmitirme confianza y por ayudarme a sacar este proyecto adelante.

También quiero dar las gracias a mi familia, por estar siempre a mi lado y brindarme su apoyo incondicional.

Y, sobre todo, a mi pareja, Jazmin, y a mis amigos: gracias por escucharme cada día hablar del mismo tema, por soportar mis preocupaciones con el TFG y por darme el ánimo necesario para seguir adelante. Gracias por hacer que toda esta etapa sea más llevadera.

Referencias

[1] F. Alavi, “Master Clean Architecture and N-layer Architecture: Never Forget Again!,” *YouTube*, Mar. 20, 2024. [Online]. Available: https://www.youtube.com/watch?v=Kl2tPE2SYBM (accessed Mar. 05, 2025).

[2] N. Ford, M. Richards, P. Sadalage, and Z. Dehghani, *Software Architecture: The Hard Parts*. Sebastopol, CA: O’Reilly Media, 2021.

[3] freeCodeCamp.org, “The SOLID Principles of Object-Oriented Programming Explained in Plain English” *freeCodeCamp.org*, Aug. 28, 2020. [Online]. Available: https://www.freecodecamp.org/news/solid-principles-explained-in-plain-english/ (accessed Mar. 09, 2025).

[4] Per-Erik Bergman, “Repository Design Pattern,” *Medium*, Apr. 20, 2017. [Online]. Available: https://medium.com/@pererikbergman/repository-design-pattern-e28c0f3e4a30 (accessed Apr. 13, 2025).

[5] A. G. Sánchez, “Inyección de dependencias: principio de inversión de dependencias,” *Medium*, May. 02, 2019. [Online]. Available: https://medium.com/@antoniojesussg96/inyecci%C3%B3n-de-dependencias-principio-de-inversi%C3%B3n-de-dependencias-96ebf7e3d885 (accessed Apr. 13, 2025).

[6] JWT.io, “JWT.IO - JSON Web Tokens Introduction,” *jwt.io*, 2024. [Online]. Available: https://jwt.io/introduction (accessed Apr. 13, 2025).

[7] I. Tzortzis, “Cookie Security Best Practices,” *Medium*, Sep. 30, 2022. [Online]. Available: https://medium.com/@iason.tzortzis/cookie-security-best-practices-d19800ad0b4b (accessed Apr. 13, 2025).

