FIGHTCODE

DESAROLLADORES

Laurar Felipe Rubio
Samuel Mata Arias

PROJECTO DE DESARROLLO

CFGS

CFGS Desenvolupament d'Aplicacions Web

www.fightcode.com

ENTRENAMIENTO

Entrena como un guerrero, piensa como un programador, tu cuerpo pelea, tu mente programa.
Código de combate, disciplina total

Resumen del proyecto

FightCode es una plataforma web desarrollada para un gimnasio innovador que ofrece una amplia variedad de disciplinas deportivas, combinando artes marciales como boxeo, karate y taekwondo con actividades de bienestar propio como pilates, yoga y natación en piscina climatizada, además cuenta con una pista de baloncesto y una tienda exclusiva de productos deportivos de alta calidad para usuarios registrados.

El objetivo principal ha sido diseñar una aplicación web atractiva, funcional y accesible que permita a los usuarios consultar actividades y horarios, realizar reservas en tiempo real, gestionar su suscripción, recargar un saldo virtual y realizar compras online. La plataforma incluye además un panel de administración para gestionar usuarios, productos, stock y actividades.

El proyecto ha sido desarrollado por un equipo de dos personas a lo largo de ocho meses, utilizando Vue 3 y Vite en el frontend, Next.js en el backend y Supabase como base de datos. La organización del trabajo se ha basado en la metodología Scrum con un tablero Kanban y control de versiones mediante GitHub. El resultado es una aplicación completa, desplegada en Vercel, que digitaliza de forma integral la gestión de un gimnasio moderno.

Abstract

FightCode is a web platform developed for an innovative gym that offers a wide variety of sports disciplines, combining martial arts such as boxing, karate and taekwondo with wellness activities such as pilates, yoga and swimming in a heated pool, as well as a basketball court and an exclusive high-quality sports products store for registered users.

The main objective has been to design an attractive, functional and accessible web application that allows users to browse activities and schedules, make real-time bookings, manage their subscription, top up a virtual balance and shop online. The platform also includes an administration panel to manage users, products, stock and activities.

The project was developed by a two-person team over eight months, using Vue 3 and Vite on the frontend, Next.js on the backend and Supabase as the database. The work was organised following the Scrum methodology with a Kanban board and version control through GitHub. The result is a complete application, deployed on Vercel, that comprehensively digitalises the management of a modern gym.

PALABRAS CLAVE - KEYWORDS

Keywords	**Palabras claves**
Web application	Aplicación web
gym	gimnasio
online booking	reservas online
online store	tienda online
user management	gestión de usuarios
Supabase	Supabase

ÍNDICE

ÍNDICE **4**
1. Presentación del proyecto **6**
1.1 Introducción 6
1.2 Contexto 7
1.3 Justificación 7
1.4 Objetivos 8
1.4.1 Objetivo general 8
1.4.2 Objetivos específicos 8
2. Estrategia y planificación **9**
2.1 Estrategia de desarrollo y viabilidad 9
2.2 Metodología de trabajo 10
2.3 Planificación 11
3. Analisis **13**
3.1 Casos de uso 13
3.2. Especificación de requisitos 14
3.2.1. Funcionales 14
3.2.1.1 Usuario 14
3.2.1.2 Admin 18
3.2.2 No funcionales 19
3.3 Análisis de alternativas tecnológicas 20
3.3.1 Tecnologías consideradas 20
3.3.1.1 Backend 20
3.3.1.2 Base de datos 20
3.3.1.3 Frontend 21
3.3.1.4 Despliegue 21
3.3.2 Tecnologías finalmente seleccionadas 22
3.3.2.1 Backend 22
3.3.2.1.1 Lenguaje de programación 22
3.3.2.1.2 Framework 22
3.3.2.1.2 Base de Datos 23
3.3.2.2 Frontend 23
3.3.2.2.1 Framework 23
3.3.2.2.2 Herramienta de construcción 23
3.3.2.2.3 UI Framework 24
3.3.2.2.4 Iconografía 24
3.3.2.2.5 Despliegue 24
4.Diseño **25**
4.1 Arquitectura 25
4.1.1 Descripción general 25
4.1.2 Diagrama entidad-relación 26
4.2 Arquitectura 28
4.3 Diseño Interfaz 31

5. Desarrollo **32**
5.1 Estructura del proyecto 32
5.2 Implementación de funcionalidades 34
5.3 Pruebas 36
6. Conclusiones **37**
6.1 Conclusiones generales 38
6.2 Consecución de objetivos 38
6.3 Valoración de la metodología y planificación 39
6.4 Visión de futuro 39

1. Presentación del proyecto

1.1 Introducción

En la actualidad, el sector del fitness y el bienestar ha experimentado una transformación profunda, evolucionando desde los gimnasios convencionales hacia centros de alto rendimiento y experiencias multidisciplinares. En este contexto nace FightCode, un gimnasio innovador que rompe con la oferta tradicional combinando disciplinas de contacto como el karate, boxeo y taekwondo con actividades de bienestar, como pilates, yoga, piscina climatizada para nadar, entre otras actividades. Además cuenta con espacios para deportes colectivos como el baloncesto.

FightCode no solo se limita a la actividad física, busca ofrecer una experiencia integral. Por ello, el centro cuenta con una tienda exclusiva para socios situada en la entrada, donde los socios pueden adquirir complementos y ropa deportiva de alta calidad.

Sin embargo, una oferta tan variada y ambiciosa requiere de una infraestructura digital a la altura. El propósito principal de este proyecto es el desarrollo de una plataforma web atractiva, funcional y accesible que actúe como eje central del gimnasio. A través de esta aplicación, los usuarios podrán obtener información completa y detallada de las actividades, consultar horarios y realizar reservas en tiempo real y acceder al apartado de la tienda online donde podrán comprar nuestros productos de forma remota.

Aparte de todo lo anterior, en su perfil, podrán ver el registro de sus compras online y de las actividades a las que se han apuntado, gestionar su suscripción, añadir dinero desde tarjeta a su saldo virtual y cambiar la foto de perfil a su gusto.

Como valor añadido y para mejorar la experiencia del usuario, la plataforma incluye un área personal avanzada. En su perfil, los socios podrán visualizar el registro de sus compras y actividades, gestionar su suscripción de forma autónoma, recargar un saldo virtual mediante tarjeta bancaria para facilitar sus pagos y personalizar su perfil cambiando la foto a su gusto personal.

1.2 Contexto

La idea surge en clase de proyecto. Mientras intentábamos formalizar una lluvia de ideas, nos levantamos frente a la ventana para tomar un respiro de aire puro, observamos el gimnasio situado al frente del aula. Ese instante, aparentemente cotidiano, se convirtió en el detonante que dio forma a nuestra propuesta.

¿Seremos capaces de crear una aplicación web para un gimnasio completamente variado en disciplinas y actividades, donde practicar deporte resulte atractivo para cualquier tipo de persona y no sea necesario estar apuntado a varios centros para disfrutar de diversas actividades?

1.3 Justificación

Fightcode está pensado para cubrir la creciente necesidad de digitalizar la gestión del ocio fitness. Se trata de una plataforma integral que facilita reservas, horarios, compras online y servicios complementarios, ofreciendo a los clientes una herramienta sencilla e intuitiva para acceder a sus actividades y productos favoritos. Gracias a esta propuesta, FightCode no solo compite con centros similares de la zona, sino que se diferencia con una oferta única ya que cuenta con sala de boxeo, taekwondo, piscina climatizada, una pista de baloncesto y tienda propia entre otras actividades atractivas para el usuario. Todo ello pensado para adaptarse a las verdaderas demandas del sector fitness y brindar una experiencia completa, moderna y accesible.

1.4 Objetivos

1.4.1 Objetivo general

Desarrollar una aplicación web para la gestión de servicios de un gimnasio, que permita a los usuarios reservar actividades, consultar horarios, realizar compras online y acceder a servicios complementarios de manera sencilla, eficiente y atractiva.

1.4.2 Objetivos específicos

- Facilitar la gestión de horarios y reservas mediante una interfaz intuitiva y accesible para todos los usuarios.
- Integrar compras online de productos deportivos y suplementos dentro de la misma plataforma, garantizando comodidad y seguridad.
- Ofrecer gran variedad de actividades.
- Optimizar la experiencia del usuario con un diseño moderno, responsive y fácil de usar desde cualquier dispositivo.
- Diferenciarse de la competencia mediante una propuesta integral que combina servicios deportivos, de ocio y comerciales.
- Promover la digitalización del sector fitness, posicionando FightCode como un referente innovador y competitivo.
- Garantizar accesibilidad y usabilidad, adaptando la aplicación a distintos perfiles de usuarios y necesidades.
- Fomentar la fidelización de clientes gracias a una experiencia completa que combina deporte, bienestar y consumo responsable.

2. Estrategia y planificación

2.1 Estrategia de desarrollo y viabilidad

El desarrollo del proyecto se ha planteado como un proyecto fullstack dividido en dos capas diferenciadas. Para el frontend se ha utilizado Vue 3 junto con Vite, tecnologías que hemos ido trabajando durante el curso siendo así más cómodo para el desarrollo. Para el backend se dudó un poco más pero se optó por Next.js, ya que trabajamos utilizado este framework previamente en las prácticas de empresa, lo que facilitó su adopción y redujo la curva de aprendizaje. Como base de datos, inicialmente se trabajó con PostgreSQL gestionado a través de pgAdmin, pero ante la necesidad de agilizar el desarrollo y cumplir con los plazos de entrega, se tomó la decisión de migrar a Supabase. Esta plataforma, también basada en PostgreSQL, ofrece autenticación integrada, almacenamiento y acceso en tiempo real, lo que permitió avanzar más rápido sin sacrificar las funcionalidades necesarias para la gestión de usuarios, reservas y compras.

En cuanto a la viabilidad, el proyecto ha sido desarrollado por un equipo de dos alumnos a lo largo de aproximadamente ocho meses, desde septiembre hasta mayo. Este margen de tiempo ha permitido abordar y completar todas las funcionalidades previstas, el sistema de reservas y horarios, compras en la tienda online, el área de perfil de usuario con saldo virtual, gestión de suscripciones y un panel admin donde gestionar todas las consultas de la base de datos, registros, editar stock y precios de los productos de la tienda, y poder gestionar las actividades; poder suspenderlas temporalmente y reanudarlas a su gusto, toda esta parte exclusivamente para los usuarios admin. Las tecnologías escogidas son todas de código abierto y con licencias de uso libre, lo que garantiza la viabilidad económica del proyecto. El propio proyecto se distribuye bajo una licencia **Creative Commons Reconeixement-NoComercial-SinObraDerivada 3.0 España (CC BY-NC-ND)**, lo que permite su consulta y uso con fines educativos, restringiendo su uso comercial y la generación de obras derivadas. El hecho de partir de tecnologías ya conocidas por el equipo fue clave para que el desarrollo fuera viable dentro del tiempo disponible.

