



FITFORGE

FPGS Desarrollo de
Aplicaciones Web

ROY LEE LUNAR MORÉ



RESUMEN

La finalidad de este proyecto es diseñar e implementar una aplicación web centrada en la mejora de la salud del usuario mediante el seguimiento personalizado del entrenamiento. Esta herramienta busca facilitar la creación de planes deportivos adaptados a cada individuo, integrando tecnologías modernas y buenas prácticas de desarrollo web.

DESARROLLO DE LA APLICACIÓN

Frontend: Se ha desarrollado con Vue 3 usando el lenguaje de programación JavaScript y Tailwind CSS para el diseño responsive.

Backend: Se ha utilizado Node.js junto con el framework Express usando el lenguaje de programación JavaScript, aplicando la arquitectura (MVC) y organizando rutas, middlewares y lógica de negocio de forma modular.

Base de datos: Como base de datos se ha usado PostgreSQL como sistema de gestión relacional, accedido mediante Prisma ORM para la manipulación de datos de forma segura y estructurada.

Autenticación: Se ha implementado un sistema de login basado en tokens JWT, almacenando los datos esenciales en el token y el resto de información se obtiene mediante endpoints.

Despliegue: La aplicación se ha desplegado en una VPS usando Docker Compose con configuración de entorno y control de variables con el uso de dotenv, usando CORS para gestionar la comunicación entre frontend y backend.

FUNCIONALIDAD

Los usuarios se registran e introducen sus datos físicos y objetivos (ganar masa muscular, perder peso, etc.), lo cual permite generar planes personalizados.

Se genera automáticamente un plan de entrenamiento según el objetivo y nivel seleccionado. Este se estructura por días, ejercicios y grupos musculares.

Pueden registrar su progreso los usuarios, modificar sus rutinas y visualizar su evolución a través de gráficas interactivas. Se incorpora un sistema de roles para diferenciar entre usuarios normales y administradores, quienes pueden gestionar los ejercicios y planes disponibles.

MECANISMOS

- Generación automática de rutinas en función del objetivo físico del usuario.
- Seguimiento del progreso con gráficas (peso, IMC, ejercicios realizados).
- Gestión CRUD de ejercicios y planes (solo para administradores).
- Sistema visual con SVG del cuerpo humano coloreado según el IMC.
- Autenticación con JWT y protección de rutas con middlewares personalizados.

ÍNDICE

1. Introducción.....	6
1.1 Introducción a la memoria.....	6
1.2 Descripción.....	6
1.3 Objetivos Generales.....	7
1.5 Motivaciones Personales.....	8
1.6 Estructura de la Memoria.....	8
1.7 Repositorio Github.....	8
2. Estudio de la Viabilidad.....	10
2.1 Introducción.....	10
2.2 Estudio de la situación actual.....	10
2.3 Objetivos del proyecto.....	11
2.4 Planificación del proyecto.....	12
2.5 Presupuesto.....	14
3. Análisis del Proyecto.....	16
3.1 Introducción.....	16
3.2 Requisitos Funcionales.....	16
3.3 Requisitos No Funcionales.....	18
3.4 Casos de uso.....	19
3.5 Diagramas de flujo.....	21
3.5.1 Login.....	21
3.5.2 Crear plan automático.....	22
3.5.2 Editar perfil.....	23
4. Diseño.....	24
4.1 Introducción.....	24
4.2 Diseño de la base de datos.....	24
4.3 Diseño de la interfaz de usuario.....	26
4.3.1 Vista Login.....	26
4.3.2 Vista Register.....	27

4.3.3 Vista Perfil.....	27
4.3.4 Vista Admin.....	28
4.3.5 Vista Ejercicios.....	28
4.3.6 Vista Rutinas.....	29
4.3.7 Vista Dashboard.....	30
4.3.8 Vista Estadísticas.....	30
4.4 Tecnologías utilizadas.....	34
Backend:.....	34
Frontend.....	35
5. Planteamiento e Implementación.....	37
5.1 Introducción.....	37
5.2 Arquitectura de la aplicación.....	37
5.2.1 Frontend.....	37
5.2.1.1 Vistas:.....	40
5.2.1.2 Componentes:.....	49
5.2.1.3 Utilidades y Stores:.....	52
5.2.2 Backend.....	53
5.2.2.1 Controladores:.....	55
5.2.2.2 Modelos:.....	63
5.2.2.3 Middlewares:.....	66
5.3 Seguridad de la aplicación.....	69
6. Pruebas.....	71
6.1 Introducción.....	71
6.2 Pruebas Con Postman - BackEnd.....	71
6.3 Pruebas Con Selenium - FrontEnd.....	72
7. Despliegue de la Aplicación.....	74
7.1 Introducción.....	74
7.2 Proceso de despliegue.....	74
7.3 Esquema final de la estructura.....	76

8. Conclusiones.....	77
8.1 Conclusiones Finales.....	77
8.2 Desviaciones Temporales.....	77
8.3 Posibles ampliaciones.....	78
8.4 Valoración General.....	78
9. Bibliografía.....	80
10. Anexos.....	81
10.1 Casos de uso.....	81
10.2 Guía de estilos.....	86
10.3 Lógica Rutinas Automáticas.....	90
10.4 Enlace a la aplicación.....	91

1. Introducción

1.1 Introducción a la memoria

En esta memoria se recoge el desarrollo de una aplicación web orientada al ámbito fitness de forma personalizada. El objetivo principal es aplicar los conocimientos adquiridos a lo largo del grado para construir una solución completa que permita a los usuarios gestionar sus entrenamientos, alimentos, realizar un seguimiento de su evolución física y acceder a planes deportivos adaptados a sus objetivos.

El documento abarca todas las fases del proyecto, desde el análisis inicial hasta la implementación y el despliegue final, incluyendo el estudio de viabilidad, la estimación de costes, el diseño técnico, el diseño de la interfaz y aspectos visuales, el proceso de despliegue, y la exposición de los objetivos y resultados alcanzados.

1.2 Descripción

El proyecto tiene como objetivo el desarrollo e implementación de una aplicación web fitness personalizada, orientada a la gestión de planes de entrenamiento y control físico del usuario. Se busca cumplir con las expectativas de los usuarios finales, ofreciendo una solución robusta, funcional y escalable, preparada para evolucionar en los próximos años.

El funcionamiento general de la plataforma se centra en la creación automática de planes deportivos personalizados según los objetivos y condiciones físicas del usuario, así como en el seguimiento de su progreso físico y deportivo. Adicionalmente, la plataforma permite la gestión de rutinas de ejercicios, el control de métricas personales (como peso, altura o IMC), y la generación de estadísticas visuales que reflejan la evolución del usuario. También se contempla la futura integración de planes nutricionales, sistemas de logros y exportación de rutinas en formatos como PDF.

Para el desarrollo del proyecto se han empleado las siguientes tecnologías:

Para ello, se han utilizado tecnologías como Node.js con Express.js para el backend (API), Vue.js y Axios para el frontend y la comunicación cliente-servidor, y

PostgreSQL como sistema gestor de bases de datos. Se ha empleado Prisma ORM para la gestión de modelos, migraciones y seeders a la base de datos, también se utilizó herramientas como Tailwind CSS y Anime.js para mejorar la interfaz, las animaciones y la responsividad de la plataforma. El desarrollo se ha basado en una arquitectura modular y escalable siempre siguiendo un patrón de arquitectura de software MVC utilizando software libre en todo el proceso.

1.3 Objetivos Generales

El objetivo general de este proyecto es desarrollar una aplicación web haciendo uso de tecnologías y estándares modernos usados en el desarrollo web, el proyecto está centrado en el ámbito del fitness y la alimentación, que permita a los usuarios gestionar planes de entrenamiento adaptados a sus objetivos físicos, llevar un seguimiento de su evolución y visualizar estadísticas personalizadas.

Para ello, se pretende diseñar e implementar una solución funcional, robusta y escalable que cubra todas las fases del desarrollo de software, aplicando metodologías ágiles, buenas prácticas de programación y tecnologías modernas tanto en el lado cliente como en el servidor. Asimismo, se busca garantizar una experiencia de usuario intuitiva, responsive y escalable contemplando que en un futuro se encuentre disponible incluso en dispositivos móviles.

1.4 Beneficios

Este proyecto permite consolidar los conocimientos sobre desarrollo web, bases de datos y metodologías ágiles, corporativas y modernas. A nivel funcional, ofrece a los usuarios como se explicó en apartados anteriores una herramienta para gestionar su entrenamiento de forma personalizada y hacer seguimiento de su progreso físico y salud. Además, sienta las bases para futuras ampliaciones, como la inclusión de planes nutricionales, sistemas de experiencia para alentar a los usuarios a llevar una vida saludable, la integración con múltiples dispositivos externos y una mayor accesibilidad que incluya a personas ancianas con discapacidades o múltiples patologías permitiendo la evolución continua de la aplicación.

1.5 Motivaciones Personales

Las motivaciones que impulsan el desarrollo de este proyecto es el uso de tecnologías actuales y modernas en el mercado, con el objetivo de crear una solución útil y funcional para los usuarios. Su interés radica en seguir las prácticas de desarrollo utilizadas en el ámbito corporativo y en la creación de aplicaciones modernas, ya sea en la nube, web o móvil. Además de la falta de aplicaciones completas que gestionen de manera integral la salud y el entrenamiento de los usuarios le ha impulsado a desarrollar esta herramienta, que permita una gestión personalizada y eficiente del bienestar físico.

1.6 Estructura de la Memoria

Esta memoria está organizada en varias secciones que cubren todas las fases del proyecto, desde su concepción hasta su despliegue final.

Comenzaremos con la **introducción**, donde se presentará el objetivo y las motivaciones del proyecto. A continuación, en el **estudio de viabilidad**, se evaluará la viabilidad técnica, económica y operativa del proyecto. En el **análisis del proyecto**, se detallarán los requisitos del sistema y se presentarán los casos de uso y diagramas correspondientes.

La sección de **diseño** abordará la arquitectura, la base de datos y la interfaz de usuario, mientras que en **planteamiento e implementación** se explicará el desarrollo del sistema, incluyendo las tecnologías y la estructura del proyecto. En la parte de **pruebas**, se describirán las pruebas realizadas y sus resultados.

El **despliegue** se explicará en una sección dedicada, detallando el proceso para poner la aplicación en producción. Finalmente, en las conclusiones, se reflexionará sobre los logros del proyecto y posibles mejoras, y se cerrará con la bibliografía.

1.7 Repositorio Github

Durante todo el desarrollo del proyecto, se ha utilizado Git y GitHub como sistema de control de versiones. Esto ha permitido mantener un registro detallado y ordenado donde se puede consultar el código fuente, historial detallado de los cambios, facilitar la integración continua de nuevas funcionalidades (CI) y el

despliegue continuo en contenedores Docker y VPS (CD); asociados al desarrollo de la aplicación.

La estructura de trabajo se ha organizado en torno a tres ramas principales: (main, develop y deploy). Esta separación permite aislar las distintas etapas del desarrollo, facilitando la implementación, integración y pruebas de nuevas funcionalidades sin representar posibles conflictos y errores.

(main): Contiene el código estable, solo se actualiza con versiones completamente probadas y funcionales. (develop): Es la rama de desarrollo, aquí se integran las nuevas funcionalidades antes de enviar a main. (deploy): Es la rama que permite preparar y gestionar el despliegue de la aplicación esto incluye (CI/CD) con GitHub Actions y DockerHub.

[FitForge - Repositorio GitHub](#)

The screenshot shows the GitHub repository page for 'FitForge' (Public). At the top, there are navigation links for Code, Issues, Pull requests, Actions, Projects, Security, Insights, and Settings. Below the header, the repository name 'FitForge' and owner 'Lubriom' are displayed, along with a star icon indicating it has 1 star. The main content area shows a list of 39 commits from Lubriom, merged into the 'main' branch. The commits are as follows:

Commit	Message	Time Ago
backend	Added change to project in gitignore	4 hours ago
frontend	Adjust in Package.json	5 hours ago
.gitignore	Added change to project in gitignore	4 hours ago
LICENSE	Initial commit	2 months ago
README.md	Initial commit	2 months ago
ayuda.md	[Back & Front] Added CRUD from exercises, updated token ...	3 weeks ago
package-lock.json	Start vue in front	2 months ago

Below the commits, there are links to 'README' and 'GPL-3.0 license'. On the right side of the page, there are sections for 'About', 'Releases', 'Packages', and 'Contributors'. The 'About' section describes FitForge as an application web to enhance your health and fitness. The 'Releases' section indicates no releases have been published. The 'Packages' section shows no packages have been published, with a link to 'Publish your first package'. The 'Contributors' section lists 2 contributors.

2. Estudio de la Viabilidad

2.1 Introducción

Este apartado tiene como finalidad realizar un estudio de la viabilidad del proyecto para determinar la factibilidad del proyecto desde una perspectiva técnica y económica, y sentará las bases para su posterior desarrollo e implementación.

Para ello, se lleva a cabo un estudio de la situación actual del mercado, la definición clara de los objetivos que se pretenden alcanzar, la recopilación de los requisitos funcionales y no funcionales del sistema, y la planificación y estimación de recursos necesarios.

2.2 Estudio de la situación actual

La idea de esta aplicación surge como respuesta a la creciente preocupación por el bienestar físico y la salud personal, en un contexto donde muchas personas desean adoptar hábitos saludables, pero no cuentan con herramientas integrales que les permitan gestionar eficazmente su progreso. Actualmente, existen múltiples aplicaciones centradas en el seguimiento de entrenamientos, pero la mayoría de ellas se enfocan de forma aislada en un solo ámbito, careciendo de una visión unificada del estado físico del usuario.

Además, muchas de estas soluciones no están adaptadas a las necesidades individuales, limitándose a ofrecer rutinas genéricas sin tener en cuenta las condiciones personales, como patologías previas, discapacidad, nivel de experiencia o el objetivo concreto que desea alcanzar el usuario (como pérdida de peso, ganancia de masa muscular o mejora del rendimiento).

Debido a esto, se detecta una oportunidad para desarrollar una herramienta accesible, personalizable y basada en datos reales del usuario, que permita gestionar rutinas de entrenamiento, registrar información física relevante, y generar planes adaptados a los objetivos particulares de cada persona. Esta solución pretende cubrir una carencia en un mercado que día a día crece aún más, apostando por un enfoque más completo, flexible y orientado a la mejora continua del estado físico de cada usuario.

En la fase de desarrollo se tendrán en cuenta 3 perfiles de usuario distintos siendo estos los siguientes:

- Usuario (Persona que usará la aplicación para gestionar sus rutinas, etc)
- Entrenador (Entrenador que hará uso de la aplicación para gestionar los entrenamientos de cada usuarios)
- Administrador (Persona con acceso total de la aplicación y control del sistema)

Una vez terminada la fase de desarrollo se configurará un despliegue para el uso general de la aplicación de forma global.

2.3 Objetivos del proyecto

Tras el análisis de las necesidades detectadas en el ámbito del entrenamiento y la salud personal, se han establecido los siguientes objetivos principales para el desarrollo del proyecto:

- Desarrollar una aplicación web que permita a los usuarios gestionar de forma sencilla y personalizada sus planes de entrenamiento.
- Implementar un sistema para registrar y visualizar la evolución física del usuario mediante datos como peso, altura, índice de masa corporal (IMC) o repetición máxima (RM).
- Permitir la creación, edición y seguimiento de rutinas deportivas adaptadas al objetivo y nivel del usuario (como fuerza o hipertrofia, y nivel básico o avanzado).
- Diseñar un sistema de autenticación que permita el acceso seguro a la plataforma, diferenciando perfiles de usuario por roles.
- Ofrecer una experiencia de usuario completamente responsive, permitiendo el acceso desde distintos dispositivos, especialmente móviles.
- Facilitar la visualización del progreso del usuario mediante gráficos y estadísticas personalizadas.
- Integrar mecanismos que permitan, en un futuro, la incorporación de funcionalidades como recordatorios, sistema de logros o integración con

dispositivos externos.

Estos objetivos buscan no solo cubrir las necesidades funcionales básicas del sistema, sino también ofrecer valor añadido al usuario mediante una herramienta versátil, accesible y centrada en la mejora continua de su estado físico.

2.4 Planificación del proyecto

Este subapartado detalla la planificación y gestión del proyecto, siguiendo las etapas habituales del ciclo de vida de desarrollo de software. Se ha procurado una distribución lógica y progresiva del trabajo, comenzando desde la recopilación de requisitos hasta el despliegue final, incluyendo también las operaciones y validaciones posteriores. A continuación, se presenta el desglose estimado del tiempo dedicado a cada fase del desarrollo, incluyendo tareas asociadas.

Requisitos (12 horas)

- Análisis de necesidades del usuario y planteamiento de funcionalidades clave.
- Investigación de recursos, tecnologías y librerías a utilizar (Express, Prisma, Vue, etc.).
- Estudio de casos similares y definición de alcance.

Diseño (25 horas)

- Estructuración del modelo de datos y definición de relaciones en la base de datos.
- Diseño de arquitectura backend basada en el patrón MVC.
- Planificación del diseño de interfaz con Tailwind CSS, y esquemas de interacción.
- Consideraciones de responsividad y experiencia de usuario.

Desarrollo (150 horas)

- Creación del backend con Node.js y Express: rutas, controladores, servicios, middlewares.
- Configuración de Prisma para la gestión de datos con PostgreSQL.
- Desarrollo del frontend utilizando Vue 3 con la API de composición y Vite.
- Implementación de formularios dinámicos y validaciones con Zod.
- Gestión de estados globales con pinia, integración de Anime.js y consumo de API REST.
- Implementación del sistema de autenticación con JWT y almacenamiento de sesión en authStore.

Pruebas (20 horas)

- Verificación del funcionamiento de rutas, autenticación, validaciones y lógica de negocio.
- Revisión del CRUD completo para ejercicios, usuarios, planes, etc. Comprobación de protecciones de seguridad (CSRF, validación de roles, etc.).
- Corrección de errores y refactorización de código para mejorar la mantenibilidad.

Despliegue (16 horas)

- Configuración del entorno de producción y variables en servidor.
- Pruebas de despliegue utilizando servicios como Render o Railway.
- Gestión de errores en producción y adaptación de configuraciones.

Operaciones / Pruebas finales (10 horas)

- Validación completa de la aplicación en el entorno de producción.
- Verificación de responsividad, rendimiento y consistencia de datos.
- Revisión de posibles mejoras y anotaciones finales.