Apéndice

Diagramas UML

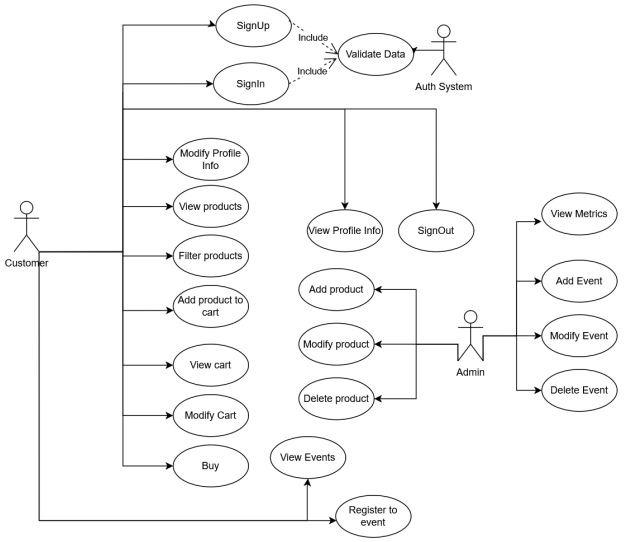

Fig. 18: Diagrama de Casos de Uso

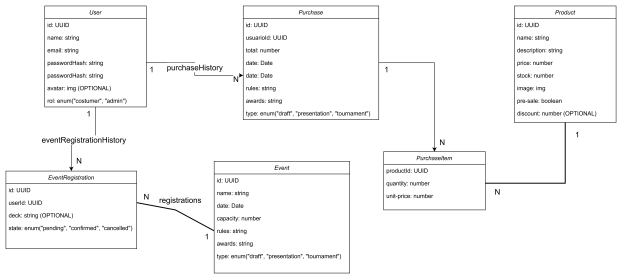

Fig. 19: Diagrama de Clases

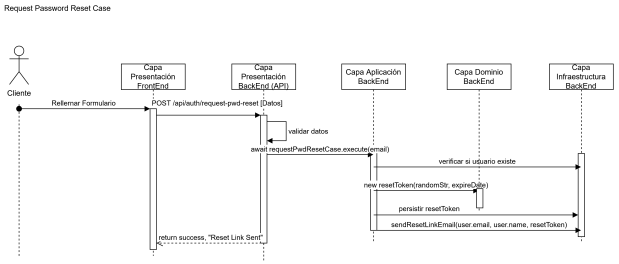

Fig. 20: Caso de Uso: solicitar cambio de contraseña

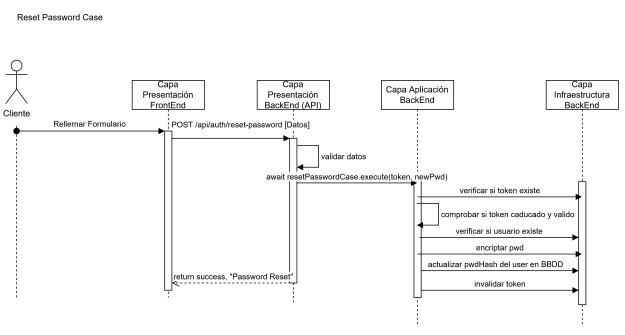

Fig. 21: Caso de Uso: reestablcer contrseña

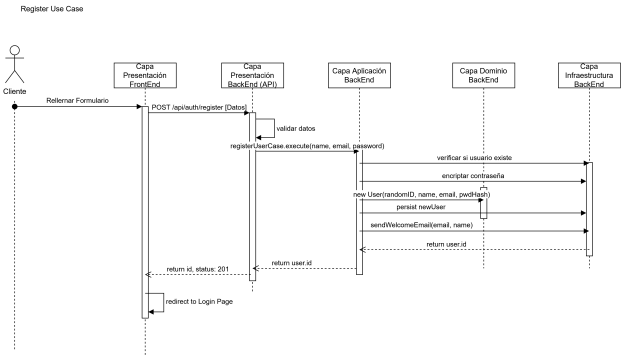

Fig. 22: Caso de Uso: registrar usuario

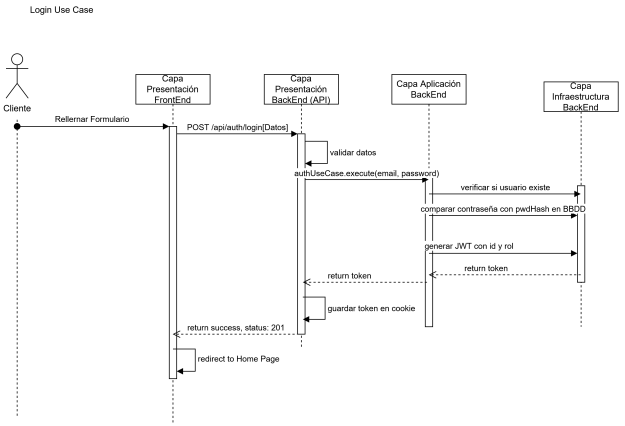

Fig. 23: Caso de Uso: login de usuario

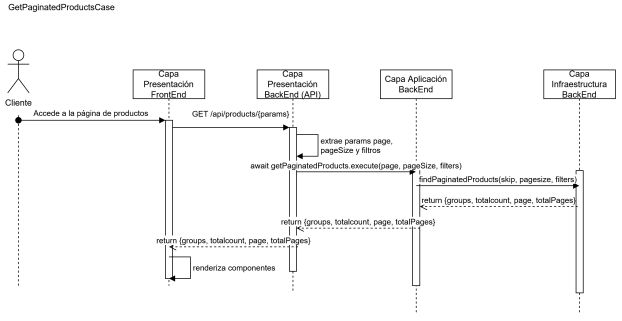

Fig. 24: Caso de Uso: recuperar productos con filtro

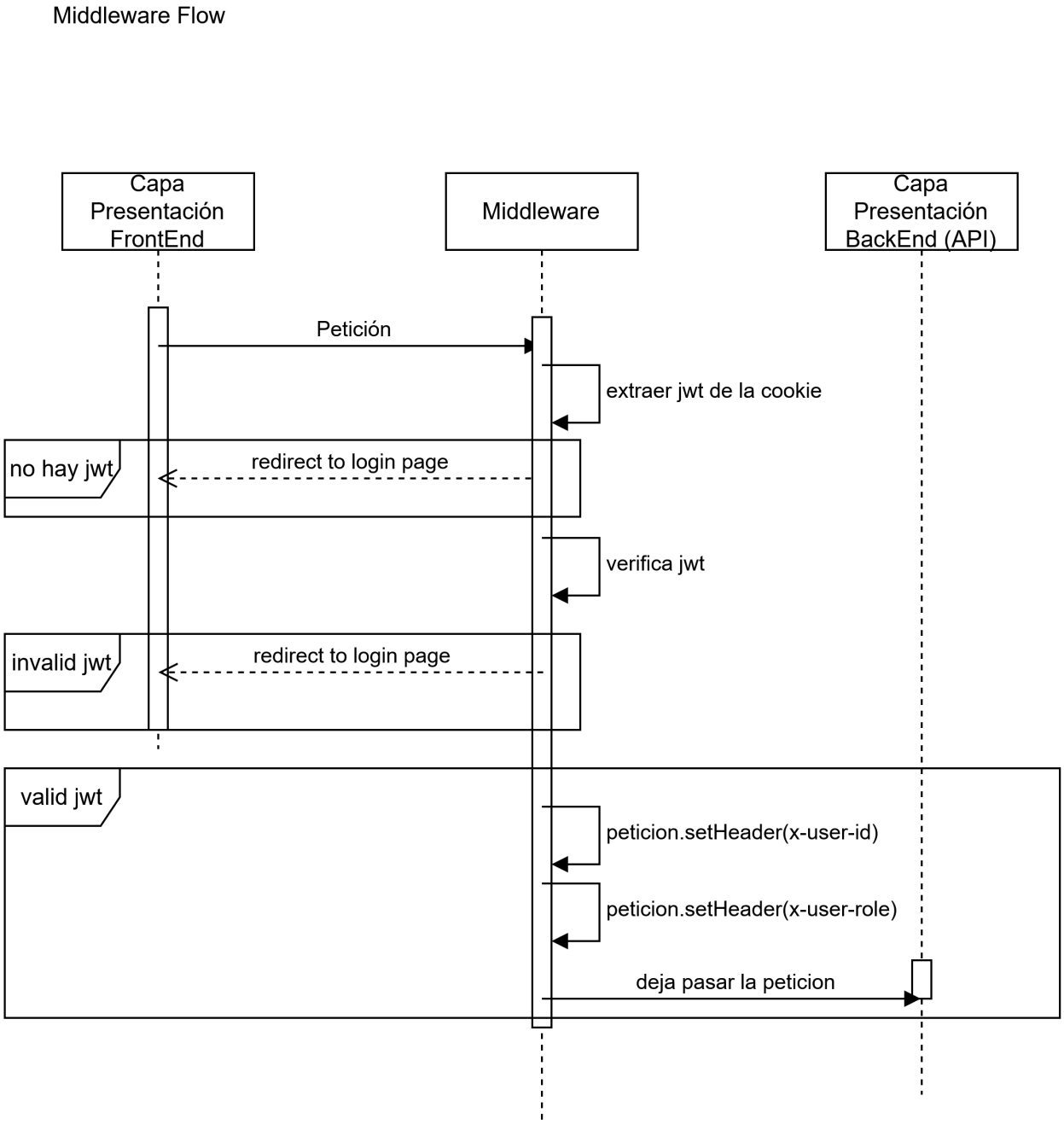

Fig. 25: Middleware

Nombre	**Descripción**	**Prioridad**
Registro de usuario	Un nuevo usuario podrá registrarse proporcionando su correo electrónico y una contraseña. Tras el registro, recibirá un email de bienvenida.	Alta
Inicio de sesión	Los usuarios podrán autenticarse con su correo y contraseña para acceder a la plataforma.	Alta
Recuperación de contraseña	En caso de olvidar la contraseña, el usuario podrá solicitar un enlace de recuperación que se enviará a su correo.	Alta
Gestión del perfil	Los usuarios podrán ver y editar sus datos personales (nombre, dirección, email, etc.), así como visualizar su historial de compras y eventos a los que ha asistido.	Media
Visualización de productos	Los clientes podrán ver un catálogo de productos con imágenes, descripciones y precios.	Alta