2.2 Metodología de trabajo

Para la organización del proyecto se ha seguido una metodología basada en **Scrum**. Como herramienta principal de seguimiento se utilizó un tablero **Kanban**, que permitía visualizar en todo momento el estado de las tareas; pendientes, en progreso, en revisión, con problemas y completadas. Esto facilitó repartir el trabajo de forma clara y detectar rápidamente cualquier bloqueo o dependencia entre tareas.

El desarrollo se llevó a cabo de forma iterativa, siguiendo un orden lógico de construcción. Se comenzó por el diseño de la interfaz en Figma en formato desktop, lo que permitió tener una visión clara del proyecto antes de empezar a programar. A partir de ahí se construyó el frontend con Vue 3 y Vite, posteriormente se definió y completó la base de datos en Supabase, y con esa base se fue desarrollando el backend con Next.js en paralelo a los ajustes del frontend. La coordinación entre los dos integrantes fue continua, aprovechando especialmente las horas de clase para trabajar juntos, resolver dudas y tomar decisiones en común. A medida que el proyecto avanzaba, el ritmo de trabajo se fue intensificando, dedicando horas adicionales fuera del horario lectivo durante el último mes para garantizar la entrega de todas las funcionalidades previstas. Antes del despliegue se realizaron pruebas unitarias para verificar el correcto funcionamiento de las principales funcionalidades y detectar posibles errores. Finalmente, el proyecto fue desplegado en Vercel, lo que requirió una fase adicional de adaptación del código para garantizar su correcto funcionamiento en producción.

Para el control de versiones y la gestión del código compartido se utilizó GitHub, siguiendo un flujo de trabajo basado en ramas, cada integrante trabajando en su propia rama y, una vez completada una funcionalidad, se realizaba un pull request para revisarlo y hacer el merge a la rama principal. Esto permitió trabajar en paralelo de forma ordenada, evitar conflictos y mantener un historial claro de todos los commits realizados a lo largo del proyecto.

2.3 Planificación

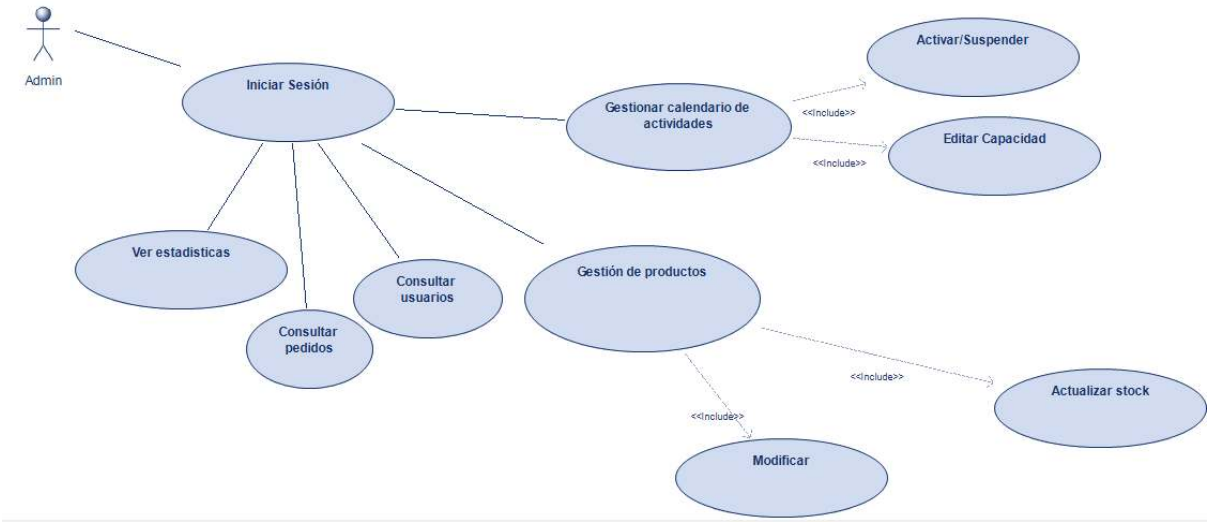
Fase	Período	Contenido
Fase 1 - Inicio y documentación	Septiembre/Octubre	Definición del proyecto(propuesta inicial), portada, licencias, resumen, abstract y palabras clave. Aparte se creó el repositorio, el kanban y los primeros bocetos de diagramas.
Fase 2 - Investigación	Octubre/Noviembre	Se investigó sobre las tecnologías a implementar, haciendo comparativas entre ellas
Fase 3 - Figma	Noviembre/Diciembre/Enero	Diseño de la interfaz en formato desktop mediante Figma. Durante esta fase también se redactó la comparativa de tecnologías, se definieron las funcionalidades principales de la aplicación y se elaboró la planificación inicial del proyecto. El diseño se fue ajustando progresivamente a medida que avanzaba el desarrollo.
Fase 4 - Frontend	Febrero/Abril	Período en el que se empezó a programar seriamente. Se inició con las vistas principales y los componentes reutilizables, y se fue complementando y refinando a lo largo del desarrollo, aplicando los últimos retoques de estilos y funcionalidades hasta el cierre del proyecto en mayo.
Fase 5 - Base de datos	Febrero/Marzo	Se empezó a diseñar la base de datos con los primeros diagramas en papel. Inicialmente se trabajó con

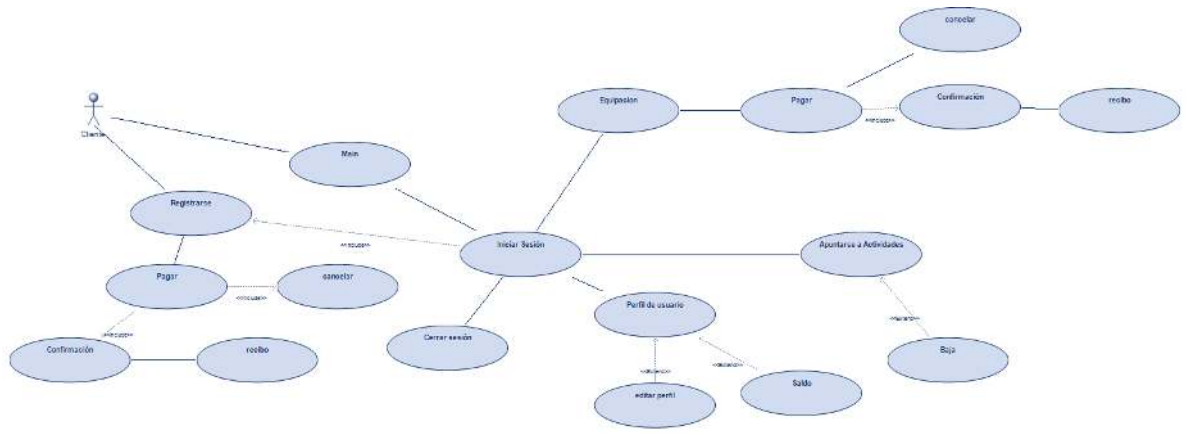
		PostgreSQL gestionado mediante pgAdmin, pero ante las dificultades de configuración en producción y los plazos de entrega se tomó la decisión de migrar a Supabase, agilizando así el desarrollo.
Fase 6- Backend	Marzo/Mayo	Con la base de datos completada, se instalaron las dependencias necesarias y se comenzó el desarrollo del backend con Next.js, implementando los endpoints REST para cada recurso de la aplicación en paralelo a los ajustes del frontend.
Fase 7- Pruebas unitarias	Mayo	Se realizaron pruebas unitarias con Jest en el backend y Vitest en el frontend, cubriendo los principales endpoints y el store de autenticación, verificando el correcto funcionamiento antes del despliegue.
Fase 8- Deploy	Mayo	Se desplegó la aplicación en Vercel, tanto el frontend como el backend, adaptando el código y la configuración para garantizar el correcto funcionamiento en producción.
Fase 9- Completar memoria + Entregar/Exponer proyecto	Mayo	Redacción y revisión final de la memoria del proyecto, incorporando todas las partes y correcciones indicadas por el profesorado. Una vez completada, se realizará la entrega y la exposición oficial del proyecto.

3. Analisis

3.1 Casos de uso

A continuación se presentan los distintos diagramas elaborados para representar el funcionamiento del sistema desde diferentes perspectivas. Estos esquemas permiten visualizar de forma clara las acciones principales que pueden realizar tanto los usuarios como los administradores.