2.5 Presupuesto

Todas las herramientas usadas durante la realización del proyecto son completamente gratuitas. Entre estas herramientas se encuentran tecnologías como Node.js, Express, Vue.js, Tailwind CSS, Anime.js, Prisma, PostgreSQL, Vite y otras librerías complementarias, todas disponibles bajo licencias de software libre.

En cuanto al despliegue, se ha alojado la aplicación web en un Servidor Privado Virtual (VPS) teniendo un coste total de 4,50€ mensuales, aunque varía su precio según el proveedor y los recursos contratados.

Además, si se plantea desarrollar este proyecto como algo a nivel profesional, es necesario realizar una estimación de costes basado en el tiempo y esfuerzo invertido. Considerando el **salario mínimo por hora en España en 2025**, que se sitúa en **7,88 €**, se obtiene el siguiente cálculo:

Etapa	Horas estimadas	Coste estimado
Requisitos	12 horas	94,56 €
Diseño	25 horas	197,00 €
Desarrollo	150 horas	1.182,00 €
Pruebas	20 horas	157,60 €
Despliegue	16 horas	126,08 €
Operaciones / Pruebas finales	10 horas	78,80 €
Total sin documentación	233 horas	1.836,04 €
Documentación (aprox.)	30 horas	236,40 €
Total con documentación incluida	263 horas	2.072,44 €

Este cálculo refleja el valor aproximado que podría tener el desarrollo de una aplicación con características similares en un entorno profesional, considerando únicamente el tiempo de trabajo. No se han incluido en esta estimación los posibles costes adicionales relacionados con licencias comerciales, certificados SSL, herramientas de diseño o recursos gráficos externos.

3. Análisis del Proyecto

3.1 Introducción

Este apartado tiene como finalidad definir las características funcionales y técnicas que debe cumplir el sistema para garantizar su correcto funcionamiento, así como establecer los fundamentos necesarios para su desarrollo y evaluación. En él se detallan los requisitos funcionales que debe ofrecer la aplicación al usuario, así como los requisitos no funcionales que establecen los criterios de calidad, rendimiento, seguridad y mantenibilidad del software.

Además, se identifican los distintos roles de usuario con sus respectivos casos de uso, con la intención de definir claramente las funcionalidades disponibles de cada usuario según su rol. Finalmente, se incluyen diagramas de flujo que representan de forma visual el comportamiento del sistema, facilitando la comprensión de los procesos clave de la aplicación.

3.2 Requisitos Funcionales

Los requisitos funcionales definen las capacidades y servicios que la aplicación debe proporcionar al usuario para cumplir sus objetivos. Se han identificado los siguientes:

1. Autenticación y gestión de usuarios

La aplicación debe permitir el registro y login de usuarios mediante un sistema de autenticación basado en correo electrónico y contraseña, con las credenciales correctamente validadas y almacenadas de forma segura. Además, el sistema debe diferenciar entre distintos tipos de usuario (como administradores, entrenadores y usuarios estándar), restringiendo o habilitando funcionalidades en función del rol asignado.

2. Gestión de datos físicos del usuario

Los usuarios deben poder introducir, editar y consultar sus datos físicos personales tales como peso, altura, índice de masa corporal (IMC) y valores como la repetición máxima (RM). Estos datos serán utilizados para calcular

métricas personalizadas y generar recomendaciones o rutinas más adecuadas al perfil del usuario.

3. Creación y personalización de planes de entrenamiento

El sistema debe permitir a los usuarios crear planes de entrenamiento ajustados a sus objetivos (por ejemplo, fuerza o hipertrofia) y su nivel de experiencia (básico o avanzado). Cada plan podrá dividirse en días de entrenamiento, a los que se asociarán ejercicios con parámetros configurables como número de series, repeticiones, peso sugerido y tiempo de descanso entre series.

4. Gestión de ejercicios por día de entrenamiento

La aplicación cuenta con una interfaz que permite añadir, editar y eliminar ejercicios específicos dentro de los días del plan de entrenamiento. Cada ejercicio podrá ser configurado de forma individual, permitiendo ajustar el esfuerzo y volumen de entrenamiento según la progresión o necesidades del usuario.

5. Registro de condiciones personales (patologías/discapacidades)

Con el objetivo de ofrecer un enfoque inclusivo y personalizado, el sistema debe permitir el registro de condiciones médicas o limitaciones físicas del usuario. Estos datos podrán utilizarse para filtrar ejercicios incompatibles o adaptar las rutinas, evitando situaciones de riesgo.

6. Seguimiento del progreso mediante estadísticas

Los usuarios deben poder visualizar su evolución a lo largo del tiempo mediante estadísticas gráficas, que reflejan métricas como cambios en el peso, IMC, volumen de entrenamiento o mejora de RM. Esta información se recopila automáticamente a partir de los datos introducidos por el usuario y sus registros de actividad.

7. Diseño responsive y accesible

Toda la interfaz de la aplicación debe estar diseñada siguiendo principios de diseño responsivo, asegurando su uso correcto en distintos dispositivos como

ordenadores, tablets y teléfonos móviles. La experiencia de usuario debe ser fluida y accesible, independientemente del tamaño de pantalla.

3.3 Requisitos No Funcionales

Los requisitos no funcionales establecen los criterios de calidad y las condiciones técnicas bajo las cuales debe operar el sistema. Estos requisitos son fundamentales para garantizar una experiencia de usuario óptima, así como para asegurar la escalabilidad y mantenibilidad del software.

1. Rendimiento y tiempo de respuesta

El sistema debe ofrecer un tiempo de respuesta adecuado y fluido en operaciones básicas como el registro, carga de datos, navegación entre vistas o visualización de estadísticas. El objetivo es mantener una latencia perceptiva baja, de forma que las interacciones del usuario no se vean interrumpidas o ralentizadas, incluso con un volumen moderado de datos.

2. Seguridad y protección de datos

Se implementarán buenas prácticas de seguridad web, como el cifrado de contraseñas con algoritmos robustos (por ejemplo, bcrypt), validaciones de entradas en el frontend y backend, protección frente a ataques CSRF por cabecera Authorization en peticiones HTTP, XSS (Cross-Site Scripting), e inyecciones de código malicioso. Además, el sistema debe cumplir con criterios básicos de privacidad en el tratamiento de los datos personales de los usuarios.

3. Arquitectura moderna y escalable

El backend se desarrollará utilizando Node.js con Express y una arquitectura basada en el patrón Modelo-Vista-Controlador (MVC), lo que permite una separación clara de responsabilidades y facilita el mantenimiento y escalabilidad del proyecto. Esto también permitirá incorporar nuevas funcionalidades sin afectar negativamente a las ya existentes.

4. Gestión de base de datos con ORM

La gestión de la base de datos se realizará mediante Prisma, un ORM moderno que facilita la interacción con la base de datos relacional PostgreSQL. Esto proporciona tipado estático, migraciones estructuradas y una experiencia de desarrollo más robusta, reduciendo los errores y ataques comunes en las consultas SQL manuales.

5. Preparación para despliegue en entorno productivo

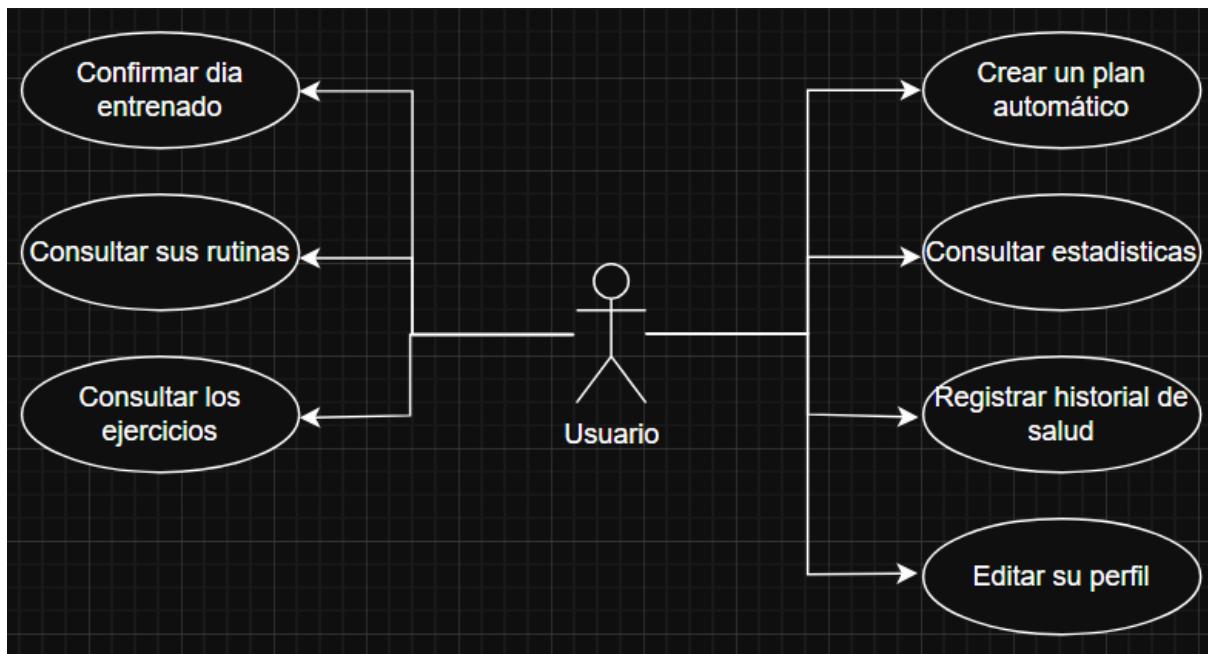
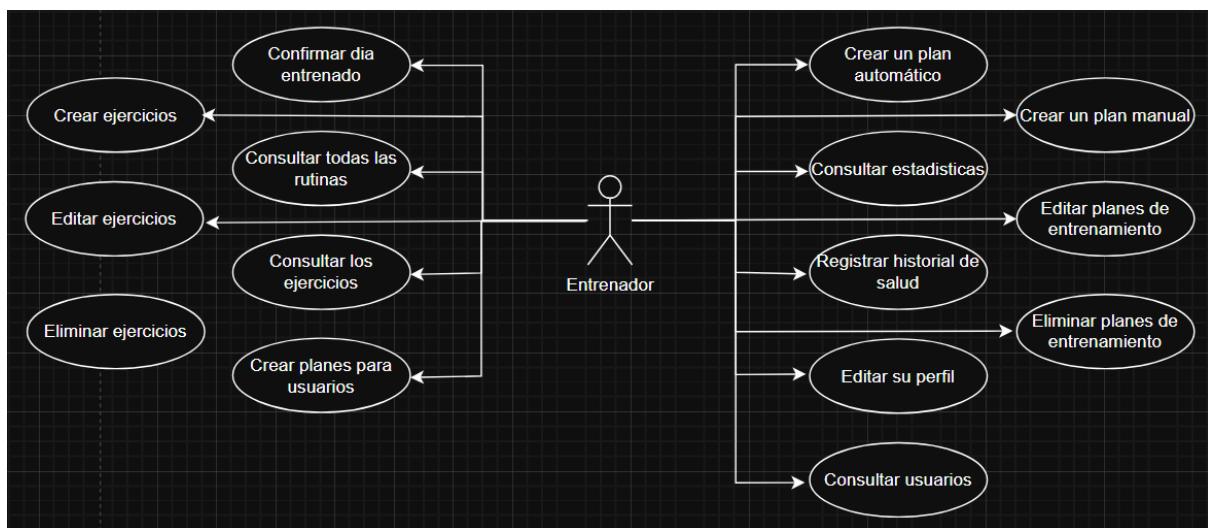
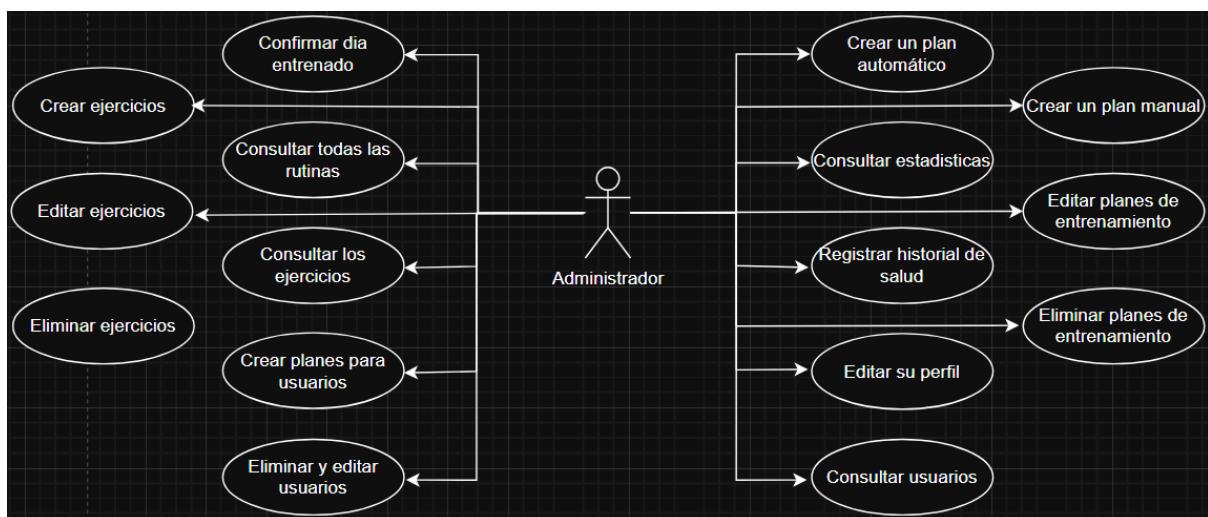
La aplicación debe estar preparada para su despliegue en un entorno de producción, utilizando herramientas modernas de despliegue y automatización como Docker, GitHub Actions o Vercel, en caso de ser necesario. El sistema también debe ser compatible con entornos de red cerrados, como podría ser un servidor local del instituto, sin requerir internet externo.

6. Preparación para futuras integraciones

La arquitectura del sistema debe facilitar, sin grandes refactorizaciones, la futura integración con APIs externas (por ejemplo, bases de datos de alimentos o dispositivos de medición) y la incorporación de nuevas funcionalidades como notificaciones, gamificación o servicios de terceros. Esto implica un diseño modular, flexible y bien documentado.

3.4 Casos de uso

Rol de Usuario:

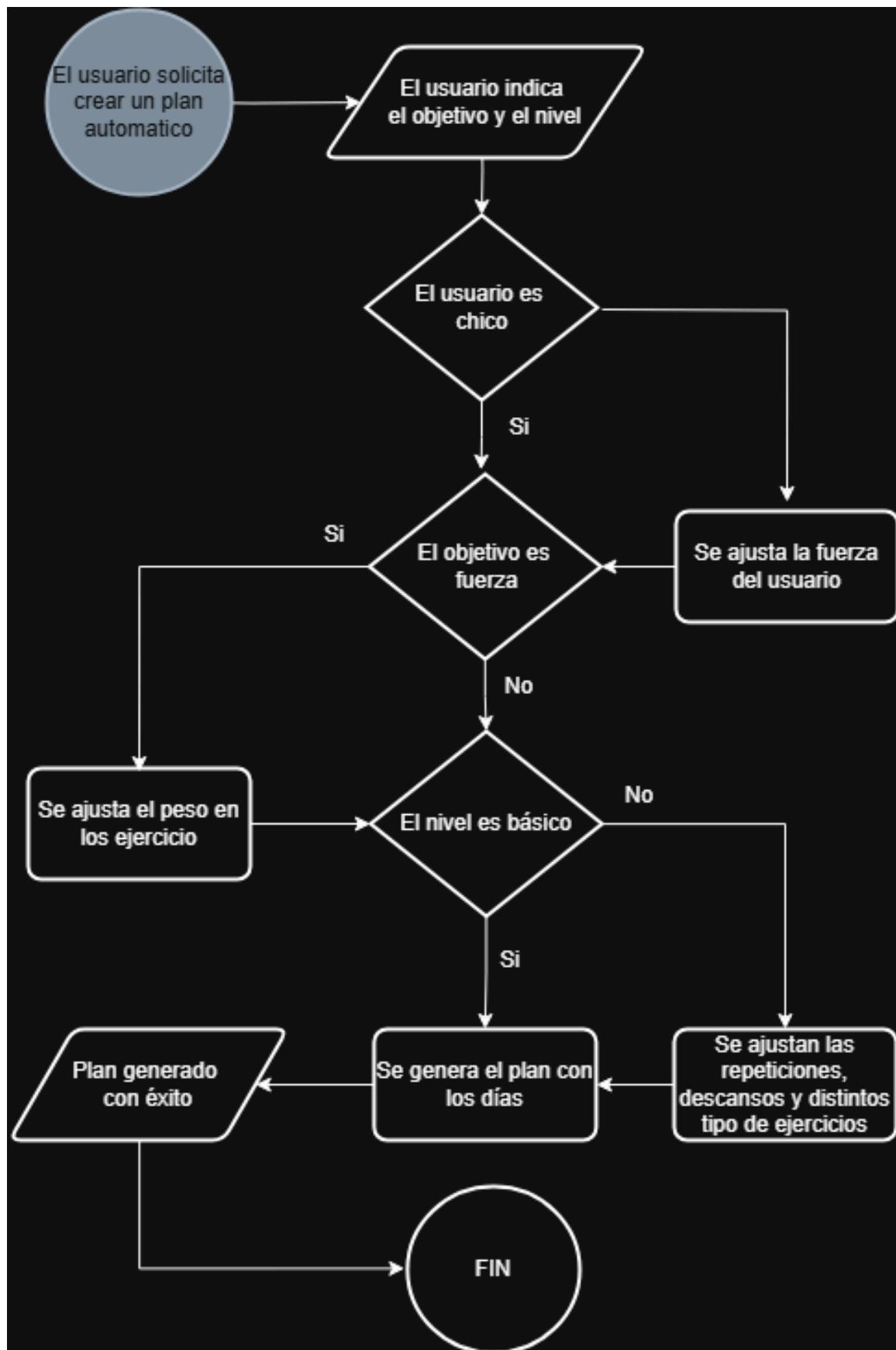
**Rol de Entrenador:****Rol de Administrador:**