Búsqueda y filtros	Se podrán aplicar filtros por categoría, precio y disponibilidad.	Media
Añadir al carrito	Los clientes podrán seleccionar productos y agregarlos a su carrito de compras.	Alta
Eliminación del carrito	Los clientes podrán eliminar productos del carrito antes de realizar la compra.	Alta
Reserva de productos	Para productos en preventa o próximos lanzamientos, los clientes podrán realizar una reserva.	Media
Finalización de compra	El cliente podrá realizar la compra, esta se guardará en el historial y el sistema deberá actualizar el stock correctamente.	Media
Cancelación de pedido	El cliente podrá cancelar un pedido, el sistema deberá actualizar el stock correctamente y marcar el pedido como cancelado.	Alta
Visualización de eventos	Los clientes podrán ver los eventos próximos en un calendario.	Alta
Inscripción a torneos	Los usuarios podrán registrarse en un torneo especificando las cartas que contiene su mazo. Se enviará un email de confirmación.	Alta
Cancelación de inscripción	Los jugadores podrán darse de baja de un torneo si no pueden asistir.	Alta
Creación, edición y eliminación de productos	Los administradores podrán gestionar el inventario de la tienda manipulando datos como el nombre, imagen, descripción, precio del producto y cantidad en stock. También se puede especificar si es un producto de preventa o si tiene descuentos.	Alta
Creación y configuración de eventos	Los administradores podrán crear eventos especificando datos como la fecha, aforo, reglas y premios para los ganadores.	Alta