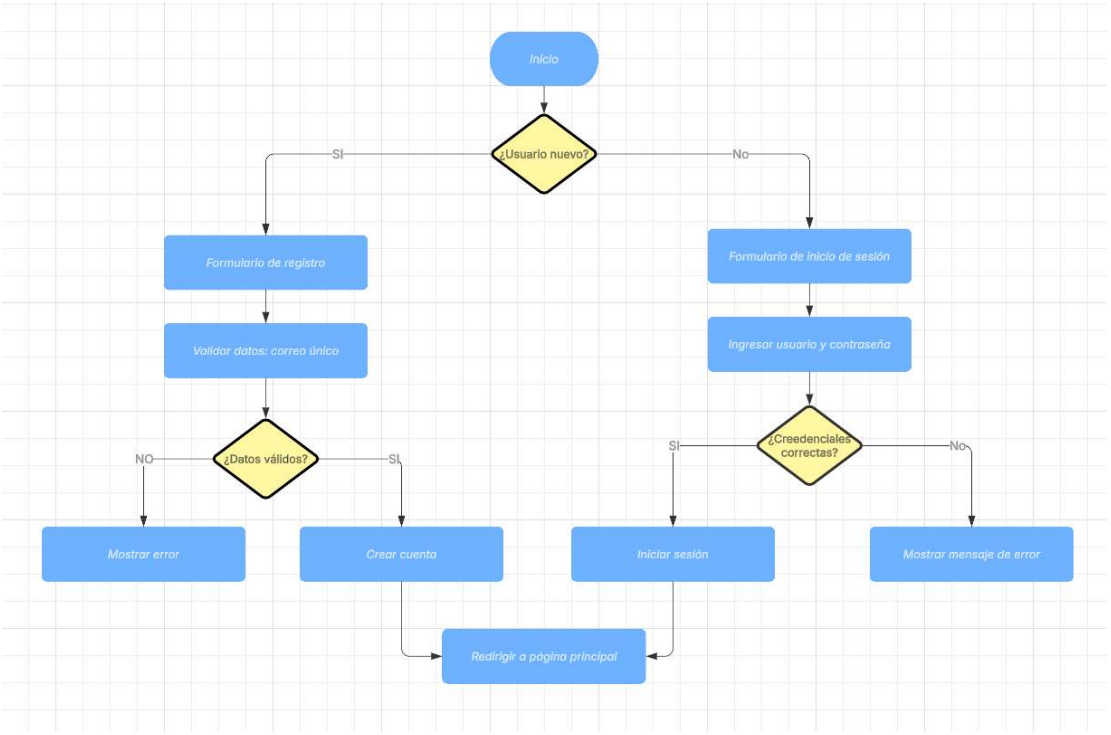
3.2. Especificación de requisitos

3.2.1. Funcionales

3.2.1.1 Usuario

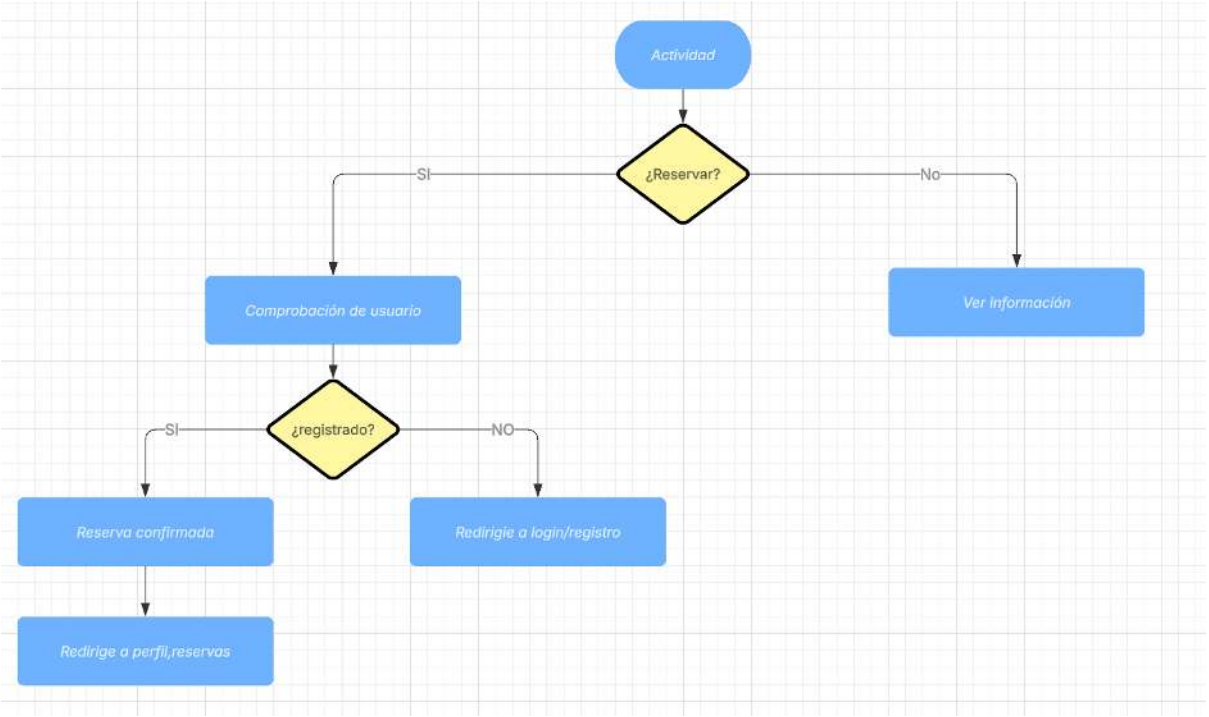
Gestión de usuarios

El sistema debe permitir el registro de usuarios proporcionando ciertos datos: nombre, apellido, correo electrónico y contraseña. La contraseña deberá cumplir con requisitos mínimos de seguridad. Estos criterios pueden variar, pero inicialmente se plantea un mínimo de 8 caracteres. En el futuro, podrían exigirse condiciones más estrictas, como incluir al menos una letra mayúscula, una minúscula, un número, un carácter especial y alcanzar un mínimo de 10 caracteres.
Durante el proceso de registro, se validarán los datos introducidos para evitar duplicados. No se permitirá la creación de cuentas con un correo electrónico ya existentes en el sistema.
Para iniciar sesión, se podrá utilizar el nombre de usuario, junto con la contraseña correspondiente. Si las credenciales son correctas, el sistema redirigirá a la página principal con la sesión iniciada.
En caso de error, se mostrará un mensaje simple indicando que las credenciales son incorrectas.
La gestión de sesiones se realizará mediante el uso de tokens para mantener la seguridad.



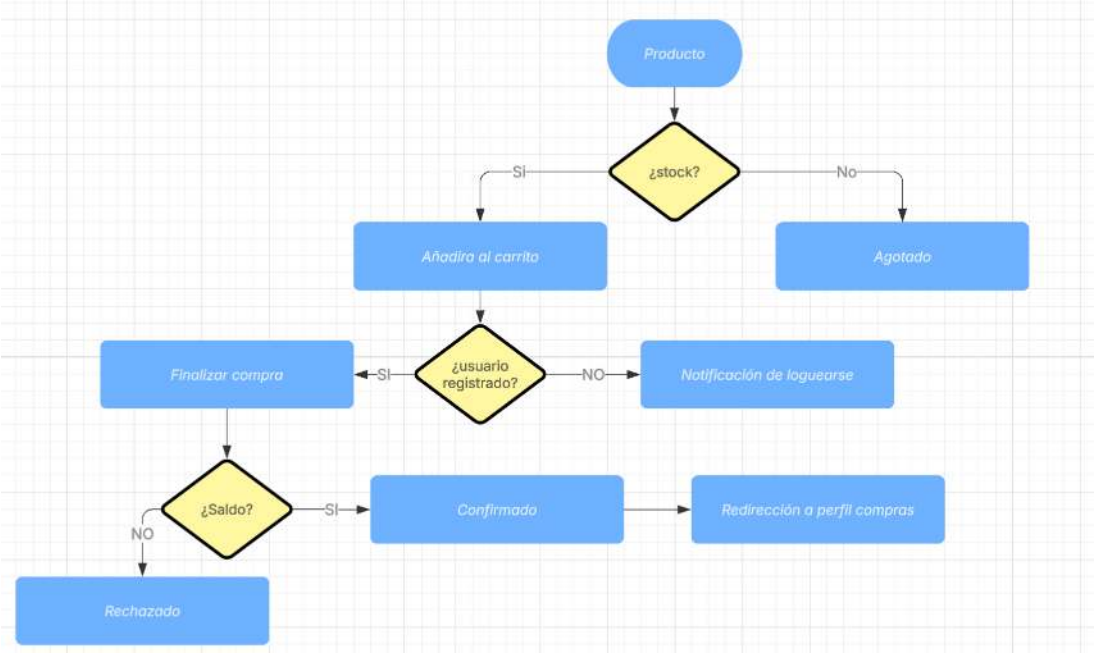
Gestión de reservas

En el sitio se mostrará un listado general de actividades, accesible desde una sección específica. Este espacio permitirá a los usuarios visualizar el conjunto de actividades disponibles de forma clara y ordenada.
Cada actividad cuenta con una página informativa individual, en la que se incluyen detalles como el título, la descripción, el horario, y la lista de entrenadores.
Además, se indicará si existe anulación de la actividad una vez se intente reservar. Desde esta misma página, será posible iniciar el proceso de reserva de la clase, lo que permitirá una transición directa entre la consulta de la información y la acción por parte del usuario.



Gestion de compra

El sistema debe permitir a los usuarios acceder a un catálogo general de productos disponibles en la tienda del gimnasio. Este catálogo incluirá artículos como ropa deportiva, accesorios de entrenamiento, suplementos y material de uso personal. Cada producto contará con detalles como el nombre, la descripción, el precio y el estado de stock actualizado.
Para iniciar el proceso de compra, el usuario podrá añadir uno o varios productos al carrito. Antes de confirmar la operación, el sistema verificará la disponibilidad real del inventario para evitar compras de artículos agotados. En caso de que un producto no esté disponible, se mostrará un mensaje informativo indicando la imposibilidad de completar la compra de ese artículo.
El sistema de compra implementará un proceso de pago simplificado, orientado a validar únicamente si el usuario dispone del saldo necesario para completar la transacción. Una vez que el usuario haya añadido los productos al carrito y decida proceder al pago, el sistema calculará el importe total de la compra y lo comparará con el saldo disponible en su cuenta.
Si el saldo es suficiente, la operación se considerará válida y la compra se completará de forma automática. En ese momento, el sistema descontará el importe correspondiente, registrará la transacción y mostrará un mensaje de confirmación indicando que la compra se ha realizado correctamente y se redirigirá al cliente al apartado en perfil donde se podrá ver la lista de compras.
En caso de que el usuario no disponga del saldo necesario, el sistema no permitirá finalizar la operación. Se mostrará un mensaje claro informando de que no es posible completar la compra debido a la falta de fondos. No se realizará ningún cargo ni se reservarán productos en esta situación.

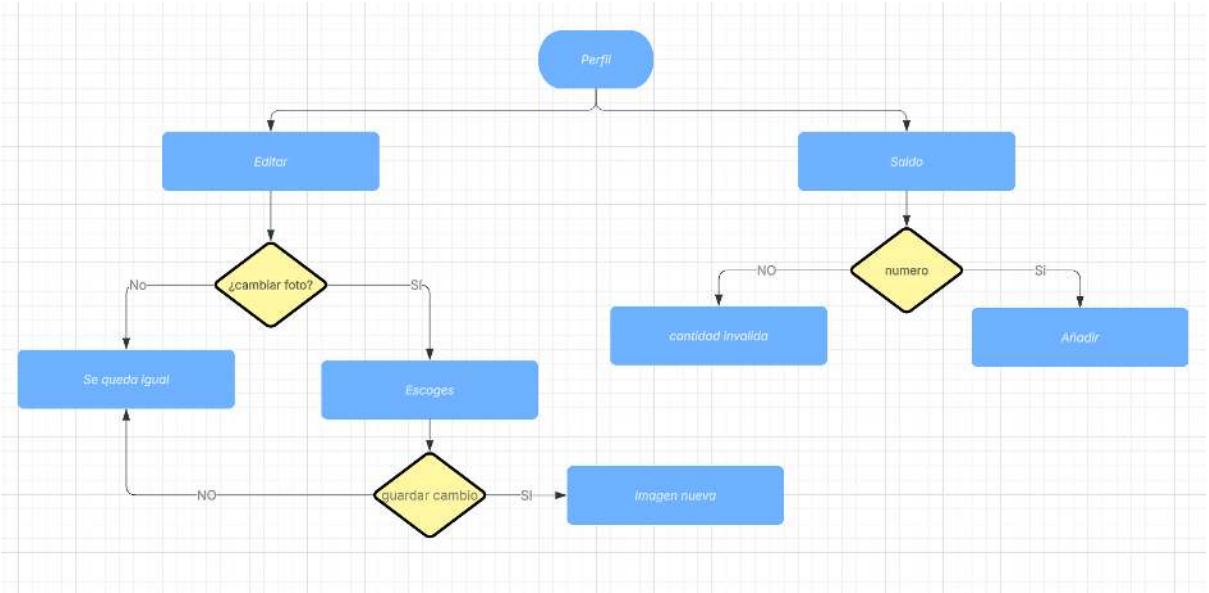


Gestión perfil

Dentro del área de configuración del usuario se ofrece la posibilidad de personalizar la imagen de perfil. Para ello, la plataforma pone a disposición una serie de imágenes predeterminadas que pueden seleccionarse directamente, evitando la necesidad de subir archivos externos. El usuario puede recorrer la galería disponible y elegir la imagen que mejor lo represente. Una vez confirmada la selección, la nueva foto se actualiza de forma inmediata en todas las secciones donde se muestre la identidad del usuario. Este proceso es directo y no requiere validaciones adicionales, ya que todas las imágenes han sido previamente aprobadas y optimizadas para su uso dentro del sistema.