3.5 Diagramas de flujo

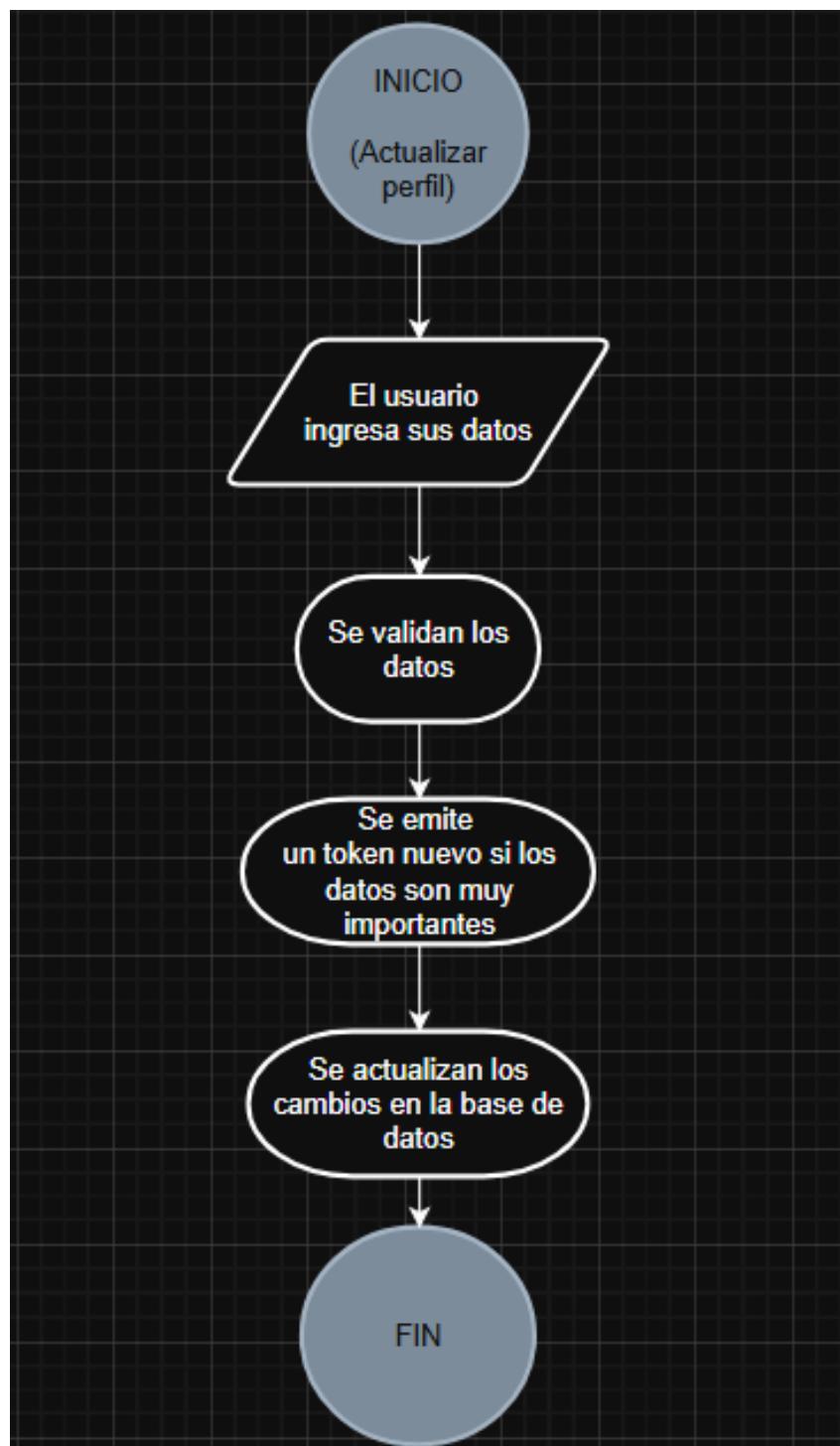
3.5.1 Login



3.5.2 Crear plan automático



3.5.2 Editar perfil



Se han elaborado los diagramas más relevantes tanto para los casos de uso como para los diagramas de flujo, entre estos se elaboraron aquellos imprescindibles para comprender con claridad la estructura del sistema y el funcionamiento general. Inicialmente, se ha abordado el comportamiento del sistema mediante diagramas de casos de uso y diagramas de secuencia.

4. Diseño

4.1 Introducción

En este apartado abordaremos todo lo relacionado al diseño, desarrollo y estructura de la aplicación web incluyendo entre sí, base de datos, vista y clases.

4.2 Diseño de la base de datos

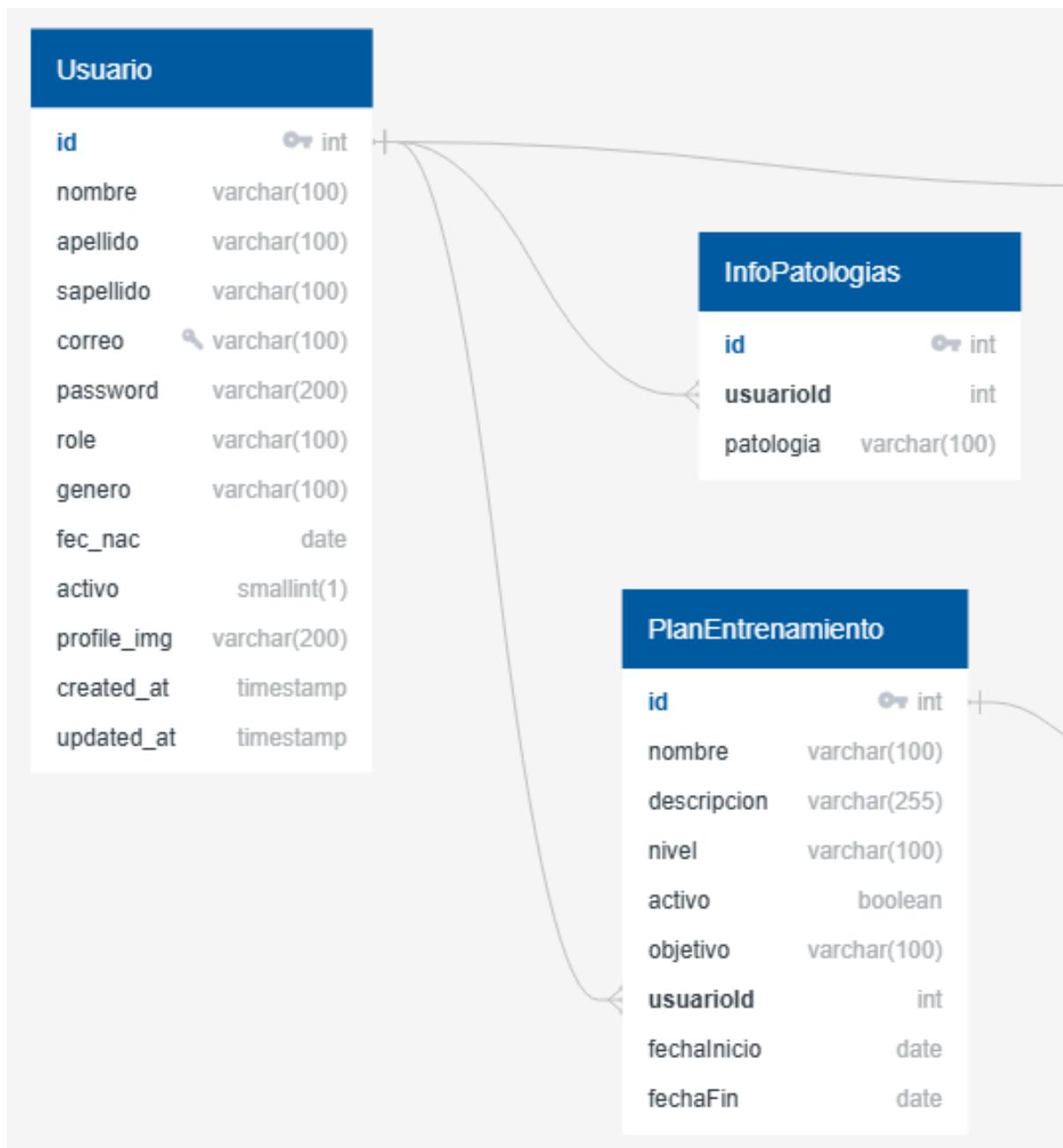


Diagrama Entidad-Relación. Parte 2:

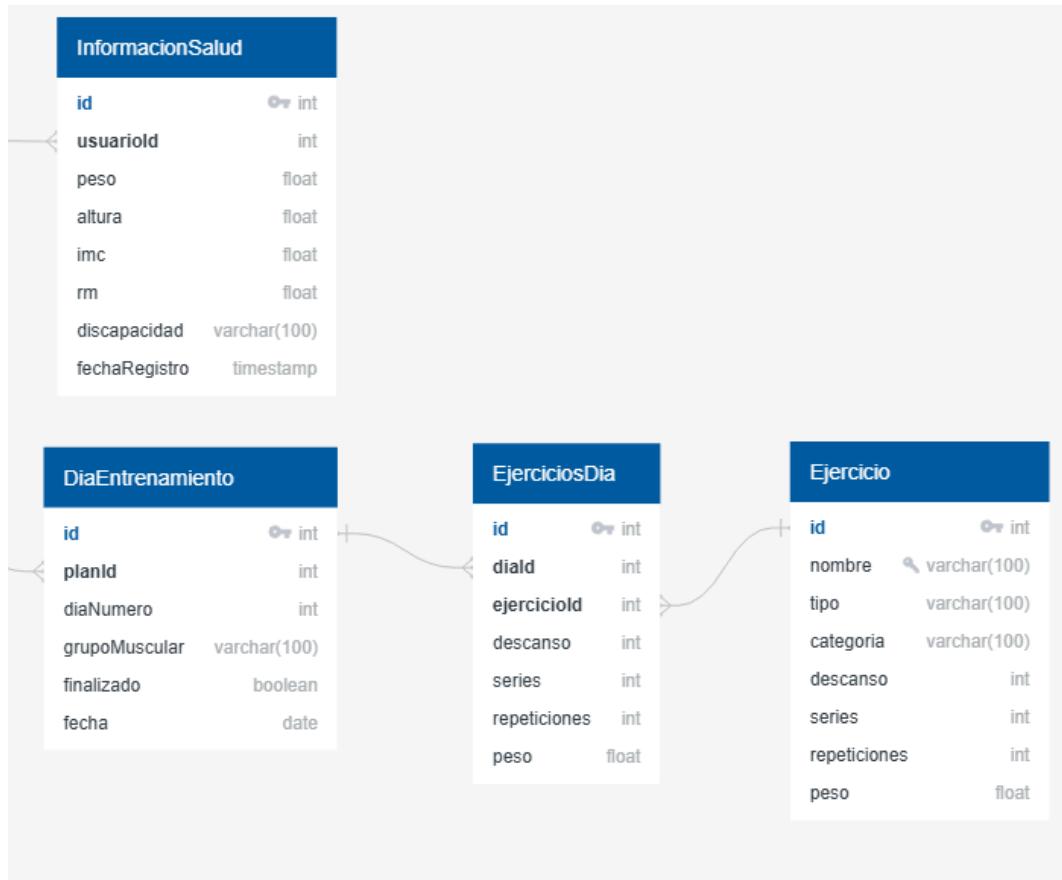
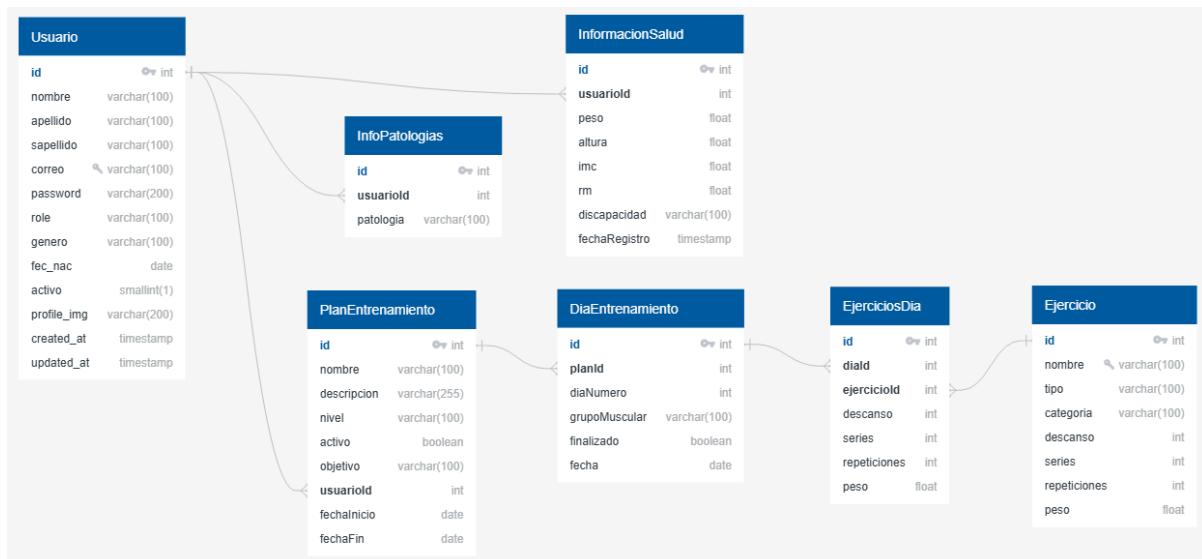


Diagrama Entidad-Relación. Vista General:



El diseño de la base de datos se ha realizado priorizando la coherencia, la integridad de los datos y la escalabilidad del sistema. Se han definido relaciones

adecuadas entre los modelos buscan reflejar de forma precisa la lógica del dominio, facilitando la consulta y manipulación eficiente de la información.

Asimismo, se ha optado por una estructura normalizada que permite evitar redundancias, garantizando la consistencia y simplificando el mantenimiento a largo plazo. Las decisiones se tomaron para responder a un enfoque orientado a la claridad y la adaptabilidad del modelo ante posibles futuras implementaciones del sistema.

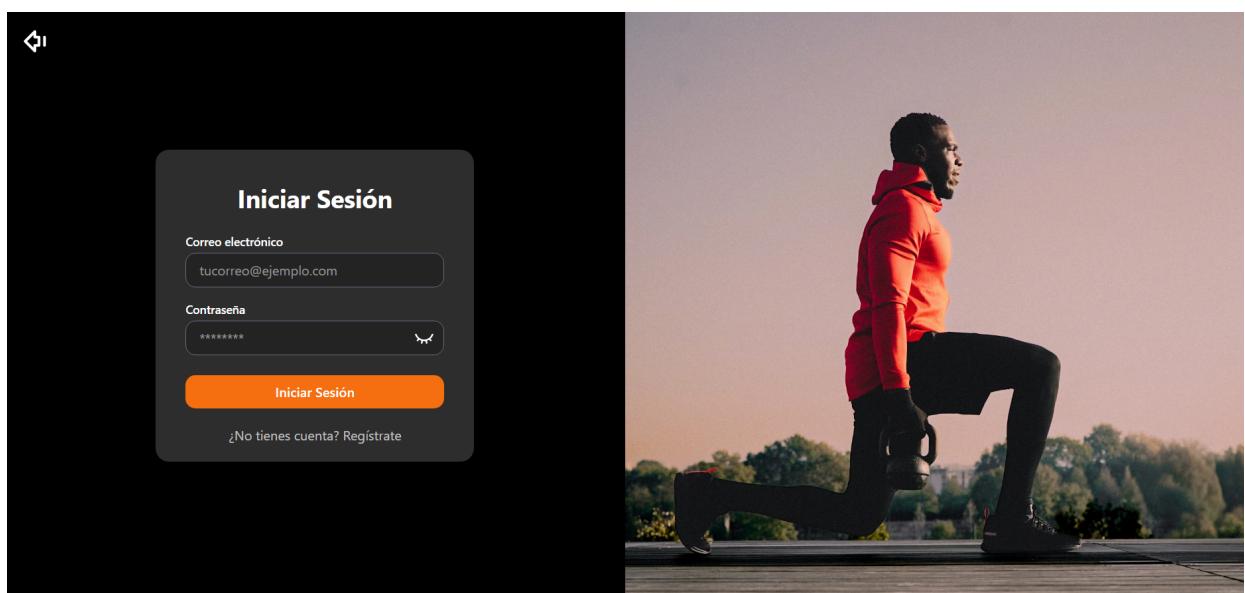
4.3 Diseño de la interfaz de usuario

Para el diseño de la interfaz del usuario se ha optado por una paleta de colores que refleje un equilibrio entre seriedad, claridad y dinamismo. Aportando un buen contraste entre los colores.



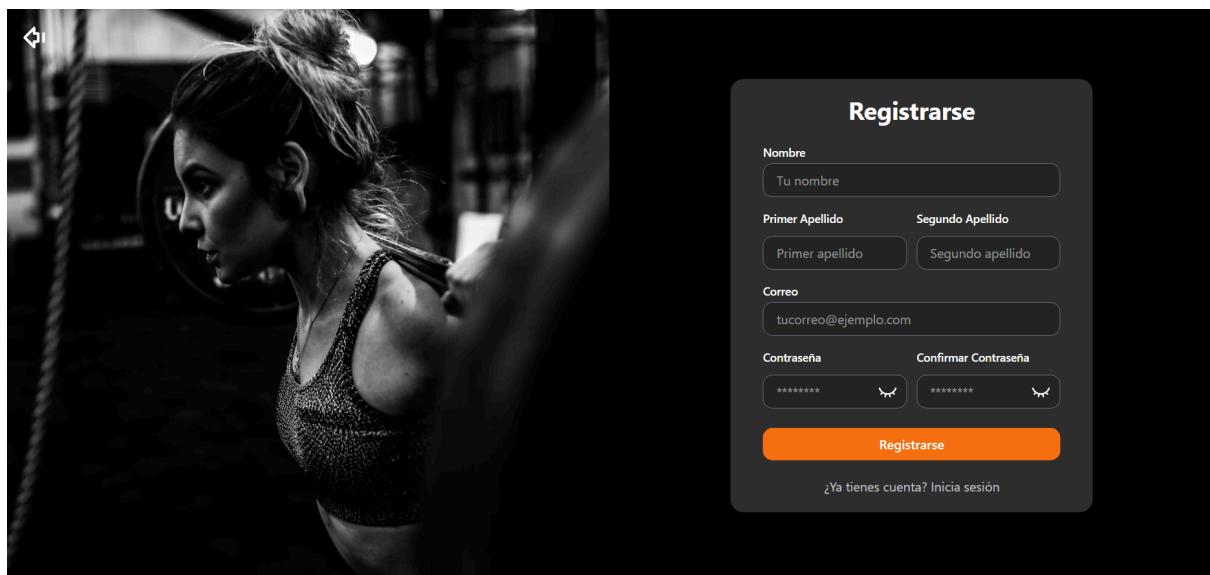
4.3.1 Vista Login

Para la vista de Login se pensó en una interfaz adaptable y minimalista con animaciones frescas que representen la finalidad de la aplicación, no se utiliza cabecera ni pie de página para centrar la atención del usuario en el objetivo de la vista que es el formulario de inicio de sesión con sus campos y su botón.