Consulta de inscripciones	Los administradores podrán ver una lista de clientes inscritos a un evento.	Baja
Gestión de pedidos	Los administradores podrán ver una lista de pedidos y gestionar su estado o cancelarlas.	Media
Visualización de métricas	Los administradores podrán ver métricas comerciales como evolución de ingresos, usuarios activos, pedidos pendientes, inscripciones a eventos, etc.	Media
Gestión de usuarios	Los adminstradores podrán ver una lista de los usuarios registrados en la aplicación y podrán dar o quitar el rol de administrador.	Alta
Historial de pedidos	Los clientes podrán consultar un historial de sus pedidos y filtrar por estado.	Alta
Historial de eventos	Los clientes podrán consultar un historial de eventos a los que hayan participado o se hayan inscrito.	Alta

TAULA 1: REQUISITOS FUNCIONALES

Nombre	**Descripción**	**Prioridad**
Interfaz intuitiva	La aplicación debe ser fácil de usar para clientes y administradores, con un diseño claro y accesible.	Alta
Autenticación y autorización	El sistema debe implementar autenticación segura con JWT.	Alta
Protección de datos	La información de los usuarios deberá estar protegida mediante técnicas de encriptación y buenas prácticas de seguridad.	Alta
Código modular	La estructura del proyecto deberá ser limpia y sus partes deberán ser modulares para facilitar el mantenimiento y extensibilidad de la aplicación	Alta

TAULA 2: REQUISITOS NO FUNCIONALES