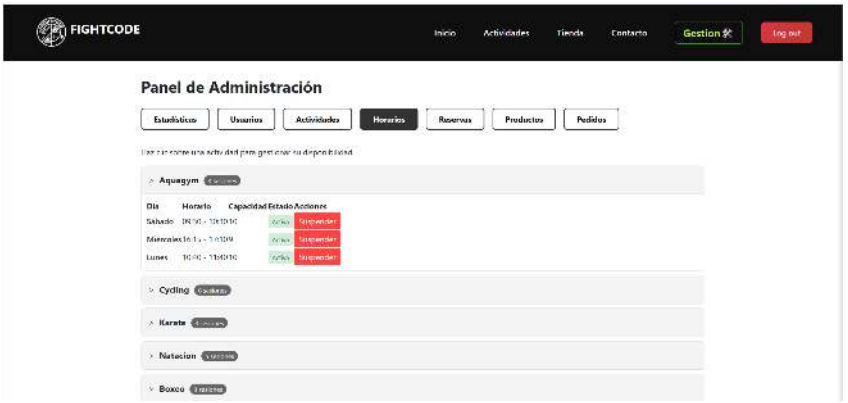
Gestión recarga de saldo

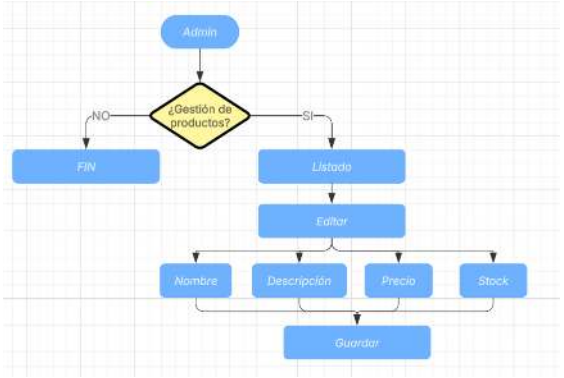
La recarga de saldo se realiza mediante un procedimiento simplificado que busca agilizar la gestión económica del usuario. Para añadir fondos, únicamente es necesario introducir la cantidad deseada en un campo numérico habilitado para este fin.
Tras confirmar la operación, el sistema verifica que el valor introducido sea válido y actualiza el saldo disponible en la cuenta del usuario. No se solicitan métodos de pago adicionales ni información complementaria, lo que permite completar la recarga de forma rápida y sin complicaciones.
Una vez finalizado el proceso, se muestra un mensaje de confirmación indicando que el saldo ha sido actualizado correctamente.

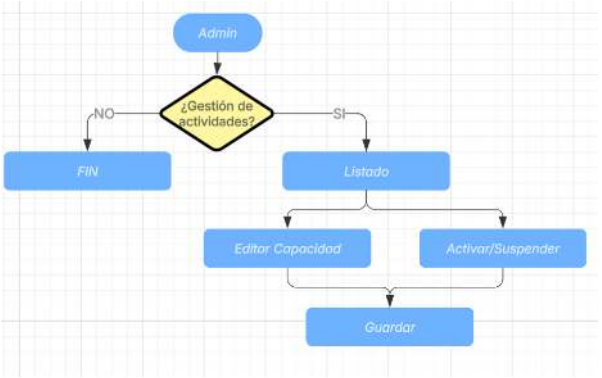


3.2.1.2 Admin

El área de administración ofrece un conjunto ampliado de herramientas destinadas a gestionar los distintos elementos de la plataforma. Desde este panel, el administrador puede supervisar y modificar la información relacionada con las actividades, los productos y los usuarios, garantizando el correcto funcionamiento del entorno.
En lo referente a las actividades, el administrador tiene la posibilidad de ajustar la capacidad máxima de cada una de ellas, así como activar o suspender horarios específicos cuando sea necesario. Estas acciones permiten mantener una organización flexible y adaptada a la demanda real del gimnasio.
En la sección de productos, el administrador puede modificar aspectos esenciales como el nombre, la descripción, el precio y el stock disponible. Esta capacidad de edición facilita mantener el catálogo actualizado y coherente con la disponibilidad del inventario.
Además, el panel administrativo incluye acceso a listados completos de usuarios registrados y de pedidos realizados. Esta información permite llevar un control general del uso de la plataforma y del movimiento de compras.
También se ofrece un apartado de estadísticas básicas, donde se muestran datos relevantes sobre actividad, ventas o participación, proporcionando una visión global del rendimiento del sistema.







3.2.2 No funcionales

Rendimiento	La plataforma debe ofrecer tiempos de respuesta fluidos en las operaciones más frecuentes, como la consulta de actividades, la visualización de productos o el acceso al perfil. Las acciones básicas deben completarse en un intervalo breve para garantizar una experiencia de uso cómoda.
Usabilidad	La interfaz debe ser intuitiva y fácil de navegar tanto para usuarios como para administradores. Los elementos visuales deben estar organizados de forma clara, permitiendo localizar rápidamente las funciones principales sin necesidad de conocimientos técnicos.
Disponibilidad	El sistema debe estar accesible de forma continua, salvo en momentos puntuales de mantenimiento. La plataforma debe permitir que los usuarios puedan consultar actividades, realizar compras o gestionar su perfil en cualquier momento.
Seguridad	La información sensible del usuario debe almacenarse de forma segura. En particular, las contraseñas no se guardarán en texto plano, sino que se almacenarán utilizando un algoritmo de hash para evitar su lectura directa.
Compatibilidad	La plataforma debe funcionar correctamente en los navegadores más utilizados y adaptarse a distintos tamaños de pantalla. El diseño debe mantenerse estable tanto en ordenadores como en dispositivos móviles.

3.3 Análisis de alternativas tecnológicas

3.3.1 Tecnologías consideradas

Durante la fase inicial del proyecto se valoraron varias tecnologías para el desarrollo tanto del backend como del frontend. Aunque todas ellas eran válidas, finalmente se descartaron por motivos de tiempo, complejidad o integración.

3.3.1.1 Backend

Next.js + TypeScript

Se planteó utilizar Next.js como framework completo (frontend + backend) junto con TypeScript para mejorar el tipado y la robustez del código.

Sin embargo, esta opción se descartó porque:

- Requería una estructura más compleja de la necesaria.
- El uso de TypeScript aumentaba el tiempo de desarrollo.
- La curva de aprendizaje era mayor para el equipo.
- El proyecto necesitaba avanzar rápido y sin sobrecarga técnica.

3.3.1.2 Base de datos

PostgreSQL autogestionado + pgAdmin

También se consideró montar una base de datos PostgreSQL manualmente y gestionarla con pgAdmin.

Se descartó porque:

- Requería configurar servidores, puertos y conexiones.
- El despliegue era más lento y menos práctico.
- No ofrecía autenticación integrada ni API automática.

3.3.1.3 Frontend

Se valoró usar Next.js también para el frontend, aprovechando su sistema de páginas y SSR.

Finalmente no se eligió porque:

- Vue ofrecía una curva de aprendizaje más rápida.
- El equipo ya tenía experiencia previa con Vue.
- Next.js estaba sobredimensionado para una interfaz sencilla.

3.3.1.4 Despliegue

Netlify se consideró como plataforma de despliegue, pero se descartó porque:

- No se integra tan bien con API Routes de Next.js.
- Vercel ofrece despliegue automático y nativo para Next.js.
- El flujo GitHub → Vercel era más rápido y directo.

3.3.2 Tecnologías finalmente seleccionadas

Tras evaluar las opciones anteriores, se optó por una combinación más ligera, rápida y fácil de integrar, que permitiera avanzar en el proyecto sin complicaciones.

3.3.2.1 Backend

3.3.2.1.1 Lenguaje de programación

Para el desarrollo del backend se ha seleccionado **JavaScript**, debido a su sencillez, su amplia adopción en el entorno web y su integración directa con Next.js. JavaScript permite implementar la lógica del servidor de forma ágil y sin necesidad de configuraciones adicionales. Aunque inicialmente se valoró el uso de TypeScript, finalmente se optó por JavaScript puro para agilizar el desarrollo y reducir la complejidad del proyecto, manteniendo un flujo de trabajo más rápido y flexible.

3.3.2.1.2 Framework

El backend se ha implementado utilizando Next.js, empleando exclusivamente su sistema de API Routes. Esta funcionalidad permite definir endpoints dentro del propio proyecto sin necesidad de crear un servidor independiente.

Este enfoque facilita la organización del código, simplifica la comunicación con el frontend y permite un despliegue inmediato en Vercel, evitando configuraciones manuales y problemas de CORS.

3.3.2.1.2 Base de Datos

Para la gestión de datos se ha seleccionado Supabase, una plataforma que ofrece una base de datos PostgreSQL en la nube junto con herramientas integradas para autenticación, almacenamiento y consultas.
Supabase proporciona un panel visual que facilita la administración de tablas y relaciones, permitiendo trabajar con PostgreSQL sin necesidad de configuraciones complejas.
Su integración con JavaScript y su SDK oficial permiten realizar operaciones de lectura y escritura de forma sencilla y eficiente.

3.3.2.2 Frontend

3.3.2.2.1 Framework

El frontend se ha desarrollado utilizando Vue 3, un framework progresivo que permite construir interfaces dinámicas y reactivas mediante componentes reutilizables.
Vue ofrece una curva de aprendizaje rápida, una sintaxis clara y una estructura modular que facilita el mantenimiento y la escalabilidad del proyecto.

3.3.2.2.2 Herramienta de construcción

Para el entorno de desarrollo se ha utilizado Vite, una herramienta moderna que proporciona una carga extremadamente rápida y un refresco instantáneo durante el desarrollo.
Vite permite trabajar con Vue de forma eficiente, reduciendo tiempos de espera y mejorando la productividad.

3.3.2.2.3 UI Framework

Para facilitar el diseño responsive y asegurar una experiencia de usuario fluida, se ha optado por utilizar Bootstrap 5.x como framework de interfaz.

Bootstrap ofrece una amplia colección de componentes predefinidos —botones, formularios, tarjetas, barras de navegación, etc.— que permiten acelerar el desarrollo y mantener una coherencia visual en toda la aplicación sin necesidad de diseñar estilos desde cero.