4.3.2 Vista Register

En este vista se sigue la misma interfaz haciendo que el sitio luzca exactamente igual con un formulario de registro con los campos necesarios para crear un usuario de manera básica.



4.3.3 Vista Perfil

La idea principal de esta vista es que cuando el usuario iniciará sesión y se dirigiera desde el dropdown a su perfil pudiera ver y consultar toda su información como IMC, datos personales y físicos, últimos días entrenados, rutinas y patologías.

Mi perfil

Datos físicos del usuario

- Altura: 1.78 cm
- Peso: 89 kg
- Género: Hombre

Últimos 7 días entrenados

- mar 03/06
- mié 04/06
- jue 05/06
- vie 06/06
- sáb 07/06
- dom 08/06
- lun 09/06

Rutinas del usuario

No tienes ninguna rutina registrada

Tu IMC: 28.09
Sobre peso

Patologías:

Copyright © 2025 FitForge | Política de privacidad

4.3.4 Vista Admin

El propósito de esta vista es mostrar una tabla con todos los usuarios y sus datos con múltiples opciones de administración como eliminar, editar y ver perfil.

Esta vista sólo es observable por aquellos usuarios que posean roles de administrador o entrenador. El rol Administrador tiene permisos para eliminar, editar y ver el perfil de cada usuario mientras que el rol de Entrenador solo tiene permisos para consultar la información de los usuarios.

ID	Foto	Nombre Completo	Correo	Rol	Activo	Acciones
11		Luisita Perez Perez	luisita@email.com	admin	Activo	
12		Carlos Gómez López	carlos.gomez@email.com	admin	Activo	
13		Valeria Torres Martinez	valeria.torres@email.com	admin	Activo	
14		Andrés Ramírez Suárez	andres.ramirez@email.com	admin	Activo	
15		Camila Morales García	camila.morales@email.com	admin	Activo	
16		Fernando Rojas Vega	fernando.rojas@email.com	admin	Activo	
17		Lucía Sánchez Navarro	lucia.sanchez@email.com	admin	Activo	

4.3.5 Vista Ejercicios

Los ejercicios que posee la aplicación se muestran en una vista que usa filas y columnas para servir los múltiples ejercicios hallados en la base de datos de forma responsiva. Estos ejercicios muestran datos importantes como su tipo, categoría y medidas específicas para cada ejercicio como lo son las repeticiones, las series, el peso y el descanso entre cada serie.

La vista dependiendo si el rol del usuario es Administrador o Entrenador mostrará opciones distintas permitiendo la creación, edición y eliminación de ejercicios.

Copyright © 2025 FitForge | Política de privacidad

Incluido a esto la vista tiene un buscador para filtrar los ejercicios por nombre, categoría o incluso tipo.

4.3.6 Vista Rutinas

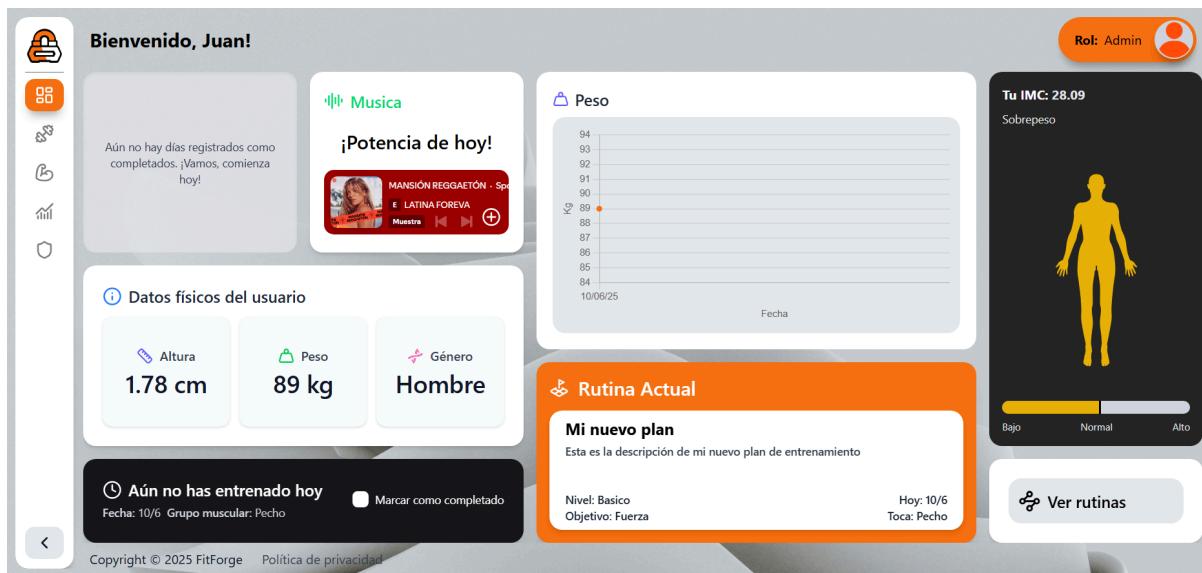
Esta vista se encarga de mostrar las rutinas que posee cada usuario mostrando de primero la más reciente. También se puede crear un plan de forma manual si el usuario posee el rol de Entrenador o de Administrador o en su caso crearlo de forma automática con solo introducir los datos que se te soliciten.

Copyright © 2025 FitForge | Política de privacidad

Las rutinas solo pueden ser editados y adaptados por Entrenadores y Administradores siendo también las mismas eliminables por los mismos usuarios que tengan dicho rol como el usuario que posee esta rutina.

4.3.7 Vista Dashboard

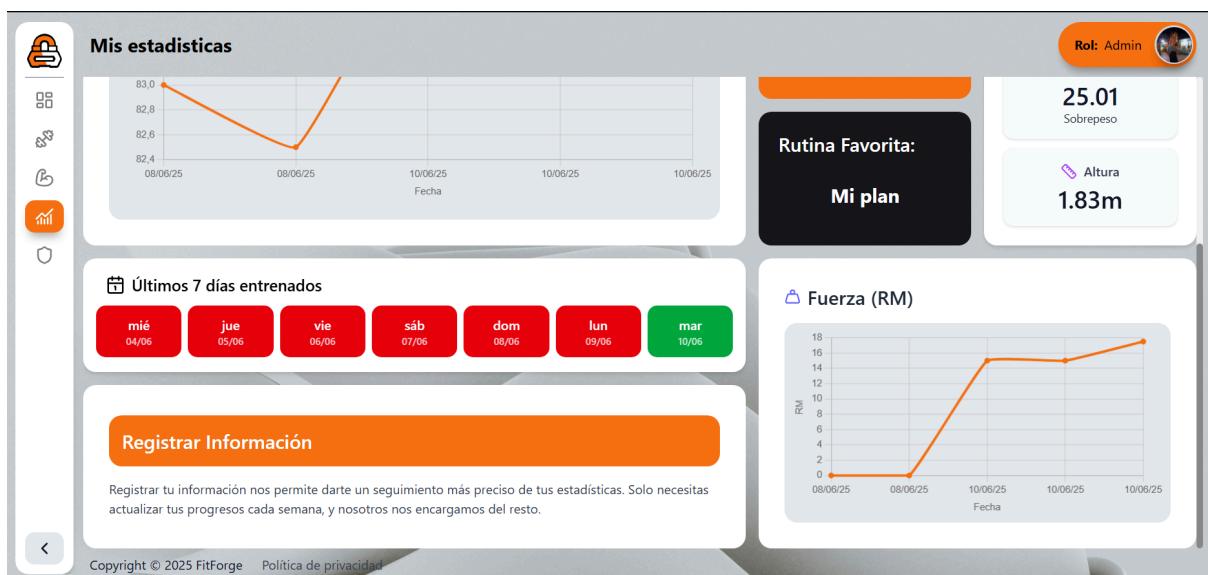
Esta es la vista principal donde el usuario llega tras iniciar sesión, en este vista se encuentra la información importante y rápida para el usuario como su IMC, su rutina actual su datos físicos, su progreso, los días entrenados y más.



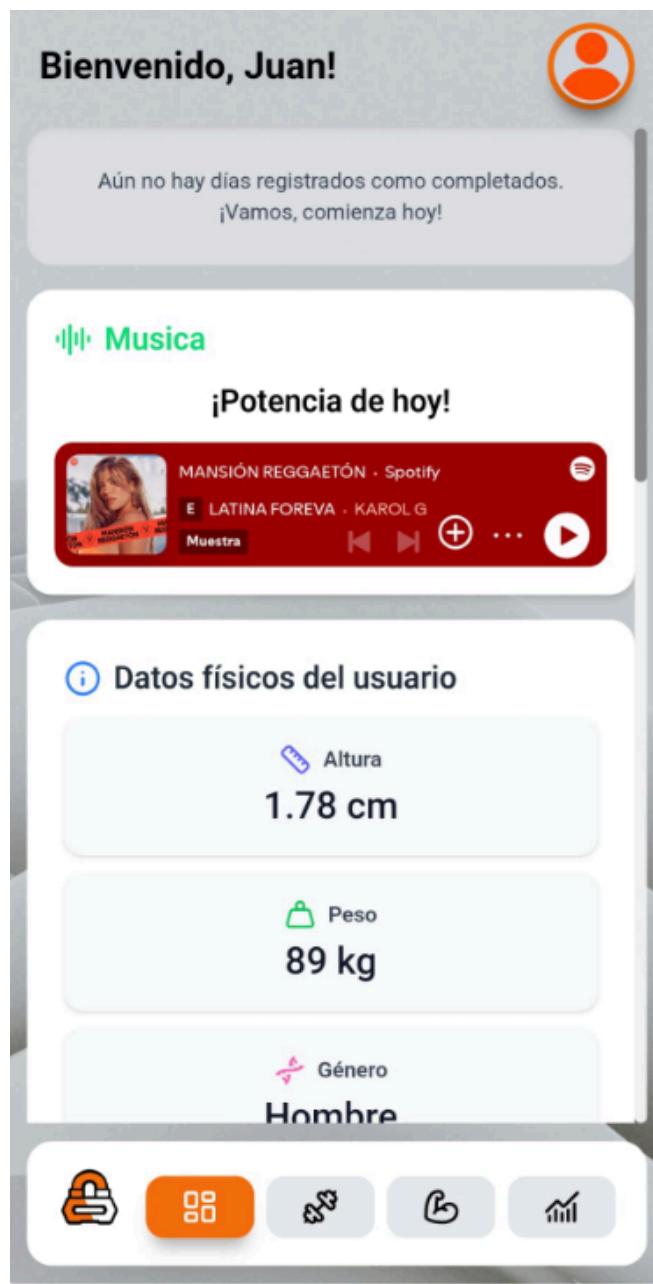
4.3.8 Vista Estadísticas

La vista de estadísticas permite al usuario consultar datos importantes como sus datos físicos e importantes actuales, cuántos días ha entrenado, cual es el grupo muscular más entrenado y el plan más entrenado.

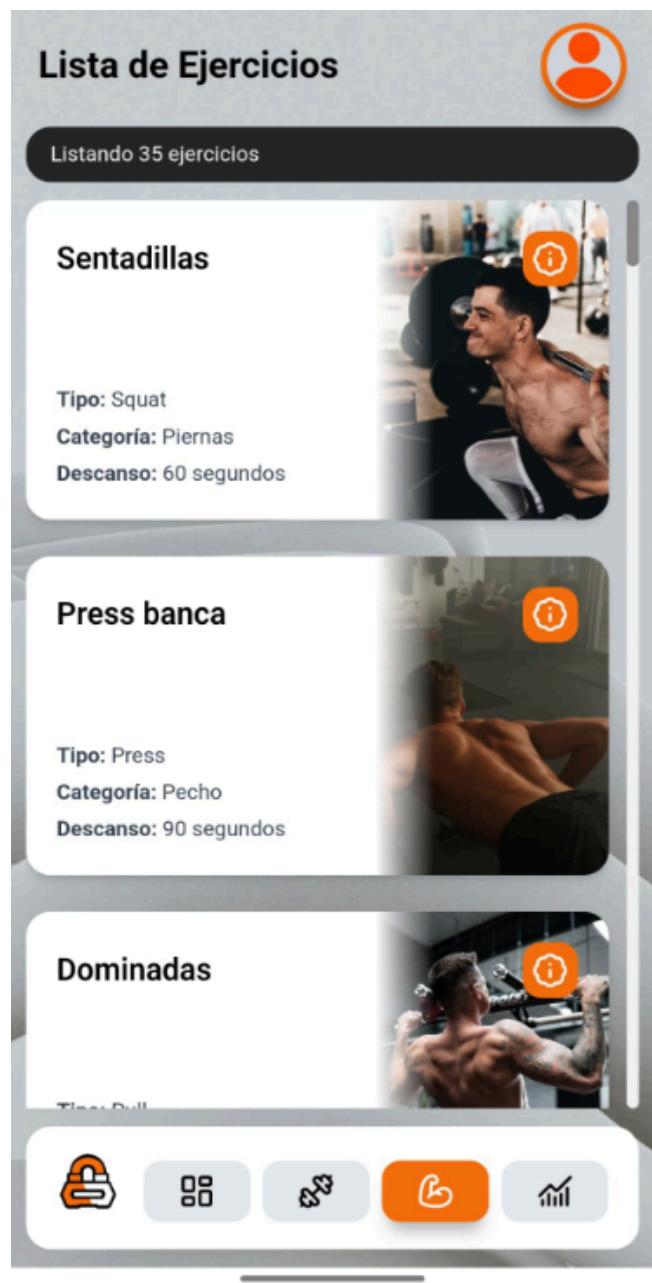
También se destaca en el informacion importante como los ultimos 7 dias entrenados, el apartado para realizar un registro y las gráficas del peso y de la repetición máxima del usuario a lo largo del tiempo.



Las vistas son totalmente adaptables a cualquier tamaño de pantalla adaptando el contenido para que sea visible incluso desde el móvil. A continuación se observan 2 vistas desde las proporciones de un teléfono móvil:



Desde móvil la vista del dashboard se ajusta distribuyendo el contenido de la misma en una sola columna con un scroll para deslizarnos por el resto de la vista, junto a ello también se transforma el layout distribuyendo la barra de navegación de forma horizontal mostrando las opciones de navegación justamente necesarias para la visualización desde el móvil. En la cabecera del dashboard también se ve que se oculta el rol del usuario mostrando solo la foto del mismo.



Por último la vista de ejercicios también muestra una distribución del contenido adaptada para el móvil donde los ejercicios se listan en una única columna ocultando la barra de filtrado.

Cabe destacar que se han incluido únicamente las vistas más representativas de la aplicación, seleccionadas en función de su relevancia. La elección de estas pantallas responde a la necesidad de mostrar el diseño, la estructura visual y la experiencia de usuario que se aplicará de forma coherente en el resto del sistema.

4.4 Tecnologías utilizadas

Para el desarrollo del proyecto se han empleado tecnologías actuales tanto en el frontend como en el backend, con el objetivo de garantizar un rendimiento óptimo, una buena experiencia de usuario y un mantenimiento estructurado del código. A continuación, se describen las más relevantes utilizadas en cada parte de la aplicación:

Backend:

- **Node.js + Express:** Se ha optado por Express como framework base sobre Node.js por su sencillez y flexibilidad en la creación de API REST.
- **Prisma:** ORM moderno que permite una interacción sencilla y tipada con bases de datos relacionales. Es utilizado para las migraciones, seeders consultar, actualizar, crear y eliminar datos.
- **Zod:** Librería de validación de datos que permite comprobar estructuras y tipos de los objetos que recibe el backend, se utiliza en validación de datos por peticiones HTTP.
- **bcryptjs:** Utilizado para encriptar contraseñas de manera segura antes de almacenarlas en la base de datos.
- **jsonwebtoken (JWT):** Se emplea para la autenticación mediante tokens, manteniendo sesiones seguras y sin necesidad de cookies.
- **Multer:** Permite gestionar la subida de archivos, como imágenes de perfil, desde el frontend al servidor.
- **dotenv:** Gestión de variables de entorno para configurar el proyecto sin exponer datos sensibles.

Frontend

- **Vue 3 + Composition API:** Framework principal para el desarrollo de interfaces. Se ha usado la API de composición para una mejor organización del código y reutilización de lógica.
- **Vite:** Herramienta moderna de construcción (build tool) que mejora notablemente los tiempos de carga y recarga en desarrollo.
- **Tailwind CSS:** Framework de estilos basado en clases utilitarias, que permite crear diseños modernos, responsivos y consistentes sin escribir CSS personalizado.
- **Pinia:** Sistema de gestión de estado oficial de Vue 3, que sustituye a Vuex y permite una integración más limpia y modular, utilizado para gestionar los estados de usuarios autenticados y de el layout del dashboard.
- **Vue Router:** Permite gestionar la navegación entre vistas de manera declarativa.
- **Vue Toastification:** Sistema para mostrar notificaciones visuales al usuario (como errores, éxitos, alertas).
- **Anime.js y AOS:** Librerías de animación que mejoran la experiencia visual del usuario al interactuar con la aplicación. Por ejemplo la animación encontrada en la vista de registro y login.
- **Chart.js (a través de vue-chartjs):** Utilizado para mostrar gráficas estadísticas del progreso del usuario como peso, rm.
- **axios:** Cliente HTTP utilizado para realizar peticiones al backend (GET, POST, PATCH, DELETE y etc).

- **html2pdf.js y jsPDF**: Herramientas para exportar partes del contenido de la app (como rutinas) a PDF.
- **Zod**: Librería de validación de datos que permite comprobar estructuras y tipos de los objetos que recibe el front, se utiliza en validación de formularios aplicando expresiones REGEX.
- **Lucide Icons (lucide-vue-next)**: Colección moderna de iconos SVG para una interfaz visual clara y atractiva.

5. Planteamiento e Implementación

5.1 Introducción

En este apartado se detalla el proceso de implementación de la aplicación, abordando las decisiones técnicas adoptadas así como los distintos componentes y vistas que se han desarrollado. Nos enfocaremos en la estructura del sistema, las tecnologías empleadas y la forma en que se ha llevado a cabo la construcción de las principales funcionalidades.