3.3.2.2.4 Iconografía

Para complementar la interfaz y mejorar la experiencia visual del usuario, se ha utilizado Bootstrap Icons, la librería oficial de iconos del framework.

Esta colección proporciona iconos vectoriales ligeros y fácilmente integrables en elementos como botones, enlaces o encabezados, contribuyendo a una interfaz más clara y accesible.

3.3.2.2.5 Despliegue

El proyecto se ha desplegado utilizando Vercel, tanto para el frontend como para las API Routes del backend.

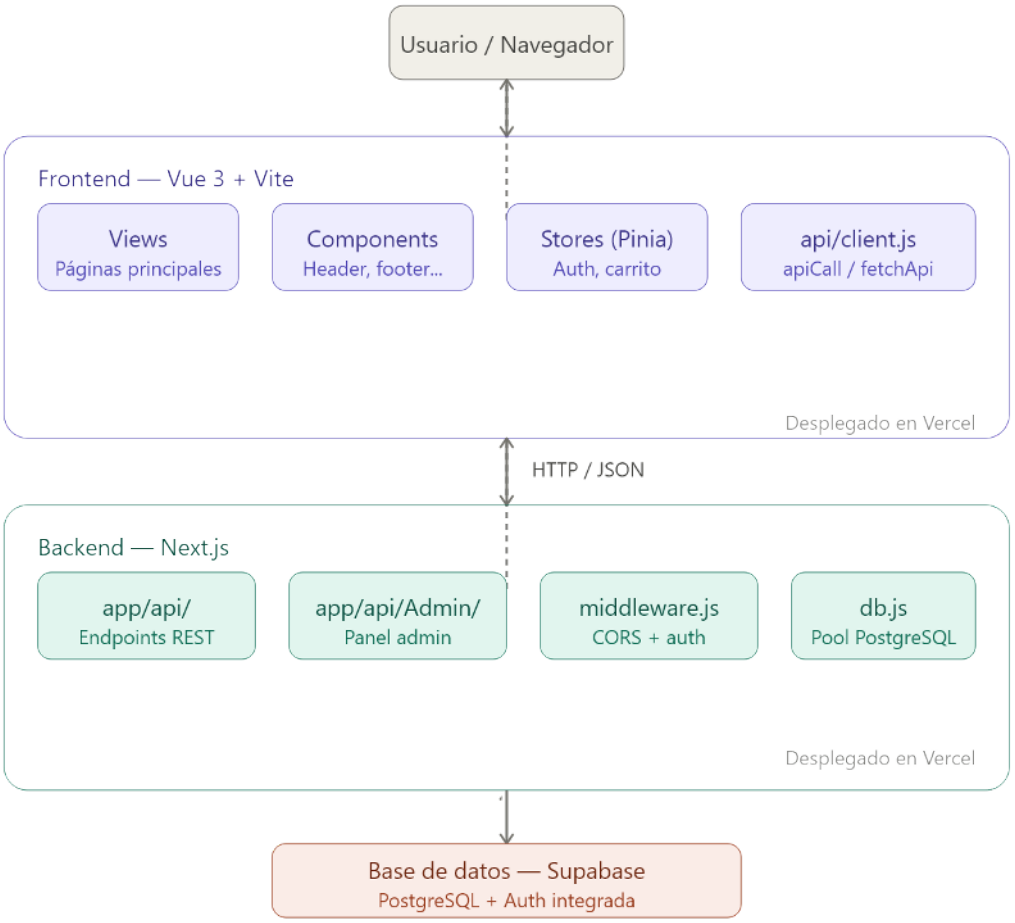
Vercel ofrece integración directa con GitHub, permitiendo que cada actualización del repositorio genere automáticamente un nuevo despliegue.

Su compatibilidad nativa con Next.js y su rapidez de build han permitido un proceso de publicación sencillo y eficiente.

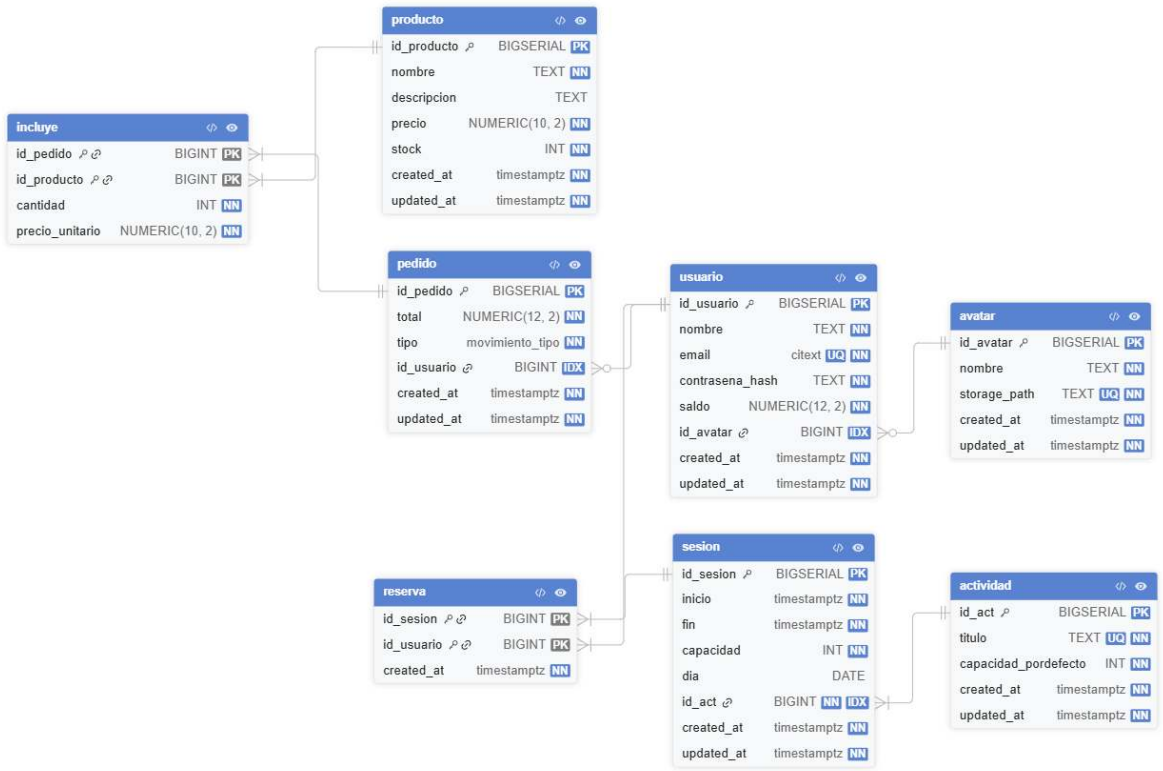
4.Diseño

4.1 Arquitectura

4.1.1 Descripción general



4.1.2 Diagrama entidad-relación



Uno a muchos (1:N)

AVATAR → USUARIO: un avatar puede ser usado por varios usuarios.

USUARIO → PEDIDO: un usuario puede tener varios pedidos.

USUARIO → RESERVA: un usuario puede tener varias reservas.

ACTIVIDAD → SESION: una actividad tiene varias sesiones.

SESION → RESERVA: una sesión puede tener varias reservas.

Muchos a muchos (M:N)

PEDIDO ↔ PRODUCTO a través de INCLUYE: un pedido puede contener varios productos y un producto puede aparecer en varios pedidos.

SESION ↔ USUARIO a través de RESERVA: un usuario puede reservar varias sesiones y una sesión puede tener varios usuarios reservados.

Esquema de base de datos

4.2 Arquitectura

Entidad: avatar

Se creó para almacenar imágenes o representaciones visuales asociadas a los usuarios.

Incluye atributos como nombre, ruta de almacenamiento y marcas de tiempo.

Cada avatar puede estar asociado a múltiples usuarios, aunque un usuario solo puede tener un avatar a la vez.

Entidad: usuario

Se desarrolló para representar a los clientes del sistema.

Incluye información como nombre, email, contraseña cifrada, saldo y un avatar opcional.

El saldo se controla mediante restricciones para evitar valores negativos.

Un usuario puede realizar pedidos y reservar sesiones.

Entidad: producto

Se definió para gestionar los artículos disponibles para la compra.

Incluye atributos como nombre, descripción, precio y stock.

Los productos pueden aparecer en múltiples pedidos a través de la tabla intermedia incluye.

Entidad: actividad

Se creó para representar las actividades generales del centro (por ejemplo, yoga, spinning, boxeo).

Cada actividad tiene un título único y una capacidad por defecto.

Las actividades sirven como base para generar sesiones concretas.

Entidad: sesion

Se diseñó para representar instancias específicas de una actividad en un día y hora concretos.

Incluye atributos como inicio, fin, capacidad y un campo generado automáticamente que almacena el día.

Cada sesión pertenece a una única actividad.

Además, se implementó un trigger para validar que la capacidad no se exceda al realizar reservas.

Entidad: pedido

Se desarrolló para registrar movimientos económicos realizados por los usuarios.

El campo tipo utiliza un ENUM llamado movimiento_tipo, que distingue entre COMPRA y RECARGA.

El total del pedido se recalcula automáticamente mediante un trigger cada vez que se modifica la tabla incluye.

Entidad: incluye

Se creó como tabla intermedia para representar los productos incluidos en un pedido.

Define una relación 1 pedido – N productos y 1 producto – N pedidos.

Incluye la cantidad y el precio unitario en el momento de la compra.

La clave primaria compuesta garantiza que un producto no se repita dentro del mismo pedido.

Entidad: reserva

Se diseñó para gestionar las inscripciones de usuarios a sesiones.

Representa una relación M:N entre usuario y sesión.

Incluye un trigger que valida que no se supere la capacidad de la sesión antes de insertar una reserva.

Si se elimina un usuario o una sesión, sus reservas asociadas se eliminan automáticamente.

Resumen

Relación	Tipo	Descripción
usuario – avatar	N:1	Un usuario puede tener un avatar; un avatar puede estar en varios usuarios
actividad – sesión	1:N	Una actividad genera múltiples sesiones
usuario – pedido	1:N	Un usuario puede realizar varios pedidos
pedido – producto (incluye)	M:N	Un pedido puede contener varios productos y viceversa
usuario – sesion (reserva)	M:N	Un usuario puede reservar varias sesiones y una sesión puede tener varios usuarios

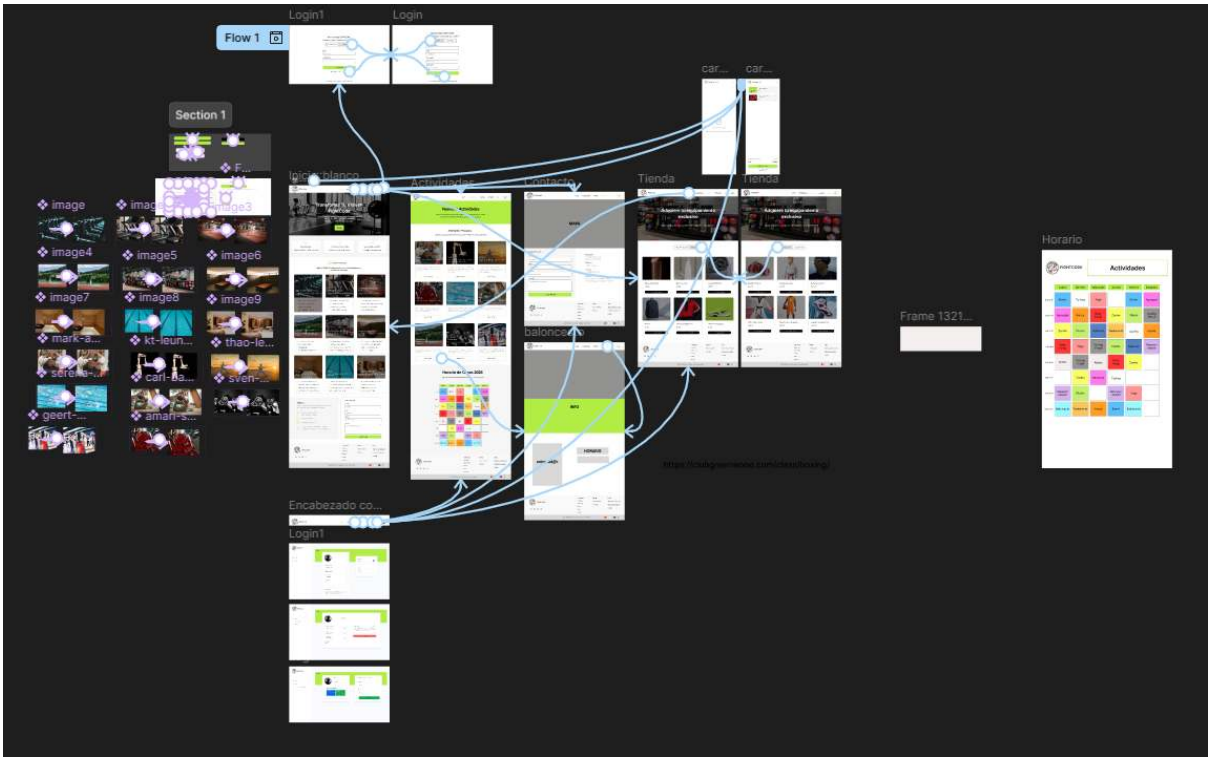
4.3 Diseño Interfaz

El diseño de la interfaz de usuario se realizó inicialmente en Figma, creando un conjunto de bocetos y prototipos que permitieron definir la estructura visual de la aplicación antes de comenzar con la implementación en código.

Figma facilitó la organización de las pantallas, la distribución de los elementos y la definición de la experiencia de uso, permitiendo visualizar cómo interactuaría el usuario con cada sección del sistema.

A través de estos prototipos se estableció la disposición de menús, tarjetas, formularios y componentes principales, asegurando una navegación clara y coherente. El uso de Figma permitió iterar rápidamente sobre el diseño, ajustando colores, tamaños y posiciones sin necesidad de modificar código, lo que aceleró el proceso creativo y evitó inconsistencias visuales.

Una vez validado el diseño, la interfaz se implementó utilizando Vue 3, Vite y Bootstrap 5, respetando la estructura definida en los prototipos. Gracias a esta planificación previa, la experiencia de usuario resultante es intuitiva, ordenada y adaptada a distintos dispositivos, manteniendo una estética uniforme en toda la aplicación.



5. Desarrollo

5.1 Estructura del proyecto

El proyecto está organizado como una aplicación fullstack con el frontend y el backend en un mismo repositorio pero en carpetas claramente separadas. La base de datos se gestiona de forma externa a través de Supabase.

Frontend (Vue 3 + Vite)
El código del frontend se encuentra bajo la carpeta src/ y está dividido en las siguientes secciones:

- **src/views/:** Contiene las páginas principales de la aplicación; actividades, login, perfil, tienda, entre otras.
- **src/components/:** Componentes reutilizables a lo largo de la aplicación como el header, footer y sidebar.
- **src/router/:** Configuración de las rutas de navegación de la aplicación.
- **src/stores/:** Estado global gestionado con Pinia, incluyendo la autenticación de usuarios y el carrito de compra.
- **src/api/:** Cliente HTTP encargado de centralizar las comunicaciones con el backend.
- **src/themes/:** Estilos CSS organizados por sección y actividad.
- **src/assets/:** Recursos estáticos como imágenes de las actividades.
- **.env.local:** Almacena variables de entorno del frontend, como la URL base del backend para las peticiones HTTP.
- **vite.config.js:** Archivo de configuración de Vite, donde se define el comportamiento del servidor de desarrollo y la compilación del proyecto.
- **vitest.config.js:** Archivo de configuración de Vitest, el framework de testing utilizado en el frontend.
- **vercel.json:** Configuración del despliegue del frontend en Vercel.

Backend (Next.js)
El backend está estructurado bajo la carpeta app/api/ siguiendo las conversiones de rutas de Next.js:

- **app/api/**: Contiene los endpoints REST organizados por recurso, cada uno en su propio archivo route.js. Cada endpoint gestiona los métodos HTTP necesarios según el recurso. Todos los endpoints utilizan las funciones de lib/cors.js para gestionar CORS y las peticiones preflight mediante OPTIONS.
- **app/api/Admin/:** Rutas exclusivas para el apartado de gestión administrativa de datos del usuario admin.
- **app/api/cron/:** Aquí se gestiona el proceso de suscripción.
- **lib/:** Contiene el archivo cors.js, que centraliza la gestión de CORS mediante tres funciones: withCORS() para respuestas exitosas, withCORSError() para errores, y handleOPTIONS() para las peticiones preflight. Esto permite la comunicación entre el frontend y el backend sin restricciones de origen cruzado.
- **middleware.js:** Intercepta todas las peticiones entrantes a las rutas /api/* antes de que lleguen a los endpoints. Añade automáticamente las cabeceras CORS a cada respuesta y gestiona las peticiones preflight OPTIONS, evitando tener que configurarlo manualmente en cada endpoint.
- **db.js:** Gestiona la conexión a la base de datos mediante un pool de conexiones de PostgreSQL. Lee la cadena de conexión desde la variable de entorno DATABASE_URL y configura SSL para conectarse de forma segura a Supabase.
- **vercel.json**: Archivo de configuración para el despliegue en Vercel. Define las cabeceras CORS globales para todas las rutas /api/* y configura las reglas de reescritura de rutas, asegurando que las peticiones se enrutan correctamente en producción.
- **.env.local:** Archivo de variables de entorno que almacena datos sensibles como la cadena de conexión a la base de datos (DATABASE_URL), la clave de la API de Resend para el envío de emails y las direcciones de correo para el formulario de contacto.
- **jest.config.js**: Archivo de configuración del framework de testing Jest, utilizado para las pruebas unitarias del proyecto.
- **next.config.js:** Archivo de configuración general de Next.js.
- **tests/:** Carpeta que contiene los archivos de pruebas unitarias del backend.

Flujo de datos

La comunicación entre las dos capas sigue un flujo claro y consistente. El frontend realiza peticiones HTTP a través del cliente centralizado src/api/client.js, que actúa como punto único de entrada para todas las llamadas al backend. El backend recibe estas peticiones en los endpoints definidos en **app/api/… /route.js**, las procesa consultando la base de datos y devuelve respuestas en formato JSON. Estos datos son gestionados en los componentes Vue mediante ref() y computed(). Por ejemplo, en actividades.vue los datos obtenidos del API se combinan con información local del catálogo de actividades y se renderizan en forma de tarjetas junto a una tabla de horarios.

5.2 Implementación de funcionalidades

Reservas de actividades:

El endpoint /api/reservas gestiona las reservas mediante tres métodos. El GET recibe el id_usuario como parámetro de entrada y devuelve todas sus reservas activas, uniendo las tablas reserva, sesion y actividad para mostrar el título, franja horaria y estado de cada sesión.

El POST crea una nueva reserva utilizando una transacción SQL para garantizar la integridad de los datos. Antes de confirmar la reserva valida tres condiciones: que el usuario no tenga ya una reserva para esa sesión, que la sesión no esté suspendida y que haya plazas disponibles. Si todas las validaciones son correctas, inserta la reserva y resta en uno la capacidad de la sesión de forma atómica. Si alguna validación falla, se ejecuta un ROLLBACK para evitar inconsistencias.

El DELETE cancela una reserva existente a partir del id_sesion e id_usuario, verificando que ambos datos estén presentes antes de ejecutar la eliminación.

En el frontend, actividades.vue carga los horarios disponibles y permite al usuario reservar directamente haciendo clic sobre la sesión deseada.

Adicionalmente, el usuario puede consultar sus reservas activas desde su perfil en perfil.vue, donde se muestran todas las sesiones a las que está apuntado con el título de la actividad, el día y la franja horaria, permitiéndole también cancelarlas directamente desde ahí.

Autenticación (Login y Registro):

El sistema de autenticación se gestiona mediante dos endpoints en el backend, /api/login y /api/registro, que validan las credenciales contra la base de datos. Las contraseñas se almacenan hasheadas utilizando bcryptjs para garantizar la seguridad. En el frontend, el store auth.js gestiona el estado del usuario mediante refs reactivos y persiste la sesión en localStorage, de forma que el estado se recupera automáticamente al recargar la aplicación.

Carrito de compras:

El carrito está gestionado en el frontend mediante el store carritoStore.js, que utiliza refs reactivos y watchers para sincronizar automáticamente el estado con localStorage, garantizando la persistencia entre sesiones. Las funciones principales son agregarAlCarrito(), eliminarDelCarrito() y totalCarrito, esta última implementada como computed para recalcular automáticamente el total del producto añadido al carrito.

Al finalizar la compra, el frontend envía un POST a /api/pedidos con el id del usuario, el total y la lista de productos. El backend procesa la compra mediante una transacción SQL que sigue estos pasos; verifica que el saldo virtual del usuario sea suficiente para cubrir el total, descuenta el importe del saldo, crea el registro del pedido en la tabla pedido, y por cada producto valida que haya stock suficiente, lo resta del total de stock y registra la línea de compra en la tabla incluye. Si cualquiera de estos pasos falla, se ejecuta un ROLLBACK para evitar inconsistencias en la base de datos.

El historial de compras realizadas es accesible desde el perfil del usuario, donde puede consultar todos sus pedidos registrados.

Panel de administración:

El panel admin cuenta con una vista propia en el frontend (admin.vue), accesible exclusivamente para usuarios con rol administrador. Al cargar la página se verifica el rol del usuario y, si no es administrador, se redirige automáticamente al inicio, en cambio sí lo es en el header verá un botón que pone Gestion y ahí podrá acceder a la información que recibe de la base de datos. La interfaz está organizada en siete pestañas, cada una conectada a sus respectivos endpoints en /api/Admin/:

- Estadísticas: resumen general con total de usuarios, nuevos registros del mes, reservas y actividades.