El objetivo es ofrecer una visión clara y ordenada del desarrollo realizado, desde la planificación hasta la ejecución práctica del proyecto.

5.2 Arquitectura de la aplicación

5.2.1 Frontend

Para la construcción del cliente se ha utilizado Vue 3, haciendo uso de la Composition API y una arquitectura modular basada en componentes reutilizables. La estructura del proyecto ha sido diseñada con el objetivo de facilitar la escalabilidad, el mantenimiento y la separación de responsabilidades.

El directorio principal *components* contiene aquellos componentes reutilizables que sirven como base para distintas vistas. Estos se agrupan en subdirectorios según su funcionalidad:

- **auths:** Contiene componentes relacionados con la autenticación y gestión de cuenta (como cambio de contraseña o preferencias de usuario).
- **basics:** Reúne componentes reutilizables para la visualización de datos, como gráficas de progreso, ventanas modales o notificaciones emergentes (Modals, Charts o Toast).
- **layouts:** Incluye los componentes de diseño que definen la estructura visual de la aplicación (como dashboardLayout.vue o authLayout.vue).

Las vistas completas se encuentran dentro del directorio views, organizadas por secciones y roles de usuario. Estas se encuentran agrupadas en los siguientes subdirectorios:

El subdirectorio Auth: Incluye las vistas de login y registro.

El subdirectorio Dashboard: Contiene las vistas principales tras el inicio de sesión, incluyendo secciones como perfil, estadísticas, configuración o la gestión de entrenamientos.

Dentro de dashboard/admin, se encuentran las vistas específicas para el usuario con rol de administrador.

Dentro de dashboard/exercises, se encuentran las vistas relacionadas con la gestión de ejercicios.

Dentro de dashboard/traines, se gestiona la creación y visualización de rutinas o planes de entrenamiento.

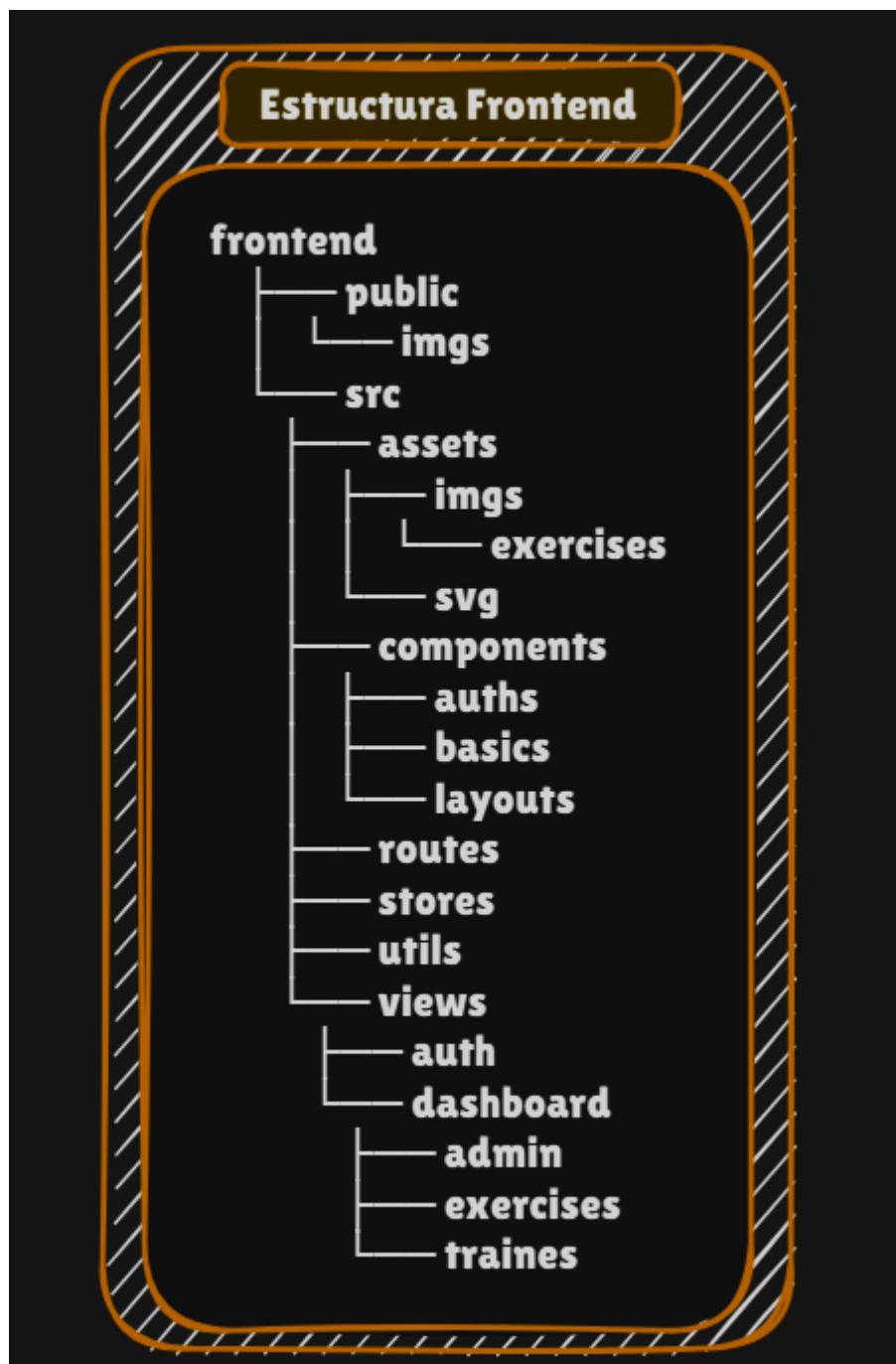
Raíz del subdirectorio Views, se encuentran las vista relacionadas con la pagina principal, estas representan las páginas Home, About Us, Private Policy y más.

Por otro lado, los recursos estáticos como imágenes y SVGs se encuentran almacenados dentro de la carpeta assets, organizados por tipos.

La lógica auxiliar se centraliza en los directorios stores (para el estado global como el layout) y utils (para funciones utilitarias como la autenticación).

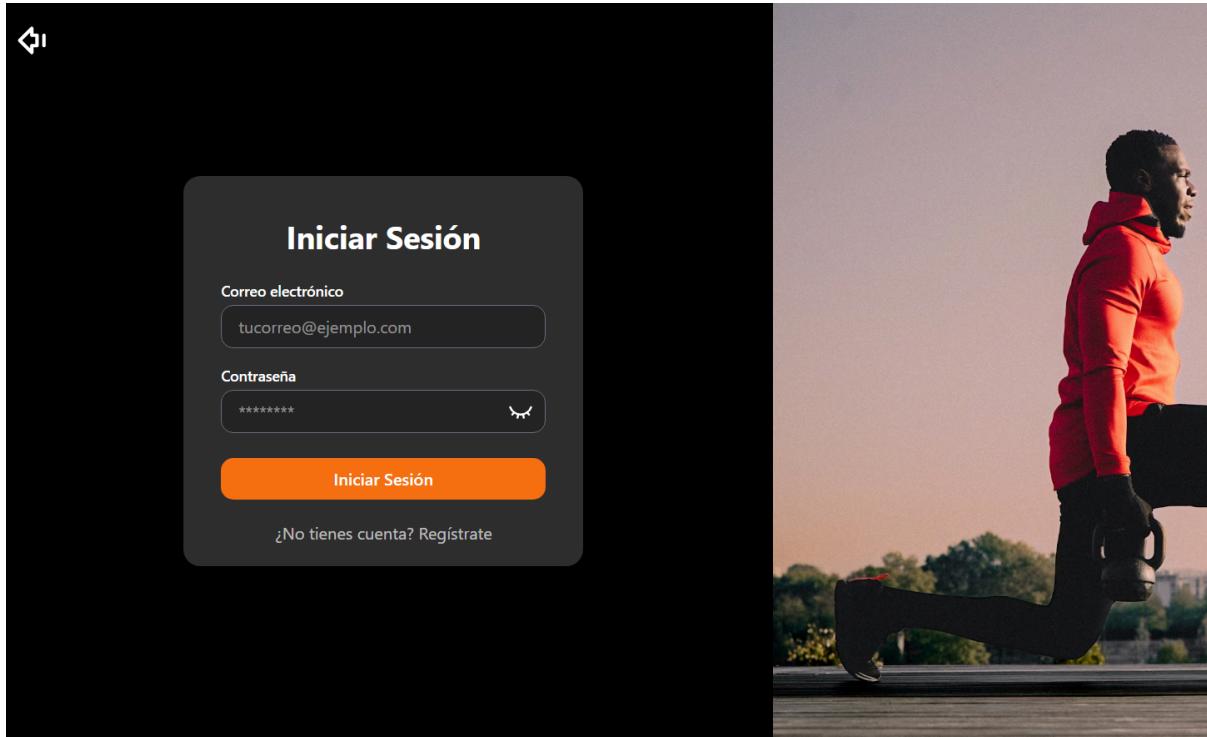
Finalmente, las rutas de la aplicación están definidas en el directorio routes, agrupadas por contexto (DashboardRoutes, ExercisesRoutes, etc.), lo cual facilita la modularidad del enrutamiento en Single Page Application.

Con esta estructura lo que se busca es conseguir una organización clara que facilite la escalabilidad del proyecto y el trabajo de forma colaborativa, asegurándonos que cada vista y componente cumpla un propósito bien definido dentro del sistema. El resultado final de la estructura para el Frontend de la aplicación es el siguiente:



5.2.1.1 Vistas:

Vista Login:

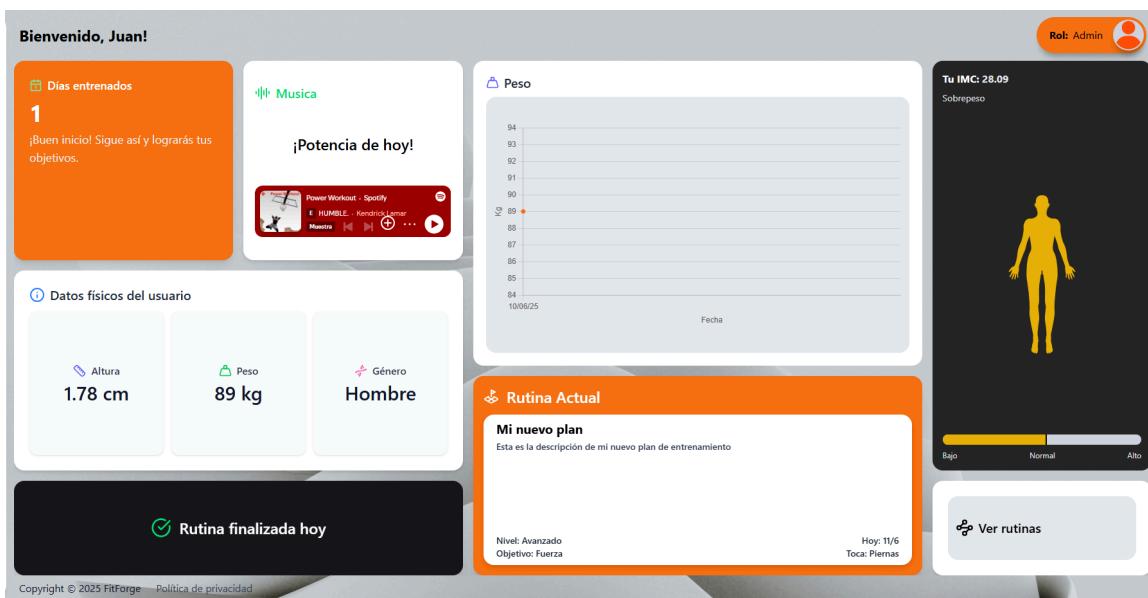


Funcionalidad principal: Haciendo una petición al backend permite al usuario iniciar sesión en la aplicación tras enviar sus datos como el correo electrónico con el que se registró y su contraseña, los campos de este formulario están validados permitiendo que aunque los campos no sean correctos se le avise mediante un mensaje de error al usuario la situación actual en la que se encuentra o el problema ocurrido.

Este componente tiene un diseño totalmente responsive estando completamente adaptado para cualquier pantalla. El rol o usuario al que está dirigido es a administradores, Entrenadores y Usuarios no autenticados .

Interacciones y elementos relevantes: Entre las cosas por destacar tenemos el enlace que nos permite realizar el cambio de página al registro del usuario, el botón que nos permite enviar el formulario, el botón para devolvernos a la página anterior y el Toast que nos indica el estado exitoso tras la respuesta correcta del backend al login.

Vista Dashboard:



Funcionalidad principal: Esta vista se encarga de hacer una promesa al backend con múltiples llamadas para consultar datos como el peso del usuario, su rutina actual, datos relevantes, indicarle al usuario si finalizó su entrenamiento hoy cuantos días ha entrenado y su IMC; Todo esto existe con la intención de poner al usuario al dia con información relevante para comenzar el dia. Esta vista es completamente disponible para todos los roles.

Interacciones y elementos relevantes: Esta vista tiene muchas interacciones como “confirmó día entrenamiento” que al confirmar el día de entrenamiento como su nombre lo indica hace una petición al backend por parte del usuario diciendo que el día de entrenamiento del plan que tenga activo lo ha finalizado.

Tenemos también el apartado de música que según el dia de la semana (Lunes a Domingo) recomienda al usuario una playlist del servicio de música Spotify para escuchar.

El apartado “Rutina actual” se encarga de tomar de la base de datos el plan de entrenamiento actual del usuario y mostrarle el grupo muscular que se entrenará hoy; al hacer click sobre él redirigirá al usuario a la vista completa del entrenamiento a su vez hay un apartado muy parecido que es ver rutinas que permite al usuario si

hace click sobre el llevarle a la página de rutinas para que el usuario pueda consultar todas las rutinas que posee. El último apartado a destacar es el contador de días entrenados que hace una petición consultando el total de días que ha entrenado el usuario y muestra un mensaje de motivación dependiendo de la suma total de días que tenga el usuario.

Vista Ejercicios:

Ejercicio	Tipo	Categoría	Descanso
Sentadillas	Squat	Piernas	60 segundos
Press banca	Press	Pecho	90 segundos
Dominadas	Pull	Espalda	90 segundos
Burpees	Hit	Full body	60 segundos
Curl de bíceps	Pull	Brazos	60 segundos
Planchas	Core	Abdomen	45 segundos
Peso muerto	Pull	Espalda	120 segundos
Press militar	Press	Hombros	90 segundos
Zancadas	Lunges	Piernas	60 segundos
Remo con barra	Pull	Espalda	90 segundos
Extensión de tríceps	Press	Brazos	60 segundos
Prensa de piernas	Squat	Piernas	90 segundos

Funcionalidad principal: En esta vista se hace una petición a la API para consultar todos los ejercicios encontrados en la tabla Ejercicios y los muestra distribuyéndolos en un GRID por columnas según el ancho de la pantalla, esta vista lo que buscar es servir de forma práctica y sencilla un vistazo previo a los usuarios los ejercicios disponibles y su ejecución mostrando datos como cuántas series, repeticiones, peso o tiempo de descanso en segundos son recomendados.

Como existe una gran cantidad de ejercicios se incluye un filtro con la intención de poder filtrar los entrenamientos por nombre, tipo o categoría. Al hacer clic en el botón de información de cada ejercicio veremos su otra información junto a un botón que permite al usuario con el rol de administrador o entrenador editar el ejercicio; al igual ocurre lo mismo con el botón de añadir que permite solamente a los entrenadores y administradores crear un nuevo ejercicio en caso de que no exista el que buscan..



Interacciones y elementos relevantes: Lo importante de la vista se centra en recopilar y filtrar los ejercicios para mostrar al usuario de manera atractiva visualmente los ejercicios incluidos en la página, se incluye también el botón que permite crear un nuevo ejercicio (**solo permitido para entrenadores y administradores**)

Entre elementos de interacción tenemos el botón de información que muestra información adicional del ejercicio al usuario dejando a la vista el botón de edición que no redirigirá a la vista para editar los valores de este ejercicio en caso de ser incorrectos.

Barra de filtrado: Para el filtrado se utiliza la función emitirFiltros() que es llamada cuando el usuario hace clic sobre el botón de filtrar, esta toma los valores del inputs del formulario de filtrado y según sus valores filtra los ejercicios. Al cargar la página lo primero que hace es consultar los ejercicios a la API y si la respuesta es correcta cargará los ejercicios en un array para mostrarlos en la vista, en el filtro es parecido, toma los mismos ejercicios solicitados y los guarda en una constante para preservarlos sin filtrar y el array de ejercicios se filtra según lo solicitado por el usuario, si el usuario hace clic en el botón de RESET vaciará el array con los ejercicios filtrados sustituyéndolos por los ejercicios impolutos guardados en la constante, esto se hace con la intención de mejorar el rendimiento de la API evitando hacer muchas consultas que afecten en sobrecarga o tiempos de respuestas muy largos para la API.

```

const filtros = ref({
  nombre: "",
  tipo: "",
  categoria: ""
});

const emitirFiltros = () => {
  exercises.value = originalData.value.filter((ej) => {
    return (
      (!filtros.value.nombre || ej.nombre.toLowerCase().includes(filtros.value.nombre.toLowerCase())) &&
      (!filtros.value.tipo || ej.tipo.toLowerCase() === filtros.value.tipo.toLowerCase()) &&
      (!filtros.value.categoría || ej.categoría.toLowerCase() === filtros.value.categoría.toLowerCase())
    );
  });
};

const resetFiltros = () => {
  filtros.value = {
    nombre: "",
    tipo: "",
    categoría: ""
  };
  exercises.value = [...originalData.value];
};

onMounted(async () => {
  emit("loading-start");

  try {
    const response = await axios.get(`${import.meta.env.VITE_API_URL}:${import.meta.env.VITE_API_PORT}/exercises/get`);
    originalData.value = response.data;
    exercises.value = response.data;
  } catch (error) {
    console.error("Error al cargar los ejercicios:", error);
  } finally {
    emit("loading-end");
  }
});

```

Vista Administración:

Lista de Usuarios						
Buscar por nombre, ID o correo	ID	Ascendente	Filtrar	Mostrando: 10	 Rol: Admin	
ID	Foto	Nombre Completo	Correo	Rol	Activo	Acciones
1		Juan Perez Perez	juan@email.com	admin	Activo	  
2		Ana Perez Perez	ana@email.com	admin	Activo	  
3		Jose Perez Perez	jose@gmail.com	admin	Activo	  
4		Roy Leeá Lué Morà	lunarmorner@gmail.com	admin	Activo	  
5		Javier Perez Perez	javier@email.com	admin	Activo	  
6		Pedro Perez Perez	pedro@email.com	admin	Activo	  
7		Maria Perez Perez	maria@email.com	admin	Activo	  
8		Luis Perez Perez	luis@email.com	admin	Activo	  
9		Luisa Perez Perez	luisa@email.com	admin	Activo	  
10		Luisito Perez Perez	luisito@email.com	admin	Activo	  

Funcionalidad principal:

Esta vista permite a los administradores consultar y gestionar toda la información clave de los usuarios en el sistema, incluyendo sus datos físicos, rutinas y últimos días de entrenamiento registrados. Al igual que otras vistas, realiza una llamada a la API para obtener los últimos datos de los usuarios. Su objetivo principal es brindar

una visión general de los usuarios y permitiendo acciones administrativas como el borrar usuarios, editar usuarios o consultar su perfil ofreciendo un correcto y apropiado funcionamiento del sistema. Cabe destacar que los entrenadores tienen acceso a esta vista pero con acciones limitadas, lo único que pueden hacer es consultar la información de cada usuario lo que les permite desarrollar planes de entrenamiento más personalizados.

Interacciones y elementos relevantes:

La vista incluye múltiples acciones interactivas como editar o eliminar usuarios o consultar el perfil de cada usuario teniendo la posibilidad de acceder rápidamente a otras secciones de administración cómo crear una rutina para dicho usuario o editar su rutina actual. También permite filtrar información por diferentes criterios (id, nombre, apellidos, etc.) a su vez se encuentra incluido el número de usuarios que deseamos tener por paginación.

Vista Ver Perfil:

Perfil de Roy Lee Lunar More

Datos físicos del usuario

- Altura: 1.83 cm
- Peso: 83.76 kg
- Género: Hombre

Últimos 7 días entrenados

- jue 05/06
- vie 06/06
- sáb 07/06
- dom 08/06
- lun 09/06
- mar 10/06
- mié 11/06

Rutinas del usuario

- Mi plan automático**: Activo, Nivel: Avanzado, Objetivo: Fuerza, Hoy: 11/6, Toca: Piernas
- Un nuevo plan**: Este es un plan con un objetivo personalizado para un usuario, + Nuevo Plan

Tu IMC: 25.01
Sobrepeso

Patologías:
No tiene patologías registradas

Roy Lee

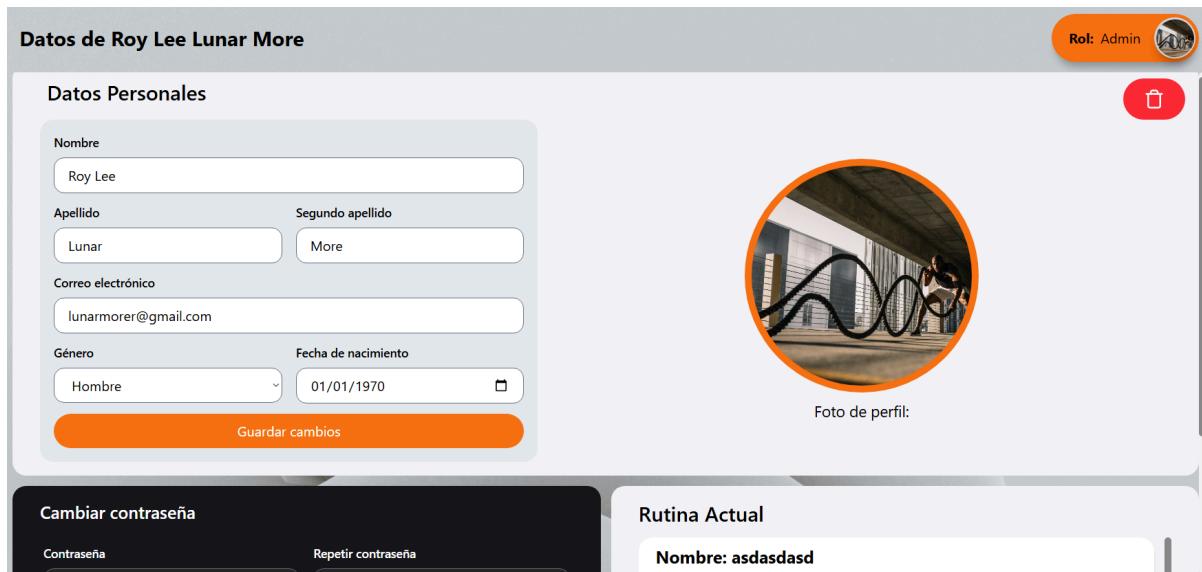
Roy Lee Lunar More
lunarmorer@gmail.com
Hombre
12 de julio de 2004

Se unió el: 7 de junio de 2025

Copyright © 2025 FitForge | Política de privacidad

Esta vista del panel de administración se carga al seleccionar el botón de ver perfil dentro

Vista Editar Perfil:



Funcionalidad principal:

Esta vista permite a los administradores editar la información personal de un usuario. El principal objetivo es proporcionar una interfaz intuitiva y completa para actualizar los datos del perfil de un usuario, ya sea para corregir errores o para arreglar algún plan de entrenamiento. Entre los datos que se pueden modificar se encuentran el nombre, apellidos, correo electrónico, fecha de nacimiento, género, peso, altura, objetivo y nivel. Además, desde esta misma vista se puede cambiar la contraseña del usuario, actualizar la foto de perfil o eliminar su cuenta. Esta funcionalidad es exclusiva del rol de administrador y representa un pilar en la administración de la página.

Interacciones y elementos relevantes:

La vista se encuentra organizada en múltiples formularios, entre ellos tenemos los siguientes: datos del usuario, credenciales de acceso y planes de entrenamiento. En la parte superior se muestra la imagen actual del usuario, con la opción de actualizarla subiendo una imagen de nuestro computador al servidor. Justo debajo, a su izquierda se encuentra el formulario principal donde todos los inputs están validados gracias a la librería Zod que permite validar formularios de muchas formas posibles, ofreciendo al administrador conocer en qué ha errado al introducir los datos.

Otro punto relevante que es importante destacar de la vista es el listado de rutinas, este permite al administrador acceder directamente a la rutina que le interesa del usuario o permitirle crear una nueva. Por último tenemos el botón de eliminar que permite al administrador eliminar desde esta misma vista el usuario que está observando.

Vista Plan:

Funcionalidad principal: Esta vista permite visualizar y gestionar un plan de entrenamiento específico, mostrando la información general del plan y sus días correspondientes. El usuario puede navegar por los días del plan de forma paginada, visualizar el detalle de los ejercicios asignados a cada día, y acceder a funciones como exportar el plan en PDF o editarlo. La información mostrada incluye nombre, descripción, objetivo, nivel y fechas del plan, así como los ejercicios con sus series, repeticiones, descanso y peso para cada día.

Elementos relevantes e interacciones: La interfaz está organizada en una estructura clara y funcional. En la parte superior se muestra el nombre y la descripción del plan, junto con botones para exportar el plan en formato PDF y para editarlo. Debajo, un sistema de paginación permite navegar entre los días del plan, mostrando hasta siete días por página, con controles para ir a la primera, anterior, siguiente y última página.

Al seleccionar un día, se muestra una lista de ejercicios programados para ese día, detallando nombre del ejercicio, series, repeticiones, descanso y peso utilizado. Si el día no contiene ejercicios, se indica que es un día de descanso. La navegación es intuitiva y visual, con indicadores claros para el día seleccionado mediante cambios de color en los botones.

Además, se manejan estados de carga y error para informar al usuario sobre la obtención de datos desde el backend. El componente se integra con el store de autenticación para enviar el token en las peticiones, y con el store de layout para actualizar dinámicamente el título de la página.

Vista Estadísticas:



Funcionalidad principal: Esta vista tiene como objetivo principal mostrar de forma visual y estructurada las métricas y estadísticas relevantes del usuario relacionadas con su progreso físico y deportivo. Entre estas métricas se incluyen el peso, la fuerza máxima (RM), el IMC, la altura, así como otras estadísticas como los días entrenados y hábitos de entrenamiento, el grupo muscular más trabajado y la rutina favorita.

Este dashboard permite al usuario obtener una visión global y actualizada de su estado y evolución a través de gráficos dinámicos, datos actuales y resúmenes

estadísticos. Además, facilita el acceso directo a la funcionalidad para registrar nueva información que permita mejorar el seguimiento del progreso.

Interacciones y elementos relevantes: El dashboard tiene muchos elementos relevantes que permiten al usuario visualizar y entender de manera práctica su progreso físico. En la sección de peso y fuerza máxima (RM), se muestran gráficos dinámicos que reflejan la evolución temporal de estas métricas, actualizándose de forma automática con los datos obtenidos desde la API. Esto facilita al usuario evaluar de manera visual los cambios en su condición física.

A su vez, se muestra información numérica relevante como el índice de masa corporal (IMC), altura y peso actual, acompañada de indicadores visuales que representan el estado o categoría correspondiente, ayudando a interpretar de forma rápida al usuario sus datos de salud general. Por último tenemos un botón con un enlace directo que facilita el acceso a la función de registrar nuevos datos del usuario, promoviendo la actualización continua de la información para un seguimiento efectivo y personalizado.

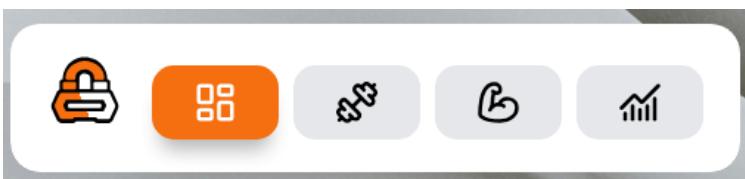
NOTA: Debido a la gran cantidad de vistas desarrolladas en la aplicación, no se incluye en este documento una descripción exhaustiva de cada una de ellas. En su lugar, se ha optado por detallar únicamente las vistas más relevantes, que representan las funcionalidades principales y el flujo esencial del sistema. Esta decisión permite centrar el análisis en los aspectos clave del proyecto, evitando una extensión excesiva que dificultaría la comprensión global y manteniendo un enfoque claro y coherente en la presentación del trabajo.

5.2.1.2 Componentes:

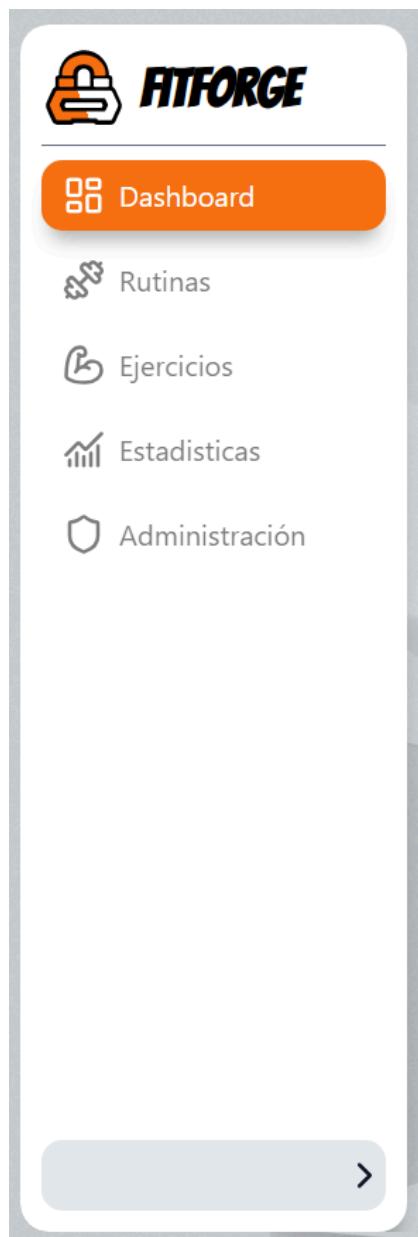
Durante el desarrollo del proyecto se han utilizado múltiples componentes con el objetivo de reutilizar fragmentos de código y evitar duplicaciones. Esta estrategia mejora la organización del código y facilita su mantenimiento.

Dado el elevado número de componentes implementados, en esta sección se describen únicamente los más representativos, aquellos que cumplen funciones clave dentro de la interfaz y aportan valor directo a la experiencia del usuario.

Cabecera Dashboard:



El componente de cabecera representa la barra lateral de navegación principal del dashboard de la aplicación. Su diseño está enfocado en la responsividad ofreciendo una experiencia de usuario fluida en dispositivos móviles, tablets y escritorios.



Funcionalidades principales:

Navegación principal:

El componente incluye un menú con accesos a las secciones más importantes de la aplicación, como: Dashboard (inicio), Rutinas, Ejercicios, Estadísticas y Panel de administración (solo está visible si el usuario tiene rol de administrador o entrenador)

Cambio entre vista expandida y contraída:

A través de un botón, el usuario puede expandir o contraer la barra lateral. En modo contraído, solo se muestran los iconos; al expandirse, aparecen también los nombres de las secciones.

Animaciones:

El componente incluye transiciones suaves (fade-slide) al mostrar/ocultar los textos de las secciones, proporcionando una experiencia visual grata al usuario.

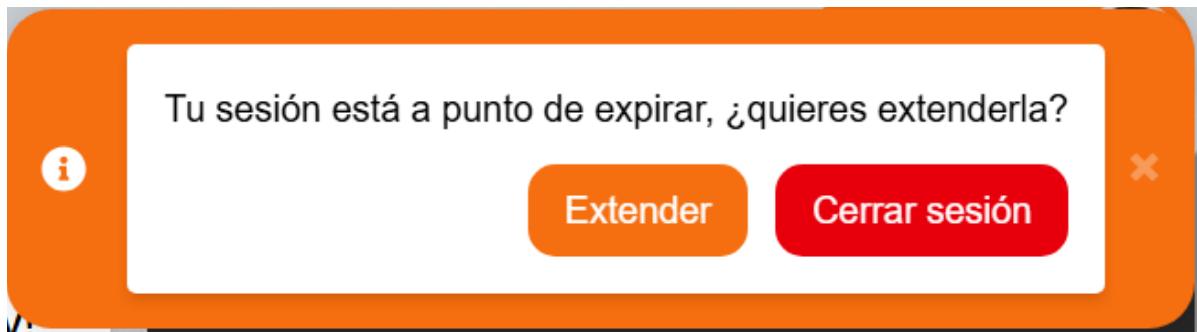
Ventana Modal:

Se ha desarrollado una ventana modal con el objetivo de mostrar información o formularios sin necesidad de cambiar de vista, facilitando así la interacción del usuario dentro de la aplicación.

Este componente aparece en el centro de la pantalla sobre un fondo transparente, y cuenta con animaciones suaves tanto al abrirse como al cerrarse, lo que mejora la UX. El modal incluye un botón para cerrarlo de forma manual o puede cerrarse la misma haciendo clic fuera de la ventana.

Una de sus principales ventajas es que está diseñado para ser reutilizable. Gracias al uso de la directiva `:is` de VUE, se puede cargar componentes dentro de la ventana permitiendo adaptar el mismo modal a distintos casos de uso dentro de la aplicación (como formularios, confirmaciones o vistas detalladas), sin duplicar código.

Toast Sesión:



Se ha creado un componente visual que informa al usuario cuando su sesión está a punto de expirar, ofreciéndole dos opciones: extenderla o cerrar sesión manualmente. Esta alerta es una notificación tipo toast, lo que permite mostrarse sin interrumpir completamente la navegación del usuario.

El diseño es simple mostrando un mensaje claro y dos botones. Al hacer clic en “Extender”, se solicita la API para que provea un nuevo token para el usuario. Lo mismo ocurre al seleccionar “Cerrar sesión” con la diferencia de que cerrará la sesión del usuario expulsandolo de la aplicación.

Esto se realiza usando la librería **vue-toastification** que permite crear, editar y gestionar múltiples Toast. Este tipo de aviso es útil para mejorar la seguridad y la experiencia del usuario, ya que le permite mantener su sesión activa de forma consciente.

5.2.1.3 Utilidades y Stores:

Auth: Este fichero define una store utilizando una librería llamada Pinia la cual es el sistema oficial de gestión de estado para Vue, este fichero se encarga de gestionar la lógica relacionada a la autenticación del usuario en el frontend.

Sus principales funcionalidades son:

- Verificar si el usuario ha iniciado sesión: Se comprueba si hay un token guardado en el almacenamiento del navegador y si es válido y no ha expirado con la finalidad de mantener al usuario autenticado.
- Guardar y eliminar el token: Permite iniciar sesión guardando el token (login) o cerrar sesión eliminándolo del almacenamiento del navegador (logout).
- Detectar si la sesión está por expirar: Si al usuario le quedan menos de 5 minutos de sesión, se le muestra un aviso con opciones para extender su sesión o cerrarla.
- Renovar el token automáticamente: Si el usuario elige extender la sesión, se hace una petición al backend para obtener un nuevo token sin necesidad de volver a iniciar sesión.
- Obtener información del usuario desde el token: Se extraen datos como el nombre, el rol, el ID o la imagen de perfil sin necesidad de hacer una petición extra al servidor.
- Comprobar el rol del usuario: Hay funciones que permiten saber si el usuario es administrador o entrenador según su rol.
- Validar el token: Se verifica si el token ha expirado o no, para asegurar que la sesión sigue activa.

LayoutStore: En este fichero se define una store que utilizará también la librería Pinia, este se encargará de manejar el estado relacionado con el título de la interfaz principal de la aplicación. Esta funcionalidad permite modificar dinámicamente el título mostrado en la parte superior de cada vista, lo cual es especialmente útil para mantener la coherencia visual en función de la ruta o el contenido mostrado.