- Usuarios: listado completo con nombre, email, rol y estado de cuenta calculado dinámicamente según si el usuario está suscrito, en periodo de prueba, caducado o admin.
- Actividades: edición de capacidad de cada actividad desde una fila expandible.
- Horarios: sesiones agrupadas por actividad en un acordeón desplegable, con opción de suspender o reactivar cada sesión individualmente.
- Reservas: vista de ocupación por sesión con usuarios apuntados, edición de capacidad y cancelación de reservas.
- Productos: edición de nombre, descripción, precio y stock de cada producto.
- Pedidos: historial de todos los pedidos agrupados por id, con detalle de productos, cantidades y totales.

Comunicación con la API

Toda la comunicación entre el frontend y el backend se centraliza en src/api/client.js, que expone dos funciones equivalentes: apiCall() y fetchApi(). Ambas construyen automáticamente la URL completa concatenando la URL base, definida en config.js mediante la variable de entorno VITE_API_URL, con el endpoint solicitado. Esto permite que durante el desarrollo las peticiones apunten al servidor local y en producción a la URL de Vercel sin necesidad de modificar ninguna llamada en el código.

Todos los componentes y stores de Vue utilizan fetchApi() como único punto de entrada para comunicarse con el backend, evitando URLs hardcodeadas y centralizando cualquier posible cambio de configuración en un solo archivo.

CORS y seguridad

La gestión de CORS se implementa en dos niveles para garantizar una cobertura completa. A nivel global, middleware.js intercepta todas las peticiones entrantes a las rutas /api/* antes de que lleguen a los endpoints, añadiendo automáticamente las cabeceras CORS a cada respuesta y gestionando las peticiones preflight OPTIONS.

A nivel de endpoint, lib/cors.js proporciona tres funciones auxiliares utilizadas en cada ruta; withCORS() para respuestas exitosas, withCORSError() para respuestas de error, y handleOPTIONS() para las peticiones preflight. Este doble nivel de configuración asegura que ninguna petición quede sin las cabeceras necesarias,

especialmente tras el despliegue en Vercel donde el vercel.json añade una tercera capa de cabeceras CORS a nivel de infraestructura.

En cuanto a la seguridad, todas las rutas validan la presencia de los campos obligatorios antes de ejecutar cualquier operación, y las contraseñas se almacenan siempre hasheadas con bcryptjs, nunca en texto plano.

5.3 Pruebas

Para verificar el correcto funcionamiento de la aplicación se realizaron pruebas unitarias tanto en el frontend como en el backend, utilizando Vitest en el frontend y Jest en el backend.

Backend (Jest):
Las pruebas del backend cubren los principales endpoints de la API, organizadas en seis archivos; compras.test.js, horarios.test.js, login.test.js, productos.test.js, registro.test.js y reservas.test.js. Por ejemplo, las pruebas del registro validan que todos los campos obligatorios estén presentes, que el formato del email sea correcto, que la contraseña tenga una longitud mínima de 6 caracteres, que los emails duplicados sean detectados y rechazados, que la contraseña se almacene hasheada y nunca en texto plano, y que la respuesta no exponga datos sensibles como el hash de la contraseña.

Frontend (Vitest):

Las pruebas del frontend se centran en el store de autenticación auth.js, verificando que loginUser() establece correctamente el usuario, activa el estado de autenticación y persiste en la sesión localStorage. También se prueba que logoutUser() limpia el estado y elimina la sesión almacenada, y que restoreSession() recupera correctamente la sesión al recargar la página, incluyendo casos de JSON inválido o ausencia de datos guardados.

Todas las pruebas han sido ejecutadas con el resultado esperado antes del despliegue final en Vercel.

6. Conclusiones

El desarrollo de FightCode ha sido un proyecto exigente que ha ido más allá que un simple proyecto curricular. A lo largo de estos meses el equipo técnico ha tenido que superar varias adversidades como problemas de salud de ambos integrantes que culminó en varios periodos de faltas combinado de problemas personales y mucha carga académica y por parte de las prácticas. Pese a todo ello, aquí está nuestro proyecto y el resultado final supera las expectativas que teníamos al inicio del curso.

Desde el punto de vista técnico, se han cumplido todos los objetivos planteados al inicio, la plataforma permite consultar actividades y horarios, realizar reservas en tiempo real, gestionar suscripciones, recargar saldo virtual y realizar compras online, todo ello acompañado de un panel de administración completo. Cabe destacar que funcionalidades como la tienda online y el sistema de suscripciones fueron las últimas en incorporarse, ya que durante gran parte del desarrollo no teníamos claro si habría tiempo suficiente para implementarlas. Finalmente, se dedicaron muchas horas extra para hacerlas posibles y poder entregar una aplicación completa.

La arquitectura fullstack con Vue 3, Next.js y Supabase ha demostrado ser una combinación sólida y escalable, y el despliegue en Vercel ha permitido que la aplicación funcione en un entorno real de producción. Uno de los mayores retos fue la gestión del tiempo, especialmente en los momentos de mayor carga, y la integración entre el frontend y el backend, que requirió coordinación constante y decisiones importantes sobre la marcha, como la migración de pgAdmin a Supabase.

No obstante, el tiempo también marcó los límites del proyecto. Quedaron fuera del alcance final dos funcionalidades que nos hubiera gustado implementar como la internacionalización de la aplicación para permitir cambiar el idioma a inglés u otros idiomas, y la mejora visual del panel de estadísticas del administrador con gráficos más vistosos e informativos. Ambas se consideran mejoras a incorporar en futuras versiones.

En conjunto, FightCode es un proyecto del que estamos orgullosos, tanto por la variedad de funcionalidades conseguidas como por el diseño y la experiencia de usuario lograda, y especialmente por haber sido capaces de sacarlo adelante en las circunstancias en las que lo hemos hecho.

6.1 Conclusiones generales

A nivel personal, este proyecto ha representado una experiencia de aprendizaje real y significativa. No solo hemos trabajado con tecnologías nuevas como Supabase o profundizado en el uso conjunto de Vue 3 y Next.js, sino que hemos vivido de primera mano lo que supone desarrollar una aplicación completa desde cero hasta su despliegue en producción.

Entre los aprendizajes más valiosos destacamos la capacidad de tomar decisiones técnicas bajo presión, como fue la migración de base de datos a mitad del proyecto, y la gestión del tiempo y las prioridades cuando los recursos son limitados. Trabajar en equipo en un proyecto de esta envergadura también nos ha enseñado la importancia de la comunicación constante y de repartir el trabajo de forma equilibrada.
Diseñar y construir una arquitectura fullstack de principio a fin ha sido uno de los retos más enriquecedores para nosotros, ya que obliga a entender cómo se conectan todas las piezas y a asumir la responsabilidad de cada decisión. Y quizás el aprendizaje más difícil de explicar pero el más satisfactorio es el que viene de enfrentarse a un problema que no funciona, dedicarle horas, y finalmente conseguir que funcione. Esa sensación es la que hace que merezca la pena.

6.2 Consecución de objetivos

Todos los objetivos planteados al inicio del proyecto han sido cumplidos satisfactoriamente. La plataforma permite gestionar horarios y realizar reservas en tiempo real, cuenta con una tienda online completamente funcional con carrito y pago mediante saldo virtual, aparte ofrece una amplia variedad de actividades deportivas. El diseño es moderno, funcional con el modo oscuro y responsive, adaptado a cualquier dispositivo, aparte la aplicación está desplegada en producción y accesible desde cualquier navegador.

Además, se ha implementado un sistema de suscripciones y un área de perfil personalizable que fomentan la fidelización del usuario, así como un panel de administración completo que permite gestionar de forma autónoma todos los recursos de la plataforma. El único aspecto que quedó fuera del alcance fue la internacionalización de la app y la mejora visual de las estadísticas del panel admin, ambas identificadas como mejoras para futuras versiones.

6.3 Valoración de la metodología y planificación

La metodología Scrum con tablero Kanban y GitHub funcionó bien, permitiendo repartir el trabajo con claridad y adaptarse a los cambios sin perder el control del proyecto. El flujo de trabajo en GitHub fue especialmente útil ya que cada integrante trabajaba en su propia rama y los cambios se integraban a la rama principal mediante pull requests, lo que permitió revisar el código antes de fusionarlo y evitar conflictos. La metodología iterativa fue clave para absorber imprevistos con el código o la incertidumbre sobre si habría tiempo para implementar la tienda y las suscripciones.

Como punto de mejora, haber definido desde el inicio un orden de prioridades más claro entre funcionalidades habría reducido la presión en los momentos más críticos. En general, la forma de trabajo fue adecuada y permitió llegar a una entrega completa.

6.4 Visión de futuro

FightCode es una base sólida sobre la que nos gustaría seguir construyendo. Las mejoras más inmediatas serían las que quedaron fuera por falta de tiempo, tal como la internacionalización de la plataforma para permitir cambiar el idioma, y la mejora del panel de estadísticas del administrador con gráficos más visuales e informativos. A medio plazo, nos plantearíamos integrar una pasarela de pago real como Stripe o PayPal para sustituir el saldo virtual por un sistema de pago directo, y añadir un sistema de valoraciones que permita a los usuarios puntuar y comentar las actividades a las que han asistido. También ampliaríamos las opciones del perfil de usuario, permitiendo una mayor personalización más allá de la foto, y seguiríamos puliendo la experiencia de navegación para hacerla aún más intuitiva.

A largo plazo, el paso natural sería desarrollar una aplicación móvil nativa que permitiera a los usuarios gestionar sus reservas y compras directamente desde el teléfono con una experiencia más optimizada de lo que ya es para pantallas pequeñas.

7.Glosario

API (Application Programming Interface): Conjunto de endpoints que permite la comunicación entre el frontend y el backend mediante peticiones HTTP.

Backend: Capa del servidor encargada de procesar la lógica de negocio, gestionar la base de datos y responder a las peticiones del frontend. En FightCode está desarrollado con Next.js.

bcryptjs: Librería de JavaScript utilizada para hashear contraseñas antes de almacenarlas en la base de datos, evitando guardarlas en texto plano.

CORS (Cross-Origin Resource Sharing): Mecanismo de seguridad de los navegadores que controla qué orígenes pueden hacer peticiones a un servidor.

Deploy/Despliegue: Proceso de publicar la aplicación en un servidor accesible desde internet. FightCode está desplegado en Vercel.

Endpoint: Ruta específica de la API que acepta peticiones HTTP para realizar una operación concreta, como crear una reserva o consultar productos.

Figma: Herramienta de diseño de interfaces utilizada para crear los prototipos y bocetos de la aplicación antes de comenzar a programar.

Frontend: Capa visual de la aplicación con la que interactúa el usuario directamente. En FightCode está desarrollada con Vue 3 y Vite.