5.2.2 Backend

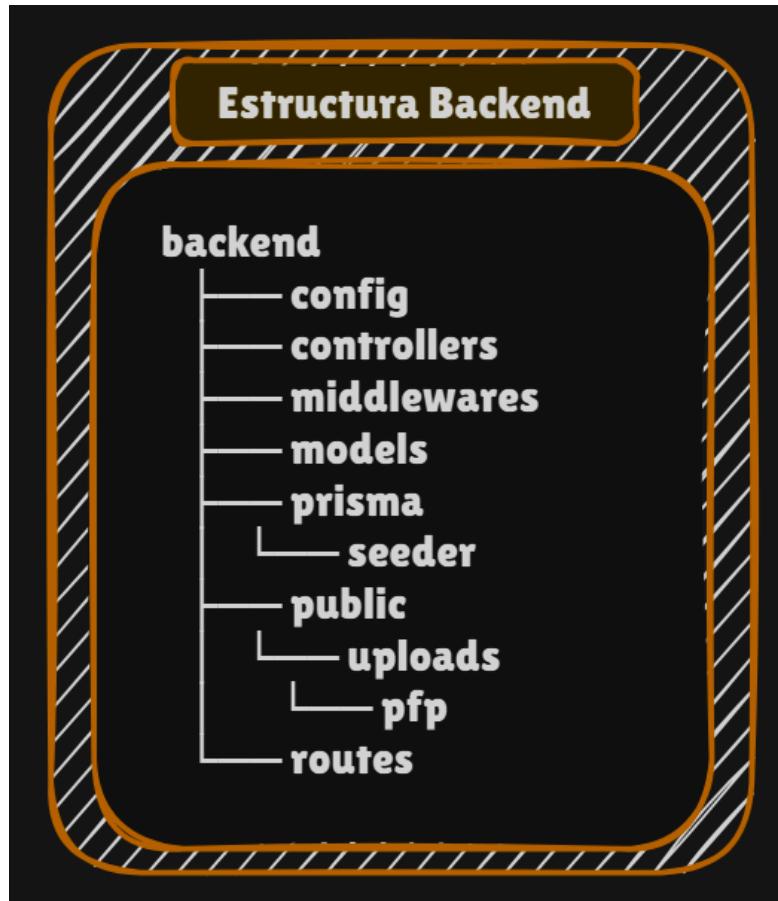
El backend de la aplicación ha sido desarrollado utilizando Node.js junto con el framework Express, siguiendo una arquitectura basada en el patrón MVC (Modelo–Vista–Controlador), que permite separar el trabajo de forma clara facilitando el mantenimiento, la escalabilidad y la organización del código.

La estructura del backend se encuentra organizada en diferentes directorios, cada uno tiene un propósito específico:

- **config:** Este directorio se encarga de contener los archivos de configuración general del servidor y de servicios externos, en mi caso puede encontrarse aquí la configuración del cliente de Prisma.
- **controllers:** Contiene todos los controladores de la aplicación, que se encargan de procesar las peticiones entrantes, aplicar la lógica de negocio necesaria y devolver las respuestas correspondientes a las solicitudes del usuario. Estos se encuentran organizados por contexto funcional, como autenticación, usuarios, entrenamientos, ejercicios, etc.
- **middlewares:** Este directorio se encarga de agrupar funciones intermedias reutilizables que se ejecutarán antes de llegar al controlador para proteger el back de peticiones que no se tienen acceso o que incluyen datos que pueden vulnerar la seguridad de nuestro servidor backend. Se incluyen middlewares para autenticación, validación de datos y autorización basada en roles en la aplicación.

- **models:** Este directorio contiene los esquemas de validación definidos con Zod, que se utilizan para validar los datos recibidos del cliente antes de ser procesados o almacenados. Estos modelos representan la estructura esperada de entidades como usuarios, planes o ejercicios en toda la aplicación.
- **prisma:** Este directorio incluye el archivo schema.prisma, que define la estructura de la base de datos. Además, cuenta con una subcarpeta seeder que se encarga de contener scripts para la carga inicial de datos, como ejercicios predefinidos o usuarios para facilitar el desarrollo e implementación de nuevas funcionalidades en un entorno de prueba sin afectar a la base de datos de producción.
- **public/uploads/pfp/:** Esta carpeta se utiliza para almacenar archivos públicos subidos por los usuarios, especialmente imágenes de perfil (pfp). El acceso a esta carpeta está configurado de forma segura para permitir la visualización de los recursos desde el cliente sin afectar la seguridad del sistema.
- **routes:** Contiene los archivos donde se realiza la definición de rutas del servidor. Las rutas están organizadas por grupos funcionales (como usuario, entrenamiento, ejercicios, etc.) y hacen uso de express.Router() para modularizar la estructura del backend. Cada archivo de ruta importa su respectivo controlador y middlewares necesarios para proteger o validar las peticiones siguiendo con las buenas prácticas del patrón **MVC**.

Esta organización busca garantizar una separación clara de responsabilidades facilitando el trabajo colaborativo, permitiendo que cada archivo y carpeta cumpla una función bien definida dentro del sistema. Además, este diseño favorece mucho la escalabilidad, ya que cada módulo puede ser extendido o modificado sin afectar al resto de la aplicación.



5.2.2.1 Controladores:

Con el fin de favorecer la mantenibilidad y la separación de responsabilidades, los controladores se han dividido en diferentes archivos, cada uno especializado en un ámbito funcional de la aplicación:

AuthController: Este controlador se encarga de gestionar toda la lógica relacionada con la autenticación de usuarios. Se encarga del registro, inicio de sesión, actualización de credenciales, y emisión de tokens JWT para el acceso a rutas protegidas.

Además utiliza **bcryptjs** para cifrar contraseñas y **jsonwebtoken (JWT)** para generar tokens JWT seguros de sesión. Los métodos que posee este controlador son los siguientes:

- Register: Este método permite registrar nuevos usuarios. Primero verifica si ya existe un usuario con el correo proporcionado y si las contraseñas coinciden. Si todo es correcto, la contraseña se cifra y se almacena en la base de datos. En caso de que el usuario ya exista pero esté inactivo, su cuenta se reactiva. Finalmente, genera un token JWT que se devuelve al cliente.
- Login: Se encarga del inicio de sesión. Busca un usuario con el correo recibido y comprueba que esté activo. Luego compara la contraseña proporcionada con la almacenada (ya cifrada). Si las credenciales son válidas, se genera un nuevo token JWT con información básica del usuario, que se enviará al cliente para mantener la sesión.
- RefreshToken: Esta función permite renovar el token JWT cuando está próximo a expirar. Recibe el token anterior, lo verifica y, si es válido, genera uno nuevo con la misma información del usuario. Este mecanismo permite extender sesiones sin que el usuario tenga que volver a iniciar sesión manualmente.

Cabe destacar que el token JWT (JSON Web Token) es un mecanismo fundamental en la autenticación de la aplicación, ya que este permite al backend verificar la identidad del usuario en cada solicitud protegida. Este token se incluye generalmente en los encabezados HTTP y es firmado digitalmente por una clave que posee el servidor, lo que impide su alteración por terceros. Gracias a ello, se evita el uso de cookies tradicionales y se mitigan ataques como **CSRF (Cross-Site Request Forgery)**, ya que las solicitudes deben incluir un token válido y firmado. Además, el JWT contiene información mínima (como el ID y rol del usuario), lo que garantiza eficiencia y seguridad en las sesiones sin afectar el rendimiento.

A continuación tenemos el proceso que realiza el método (login) para validar usuarios y devolver un token firmado.

```

static async login(req, res) {
  const { correo, password } = req.body;

  try {
    const user = await prisma.usuario.findUnique({ where: { correo } });
    if (!user) return res.status(404).json({ error: "Credenciales invalidas" });
    if (!user.activo) return res.status(404).json({ error: "Credenciales invalidas" });

    const isPasswordValid = bcrypt.compareSync(password, user.password);
    if (!isPasswordValid) return res.status(401).json({ error: "Contraseña incorrecta" });

    const data = {
      id: user.id,
      nombre: user.nombre,
      correo: user.correo,
      profile_img: user.profile_img,
      role: user.role
    };

    const token = jwt.sign(data, SECRET, { expiresIn: "2h" });

    res.json({ message: "Login exitoso", token });
  } catch (error) {
    res.status(500).json({ error: "Error interno" });
    console.log(error);
  }
}

```

Como se observa se comprueban los datos ingresados para ver si coinciden con los de la base de datos y se construye un objeto con los datos que queremos que contenga nuestro token y con la función (sign) se firma el token proporcionando los datos a contener, la clave con la que se firma el token y su tiempo de expiración, luego se le devuelve al usuario para que el usuario con este token pueda realizar sus peticiones a la API.

UsersController: El UserController centraliza la gestión de los usuarios dentro del sistema, permitiendo las operaciones básicas CRUD (crear, leer, actualizar y eliminar) tanto a nivel general como individual. Este controlador interactúa con el modelo UserModel y se encarga de validar datos, manejar imágenes de perfil y actualizar tokens JWT cuando sea necesario. Los métodos disponibles son los siguientes:

- GetAll: Recupera a todos los usuarios del sistema. Esta función está pensada para vistas administrativas donde se requiere acceder a la totalidad de registros sin paginación.

- GetById: Obtiene la información detallada de un usuario específico a través de su id. Es útil para mostrar detalles de un perfil o cargar información previa en formularios de edición.
- GetPaginate: Implementa un sistema de paginación configurable mediante los queryParams page, limit, query, sortField y sortDirection. Devuelve una lista de usuarios filtrada y ordenada, junto con información como el número total de páginas y elementos.
- Create: Permite crear un nuevo usuario desde el backend. Aunque no se utiliza para el registro habitual (ya que este se gestiona desde AuthController), es útil en contextos donde un administrador necesita dar de alta usuarios manualmente.
- Update: Actualiza los datos de un usuario existente permitiendo modificar información general, manejar cambios de contraseña (verificando coincidencia), y gestionar el reemplazo de la imagen de perfil (eliminando la antigua si no es la predeterminada). Si el usuario está actualizando su propio perfil, generará de forma automática un nuevo token JWT con los datos actualizados para asegurar que los datos entre el frontend y el backend sean coherentes entre sí.
- Delete: Este método se encarga de realizar un borrado lógico del usuario modificando su estado a activo: 0. Esta práctica evita la pérdida permanente de datos y mantiene la integridad referencial en la base de datos que es completamente utilizada en aplicaciones modernas.

EjerciciosController: Éste controller se encarga de gestionar todas las operaciones relacionadas con los ejercicios disponibles en el sistema. A través de este controlador se pueden consultar, crear, modificar y eliminar ejercicios, garantizando que los datos estén validados y estructurados correctamente antes de su persistencia en la base de datos. Este controlador interactúa directamente con EjercicioModel y está diseñado para ser utilizado tanto en vistas administrativas

como en procesos automáticos de creación de planes de entrenamiento. A continuación, se detallan sus métodos:

- GetById: Recupera un ejercicio individual a partir de su id, enviado como parámetro en la ruta. Devuelve los detalles del ejercicio correspondiente en formato JSON. Es utilizado para cargar los datos en formularios de edición o vistas detalladas.
- GetAll: Obtiene la lista completa de ejercicios registrados en el sistema, sin filtros ni paginación. Es útil para poblaciones iniciales de formularios, filtros o catálogos de ejercicios.
- Create: Crea un nuevo ejercicio a partir de los datos enviados en el cuerpo de la petición. Antes de la inserción, se comprueba que no exista un ejercicio con el mismo nombre mediante una búsqueda por nombre.
- Update: Actualiza los datos de un ejercicio existente identificado por su id. Se permite la modificación de todos los campos principales (nombre, tipo, categoría, descanso, series, repeticiones y peso).
- Delete: Elimina un ejercicio a partir de su id. La operación se realiza directamente sin implementar un borrado lógico, eliminando el registro de forma permanente de la base de datos.

EntrenamientoController: Este controlador es el más importante de todos porque es el responsable de toda la lógica de los planes de entrenamiento y su gestión, incluyendo su creación, recuperación, actualización y eliminación. Además, provee funciones específicas para seguimiento diario y generación automática de rutinas según los objetivos y nivel del usuario. El controlador interactúa principalmente con los modelos PlanEntrenamientoModel y DiaEntrenamiento.

Métodos principales:

- CreatePlan: Este método es el encargado de crear un plan de entrenamiento de forma manual con los datos introducidos desde el cliente (nombre, descripción, objetivo, nivel, fechas y usuario). Asociará Ejercicios a días y días de entrenamiento a un plan encontrándose todo completamente relacionados entre sí en la base de datos.
- FinalizarDia: Esta función se llama cuando el usuario ha finalizado un dia de entrenamiento de su plan actualizando el dia mismo al estado finalizado.
- GetDiasEntrenados: Recupera todos los días ya entrenados dentro de todos los planes, ayudando a visualizar cuantas sesiones de entrenamiento ha realizado al final el usuario.
- CreateAutoPlan: Se genera automáticamente un plan de entrenamiento completo para las próximas 4 semanas. Considerando género, nivel, IMC o fuerza (RM), se seleccionan grupos musculares según el día de la semana y luego los ejercicios por dicho grupo muscular ajustando las repeticiones, series y descansos según el objetivo y los datos del usuario facilitando ejercicios de forma automática y personalizada. Además, desactiva todos aquellos planes automáticamente previos que se encuentren activos del mismo usuario.
- GetPlanById: Consulta un plan específico usando su id. Devuelve todos los detalles incluyendo días y ejercicios asociados.
- GetPlanesPorUsuario: Permite obtener todos los planes de entrenamiento asignados a un usuario, facilitando la visualización en vistas personales o administrativas.
- Update: Actualiza un plan existente (incluyendo sus días y ejercicios) con los datos enviados desde el cliente.
- Delete: Este método es el encargado de eliminar de forma permanente un plan de entrenamiento por su id.

- GetDiasEntrenadosUltimos7: Devuelve un array de los últimos siete días indicando si el usuario entrenó en cada fecha.
- TogglePlan: Permite cambiar el campo “activo” de un plan (activarlo o desactivarlo), permitiendo a todos los planes de un usuario desactivarlos si activa otro.

ImageController: El controlador de imágenes (ImageController) es el encargado de gestionar el acceso a los archivos multimedia estáticos almacenados en el servidor, principalmente imágenes asociadas a las fotos de perfil de los usuarios dentro de la aplicación. Dentro del controlador, hay un único método que es el más importante, este es `getImage`, el cual permite recuperar una imagen concreta a partir de su nombre de archivo.

Este método funciona recibiendo como dato en la request el nombre del archivo que se está buscando (por ejemplo, `default.png`). A partir de este nombre, se construye una ruta absoluta, que concatenan de forma segura la ruta absoluta donde se encuentran alojadas las imágenes, en este caso es `public/uploads/pfp`, controlando el acceso indebido a otras rutas del servidor.

Una vez construida la ruta completa, se verifica que la imagen realmente existe, si el archivo existe, se procede a enviarlo directamente al cliente mediante la función `res.sendFile`, para entregar la imagen como la respuesta HTTP. Esto permite que la imagen se pueda usar en el navegador como recurso en una vista desde VUE (por ejemplo, dentro de una etiqueta `` en HTML).

En el caso de que no exista la imagen se produce un error durante el proceso y se devuelve una respuesta al front con dicho resultado. A continuación podemos ver la estructura de dicho método:

```
static async getImage(req, res) {
  try {
    const filename = req.params.filename;
    const filepath = path.join(imgFolder, filename);

    if (!fs.existsSync(filepath)) {
      return res.status(404).send("Imagen no encontrada");
    }

    res.sendFile(filepath);
  } catch (error) {
    console.error("Error al actualizar la imagen:", error);
    return res.status(500).send("Error al subir la imagen");
  }
}
```

InfoController: Este controlador se encarga de gestionar toda la información relacionada con la salud y condición física del usuario, incluyendo métricas corporales, estadísticas, y patologías registradas. Su principal función es acceder a los datos para que puedan ser visualizados o actualizados desde el frontend (VUE) sin afectar la seguridad del servidor, especialmente en secciones como el perfil o el dashboard.

El controlador interactúa principalmente con los modelos InformacionSaludModel, InfoPatologiaModel y UserModel. Los métodos disponibles en el controlador son los siguientes:

- **GetLast:** Toma el registro más reciente de información de salud correspondiente a un usuario. Permitendonos mostrar los últimos datos actualizados como peso, altura o IMC en vistas principales como el perfil o el panel de control.
- **GetAll:** Devuelve el historial completo de salud del usuario, permitiendo analizar la evolución del mismo a lo largo del tiempo. Esta información puede ser utilizada para generar gráficas o informes.
- **GetPatologias:** Con este método obtenemos la lista de patologías asociadas a un usuario específico. Es útil en la visualización detallada del estado de

salud.

- **GetMetricas:** Este método obtiene las métricas corporales más recientes del usuario, incluyendo valores como peso, altura o IMC. Los resultados son utilizados en la vista de estadísticas para demostrar el progreso en charts y en campos específicos de la vista.
- **GetEstadisticas:** Proporciona estadísticas generales del usuario a partir de los datos registrados tras hacer consultas a la base de datos que devuelven datos como grupo muscular más entrenado, rutina con más días entrenados, la suma total de todos los días entrenados y más.
- **UpdateInfo:** Permite actualizar la información del perfil de salud y físico del usuario incluyendo la modificación de datos personales (como fecha de nacimiento o género), actualización de métricas (peso, altura, IMC, etc.) y actualización de patologías. También realiza comprobaciones para evitar duplicidades y eliminar aquellas patologías que el usuario ya no tiene registradas para evitar registros duplicados.

5.2.2.2 Modelos:

Con el objetivo en mente de garantizar una estructura clara, escalable y alineada, los modelos han sido organizados representando cada uno una entidad principal del dominio de la aplicación. Estos modelos son los encargados de encapsular la lógica de acceso y manipulación de datos asociados a dicha tabla en la base de datos facilitando la interacción con la base de datos a través de Prisma. Los modelos que hemos definido para nuestra API son los siguientes:

UserModel: Este modelo se encarga de gestionar todas las operaciones relacionadas con los usuarios del sistema. A través de los métodos definidos, permite realizar consultas, crear, actualizar, eliminar y paginar los registros de usuario en la tabla de la base de datos con el mismo nombre.

Acciones principales:

- Obtener todos los usuarios o uno por ID.
- Buscar y paginar resultados mediante filtros.
- Crear nuevos registros.
- Actualizar campos específicos, incluyendo el cifrado de contraseñas.
- Eliminar usuarios.
- Contar resultados para facilitar la paginación.

InformacionSaludModel: Este modelo gestiona los datos relacionados con la salud del usuario, permitiendo almacenar y recuperar información física que se registra a lo largo del tiempo. Este modelo se encuentra relacionado con la tabla informacionSalud de la base de datos.

Acciones principales:

- Registrar nueva información de salud para un usuario.
- Obtener la primera, última o una entrada específica de información según el ID del usuario y el tipo de consulta solicitada.
- Eliminar una entrada de información según el ID proporcionado

InfoPatologiaModel: Este modelo es parecido al anterior solo que este permite gestionar la información sobre patologías asociadas a cada usuario de la tabla InfoPatologias en la base de datos.