Full-stack: Término que describe un proyecto que abarca tanto el desarrollo del frontend como del backend.

Hash: Transformación irreversible de un dato, como una contraseña, en una cadena de caracteres cifrada para su almacenamiento seguro.

Kanban: Método visual de gestión de tareas basado en columnas que representan el estado de cada tarea: pendiente, en progreso, completada, etc.

Pull request: Solicitud para fusionar los cambios de una rama a otra en GitHub, permitiendo revisar el código antes de integrarlo.

Responsive: Término que describe un diseño web que se adapta y funciona correctamente en distintos tamaños de pantalla, como ordenadores, tablets y móviles.

REST: Estilo de arquitectura para el diseño de APIs que utiliza los métodos HTTP (GET, POST, PUT, DELETE) para operar sobre recursos.

Scrum: Metodología ágil de desarrollo de software basada en ciclos iterativos y entregas progresivas.

8.Bibliografía

draw.io. (s/f). Drawio.com. Recuperado el 16 de mayo de 2026, de https://www.drawio.com

Next.Js by Vercel - the React framework. (s/f). Nextjs.org. Recuperado el 16 de mayo de 2026, de https://nextjs.org

Otto, M., & Thornton, J. (s/f). *Get started with bootstrap*. Getbootstrap.com. Recuperado el 16 de mayo de 2026, de https://getbootstrap.com/docs/5.3/getting-started/introduction/

Outdated browser. (s/f). Lucid Software. Recuperado el 16 de mayo de 2026, de https://lucid.app/documents#/home?folder_id=recent

Overview – Vercel. (s/f). Vercel.com. Recuperado el 16 de mayo de 2026, de https://vercel.com/lauras-projects-e36d72dd

Sign in to. (s/f).

The Postgres development platform. (s/f). Supabase. Recuperado el 16 de mayo de 2026, de https://supabase.com

Vite. (s/f). Vitejs. Recuperado el 16 de mayo de 2026, de https://vite.dev

Vue.Js. (s/f). Vuejs.org. Recuperado el 16 de mayo de 2026, de https://vuejs.org

W3Schools online web tutorials. (n.d.). W3schools.com. Retrieved May 17, 2026, de https://www.w3schools.com/

Jest. (n.d.). Jestjs.Io. Retrieved May 17, 2026, de https://jestjs.io

MDN Web Docs. (n.d.). MDN Web Docs. Retrieved May 17, 2026, de https://developer.mozilla.org

Pinia 🍍. (n.d.). Vuejs.org. Retrieved May 17, 2026, de https://pinia.vuejs.org

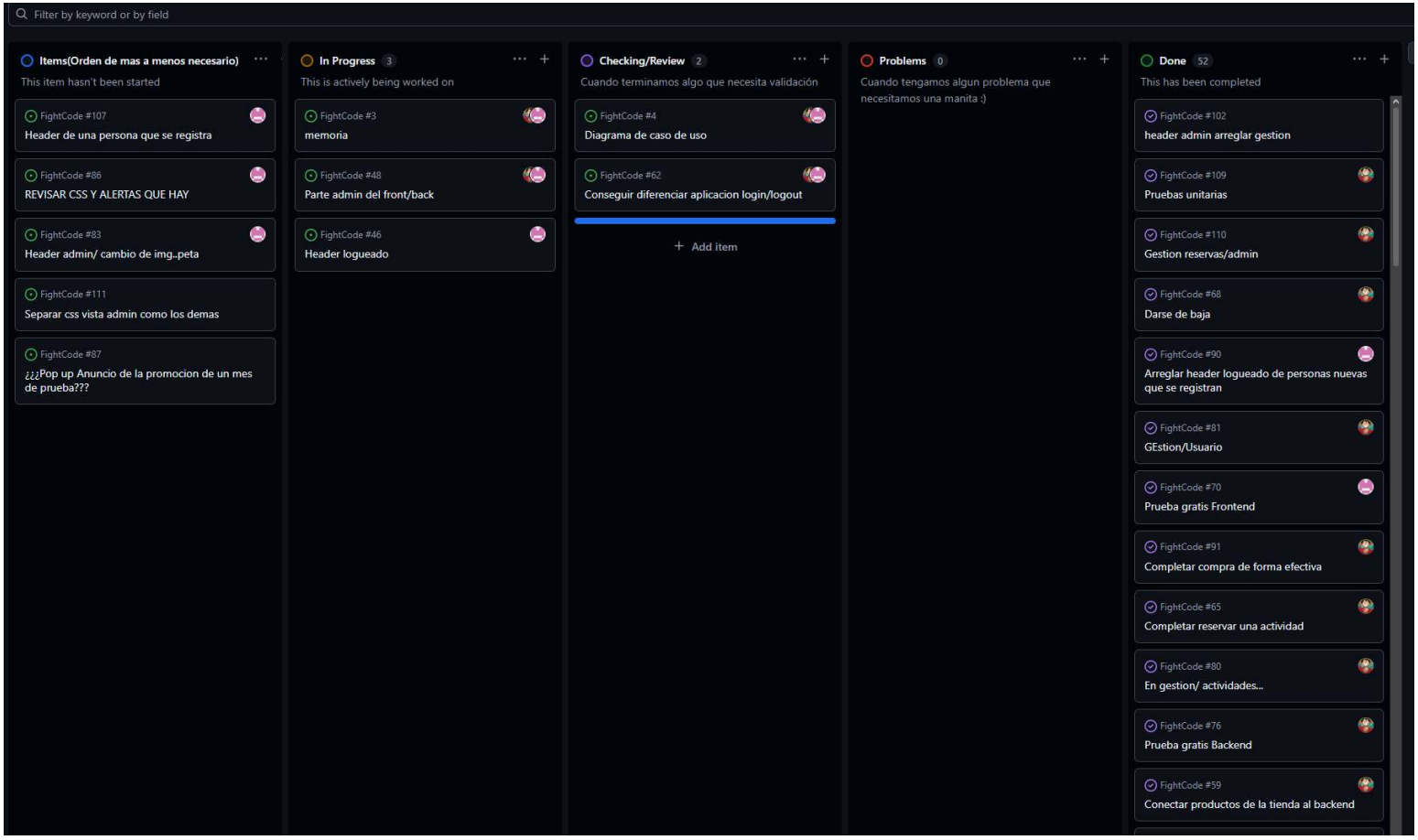
Vladimir, Fu, A., Perkkiö, A., Ogawa, H., Patak, & Sánchez, J. (n.d.). *Vitest*. Vitest.dev. Retrieved May 17, 2026, de https://vitest.dev/

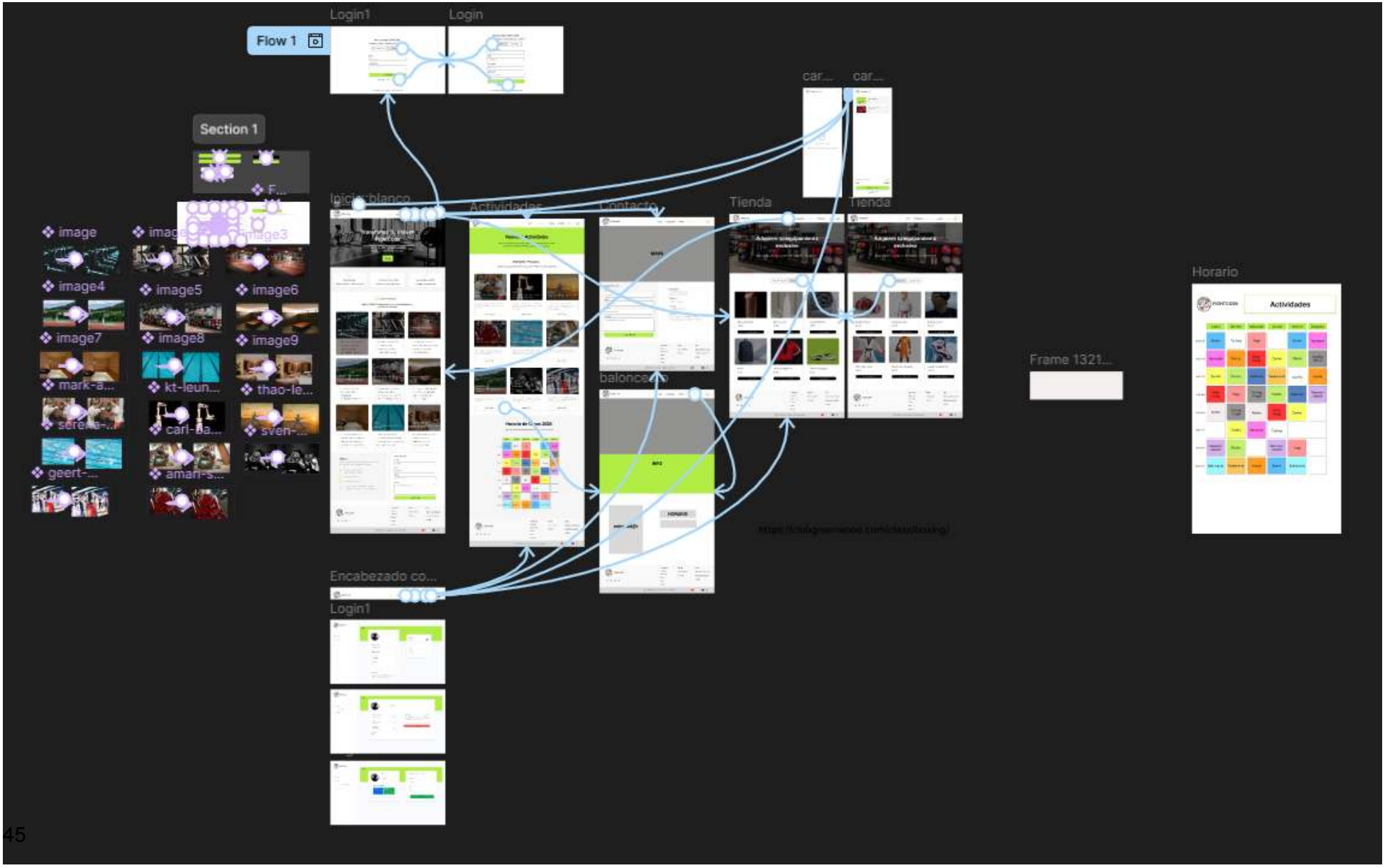
9.Annexos

9.1 GitHub

★ **URL del Repositorio del proyecto:** https://github.com/york247/FightCode.git

9.2 Proceso en kamban




Login1
Login
Flow 1
Section 1
car...
car...
Inicio blanco
Actividades
Contacto
Tienda
Tienda
Horario
Actividades
Frame 1321...
baloncesto
Encabezado co...
Login1
image
image4
image5
image6
image7
image8
image9
mark-a...
kt-leun...
thao-le...
carl-da...
sven-...
geert-...
aman-s...