Acciones principales:

- Consultar todas las patologías asociadas a un usuario.
- Registrar una nueva patología.
- Eliminar una patología específica mediante su ID.

PlanEntrenamientoModel: Este modelo es el encargado de gestionar toda la lógica relacionada a los planes de entrenamiento de los usuarios. A través de sus métodos, permite crear, consultar, actualizar y eliminar planes, así como realizar operaciones específicas como finalizar días, activar planes, obtener métricas recientes y generar estadísticas personalizadas.

Acciones principales:

- Crear planes personalizados, incluyendo días de entrenamiento y sus respectivos ejercicios.
- Consultar planes existentes, tanto por ID como todos los de un usuario, incluyendo su estructura completa.
- Actualizar un plan completo, reemplazando días y ejercicios asociados de forma transaccional.
- Eliminar un plan de entrenamiento específico.
- Finalizar días individuales de un plan.
- Alternar (activar/desactivar) el estado de un plan como activo.
- Obtener días de entrenamiento completados por el usuario.
- Recuperar métricas recientes relacionadas con la salud del usuario (peso, IMC, etc.).
- Generar estadísticas de entrenamiento como:
 - Cantidad de días completados.
 - Grupo muscular más entrenado.
 - Rutina más completada.

EjercicioModel: Este modelo gestiona los ejercicios disponibles en la base de datos permitiendo realizar acciones CRUD (crear, leer, actualizar y eliminar) y facilitar la búsqueda de ejercicios por nombre o ID.

Acciones principales:

- Buscar ejercicio por nombres.
- Obtener un ejercicio según el ID enviado.
- Obtener todos los ejercicios ordenados alfabéticamente.
- Crear nuevos ejercicios para la base de datos
- Actualizar ejercicios existentes con solo los campos proporcionados
- Eliminar un ejercicio específico mediante un ID.

5.2.2.3 Middlewares:

Middlewares de validación y control en el backend

El backend incorpora una serie de middlewares responsables de validar, filtrar o modificar datos antes de que lleguen a los controladores. Estos se encuentran ubicados en el directorio de middlewares y están organizados según su propósito específico.

La mayoría de los middlewares se encargan de validar datos usando la librería zod, y comparten una estructura común:

- **Definición de un esquema zod** para validar el cuerpo de la request (req.body) o parámetros (req.params) de una petición.
- **Envío de errores personalizados** si la validación falla.
- **Invocación de next()** para continuar el flujo si los datos son correctos.

Un ejemplo representativo de esta estructura es el middleware validateExerciseCreate:

```
import { z } from "zod";

const passwordRegex = /^[a-zA-Z]+[a-zA-Z\d]+[a-zA-Z\d]{7,}$/;

const loginSchema = z.object({
  correo: z.string().email({ message: "Email no válido" }),
  password: z.string().refine((val) => passwordRegex.test(val), {
    message: "La contraseña debe tener mínimo 7 caracteres, una letra, un número y un carácter especial"
  })
});

export const validateLogin = (req, res, next) => {
  try {
    loginSchema.parse(req.body);
    next();
  } catch (error) {
    return res.status(400).json({
      errors: error.errors.map((err) => ({
        field: err.path[0],
        message: err.message
      }));
    });
  }
};
```

Listado de middlewares

- **validateExercise.js / validateExerciseUpdate.js:** Validan los datos requeridos para crear o actualizar un ejercicio (nombre, tipo, categoría, descanso, series, repeticiones y peso).
- **validateId.js:** Verifica que el *QueryParam* enviado por la URL sean válidos, en este caso valida el (ID).
- **validateImage.js:** Comprueba si el archivo de imagen enviado cumple con las restricciones de formato y tamaño para subirlas al backend (Ejemplo: Formato JPG; JPEG, WEBP y PNG; Tamaño máximo 10MB).
- **validateInfo.js:** Valida datos básicos del usuario, como edad, altura, peso, etc al registrar tu información por primera vez.
- **validateLogin.js / validateRegister.js:** Validan las credenciales de login y los datos de registro (correo, nombre, contraseña).
- **validatePlanUpdate.js:** Comprueba que los datos enviados para modificar que un plan sean válidos (nombre, descripción, diaz, series repeticiones y más).
- **validateStats.js:** Comprueba que los datos para registrar las estadísticas del usuario para hacer su historial de información (fechas, peso, altura, rm y más).
- **validateToken.js:** Verifica la presencia y validez del token JWT enviado por la cabecera *Authorization* antes de acceder a rutas protegidas, se verifica su firma y su expiración.
- **validatedUser.js / validatedUserUpdate.js:** Validan la información del usuario tanto en la creación como en la actualización de su perfil.

Middleware CORS

A diferencia del resto, el middleware cors.js no realiza validaciones con zod, sino que configura políticas de **Cross-Origin Resource Sharing (CORS)** usando la librería cors de Node como podemos ver en el siguiente ejemplo:

```
import cors from "cors";

export const corsMiddleware = () =>
  cors({
    origin: "*",
    methods: ["GET", "POST", "PUT", "PATCH", "DELETE"],
    allowedHeaders: ["Content-Type", "Authorization"]
 });
```

Permitiendo que el cliente (frontend) se comunique con el backend sin restricciones, lo cual es útil durante el desarrollo. En producción, puede configurarse para aceptar solo orígenes específicos se observa en la imagen que permite cualquier origen indicando los métodos y cabeceras disponibles para las solicitudes HTTP.

Middleware Roles

A diferencia de los demás middlewares, el middleware de roles no se encarga de configurar cabeceras o seguridad a nivel de red, sino que controla el acceso a rutas protegidas en función del rol del usuario autenticado. Este middleware verifica que el token JWT incluido en la cabecera de la petición sea válido y, además, extrae el rol del usuario para comprobar si está autorizado a acceder a una ruta específica.

```
export const validateRole = (...allowedRoles) => {
  return (req, res, next) => {
    const { usuario } = req;

    if (!usuario || !usuario.role) {
      return res.status(403).json({ mensaje: "Acceso denegado: sin rol definido." });
    }

    if (!allowedRoles.includes(usuario.role)) {
      return res.status(403).json({ mensaje: "Acceso denegado: rol no autorizado." });
    }

    next();
  };
};
```

Este enfoque nos permite definir rutas accesibles únicamente para ciertos roles, como "admin", "entrenador" o ambos ("both"), garantizando así que los recursos sensibles solo estén disponibles para los perfiles adecuados.

Por ejemplo:

```
userRouter.delete("/delete/:id", validateToken, validateRole("admin"), validateId ,UserController.delete);
```

```
exerciseRouter.post("/create", validateToken, validateRole("admin", "entrenador") ,validateExerciseCreate, EjerciciosController.create);
```

5.3 Seguridad de la aplicación

La aplicación incorpora diversas medidas de seguridad para proteger los datos de los usuarios como la integridad del sistema. La autenticación y autorización se gestionan mediante tokens JWT (JSON Web Tokens) firmados por una clave privada del servidor, estos garantizan protección en las sesiones de cada usuario. Cada token contiene la mínima información necesaria, como el identificador del usuario, su rol y la imagen de perfil, el token posee un tiempo límite antes de que expire la sesión activa, reduciendo así el riesgo de accesos no autorizados.

No se implementa la protección contra ataques CSRF (Cross-Site Request Forgery) en formulario ya que la autenticación de peticiones del usuario se realiza mediante los tokens JWT enviados en la cabecera de Authorization de las peticiones, ya que los navegadores no envían automáticamente estos encabezados en peticiones cross-origin, a diferencia de las cookies. Por tanto, al usar tokens en headers personalizados en lugar de cookies para la gestión de sesión, la aplicación minimiza el riesgo de CSRF de forma natural.

Para proteger las credenciales de acceso de los usuarios los mismos se almacenan de forma segura mediante cifrado con algoritmos robustos (bcrypt), impidiendo que puedan ser recuperadas en texto plano en caso de que haya una vulnerabilidad en la base de datos.

Las peticiones al backend requieren de la inclusión de una cabecera en la petición HTTP llamada Authorization, permitiendo la validación de cada solicitud y asegurando que solo usuarios autenticados puedan acceder a los recursos protegidos del sistema. Además, la aplicación valida los datos recibidos tanto en el frontend como en el backend, incluyendo formularios y parámetros de consulta (query params), para evitar ataques comunes como la inyección de código o datos malintencionados.

Se controla el acceso a determinadas rutas y recursos mediante un sistema de roles, que limita el acceso para aquellos usuarios que no tengan el rol adecuado. Esto se complementa con validaciones en el servidor para evitar accesos indebidos a zonas restringidas incluso en caso de manipulación del frontend.

En conjunto, estas estrategias conforman una arquitectura segura que protege la aplicación y sus usuarios frente a amenazas habituales en aplicaciones web.

6. Pruebas

6.1 Introducción

En este apartado se abordará el desarrollo de pruebas orientadas a verificar que la aplicación mantiene su correcto funcionamiento y estabilidad, incluso tras la incorporación de nuevas funcionalidades o actualizaciones. Estas pruebas permiten detectar posibles errores o comportamientos inesperados que puedan arruinar el sistema o generar fallos críticos en el servidor. Gracias a esta implementación, es posible garantizar que futuras modificaciones no introduzcan errores ni afecten negativamente al rendimiento o la experiencia del usuario.

6.2 Pruebas Con Postman - BackEnd

Para la realización de las pruebas en el servidor (Backend), se ha utilizado la herramienta Postman, que permite enviar peticiones HTTP a los distintos endpoints de nuestra API y verificar sus respuestas. Esta herramienta es útil para comprobar el funcionamiento de nuestro backend, tanto en flujos normales como en situaciones importantes que pueden resultar en errores.

Para las pruebas se realizó una colección en POSTMAN, esta colección nos permite agrupar múltiples peticiones y pruebas en un mismo lugar permitiéndonos correr muchas peticiones en un instante. Se van a realizar 3 pruebas con 2 métodos HTTP distinto, la primera se iniciará sesión como normalmente se hace en nuestra API y con el token proporcionado haremos las otras 2 peticiones que se encargará de consultar todos los ejercicios de nuestra API y todos los usuarios.

The screenshot shows the Postman application interface. On the left, there are navigation tabs for Home, Workspaces, and API Network. Below these are sections for Collections, Environments, Flows, and History. The main content area is titled "BackendFitForge - Run results" and shows a summary of the run: "Ran today at 12:42:47 · View all runs". It lists three tests: "All Tests Passed (3) Failed (0) Skipped (0)". The "Iteration 1" section details three individual tests: "POST Login test" (http://185.203.216.150:8081/login), "GET Get exercises test" (http://185.203.216.150:8081/exercises/get), and "GET Get users test" (http://185.203.216.150:8081/users/get). Each test is marked as "PASS" with a green status bar and includes a description of the expected response code. A notification at the bottom left says "You've made new changes" with a link to "Post an update to inform others". The bottom of the screen shows various Postman tools and settings.

6.3 Pruebas Con Selenium - FrontEnd

Para la realización de pruebas en el cliente (Frontend), se ha utilizado la herramienta Selenium IDE, esta es una extensión para navegadores que permite ejecutar pruebas automatizadas definiendo pasos simulando la interacción del usuario con la interfaz gráfica de la aplicación de forma automática. Esta herramienta resulta especialmente útil para verificar el comportamiento y la respuesta en un flujo normal de uso de la aplicación.

Como prueba, se diseñó un test automatizado del proceso de inicio de sesión. En esta prueba, Selenium ejecuta una serie de pasos que incluyen la carga de la vista de login, la introducción de un correo electrónico y una contraseña de prueba, y la posterior validación de que el usuario accede correctamente a la aplicación. Esta verificación asegura que el sistema de autenticación funcione de forma adecuada y que el flujo de la aplicación no presente errores en su comportamiento básico.

Selenium IDE - FitForge* - Opera

Project: FitForge*

Executing ✓ Iniciar Sesión*

http://localhost:5173/login

Command	Target	Value
1 ✓ open	/login	
2 ✓ click	email	lunarmorera@gmail.com
3 ✓ click	password	xdcvfg560*

Run: 1 Failures: 0

Log Reference

Running 'Iniciar Sesión'

1. open on /login OK
 2. click on email with value lunarmorera@gmail.com OK
 3. click on password with value xdcvfg560* OK

Warning implicit locators are deprecated, please change the locator to id=email
 Warning implicit locators are deprecated, please change the locator to id=password

'Iniciar Sesión' completed successfully

7. Despliegue de la Aplicación

7.1 Introducción

Para garantizar la disponibilidad y accesibilidad de la aplicación desde cualquier lugar, se optó por realizar el despliegue en una VPS (Servidor Privado Virtual). Esta decisión permite contar con un entorno controlado, permanente y personalizable, lo cual es ideal para entornos productivos. Además, se ha empleado Docker como herramienta de contenerización para facilitar la configuración, replicación y puesta en marcha de los diferentes servicios que componen la aplicación.

El uso combinado de Docker y una VPS proporciona ventajas como el aislamiento de servicios, la automatización del despliegue, y una mayor portabilidad del sistema completo.

7.2 Proceso de despliegue

El despliegue de la aplicación se ha realizado utilizando contenedores Docker para asegurar la portabilidad, escalabilidad y facilidad de mantenimiento del sistema. Gracias a Docker y a la herramienta docker-compose, ha sido posible orquestar todos los servicios necesarios para la ejecución del proyecto, tanto en entorno local como en producción.

Se han definido cuatro servicios principales: postgres, pgadmin, backend y frontend, cada uno de estos servicios se encuentra aislado en su propio contenedor, lo cual facilita la detección de errores. La inclusión de pgadmin permite gestionar de forma visual y eficiente la base de datos PostgreSQL, mientras que los servicios de backend y frontend se han configurado con sus respectivos Dockerfile para garantizar que cada entorno cuente con sus dependencias y configuraciones personalizadas, estos sirven para hacer build de la imagen utilizando las técnicas de construcción de Docker Multi-Stage garantizando que las imágenes sean lo más ligeras posibles a la hora de construir el proyecto y exponer el servicio.

Se hace uso de redes internas (networks) que aseguran la comunicación entre contenedores sin necesidad de exponer servicios innecesarios al exterior, lo que

mejora la seguridad y la velocidad de comunicación entre servicios. Asimismo, el uso de volúmenes persistentes garantiza que la información crítica, como la base de datos o las imágenes de usuario subidas, no se pierda tras reinicios o actualizaciones.

Este diseño permite un rápido despliegue del entorno completo con un solo comando, sino que también promueve buenas prácticas de despliegue y portabilidad, fundamentales en entornos productivos reales.

El proceso de despliegue se lleva a cabo mediante un archivo docker-compose.yml, que es el encargado de organizar la creación y ejecución de los contenedores necesarios. A continuación, se definen los servicios construidos con Docker:

PostgreSQL: Base de datos principal de la aplicación, se encuentra expuesta en el puerto 5432 y con un volumen persistente asignado para la persistencia de datos.

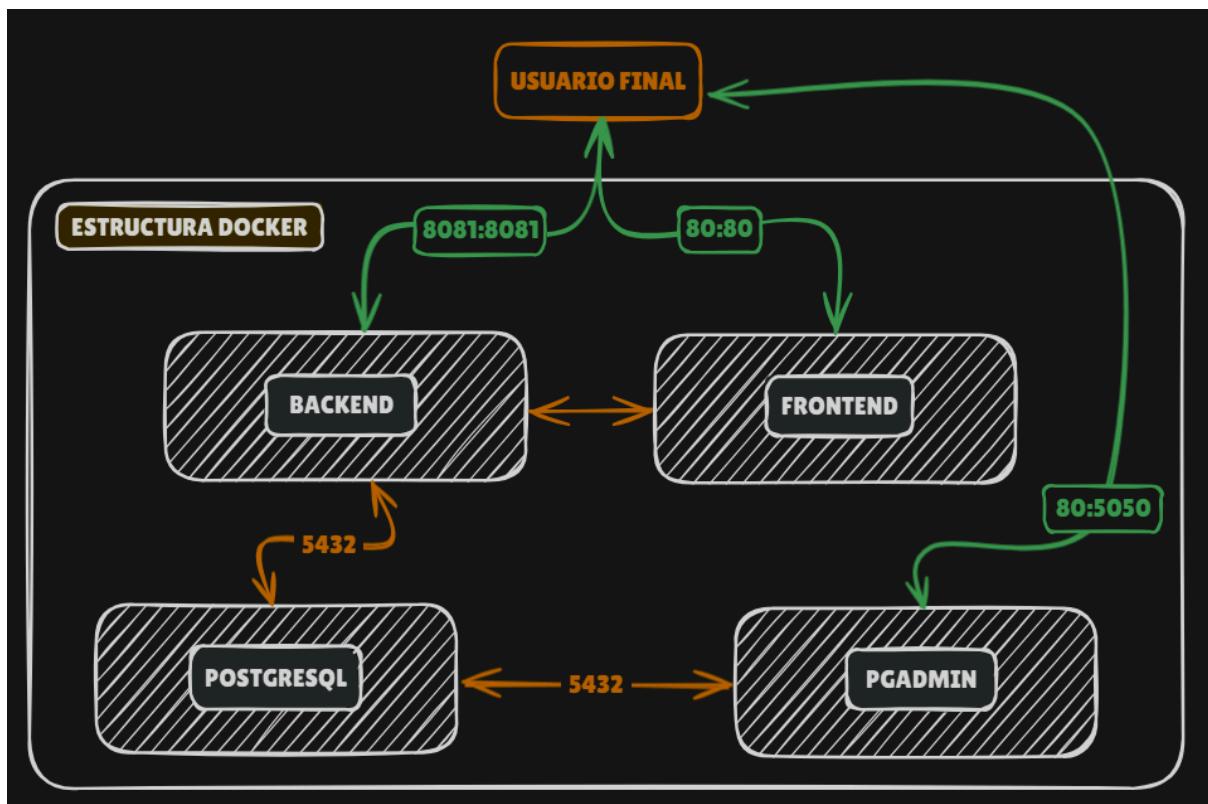
pgAdmin: Herramienta web para la administración de la base de datos, accesible desde el puerto 5050 también posee un volumen persistente.

Backend: Nuestro servidor backend de Node.js con Express, configurado para conectarse a la base de datos, se encuentra expuesto en el puerto 8081 y se encuentra gestionado y configurado mediante un Dockerfile personalizado.

Frontend: Nuestra aplicación cliente frontend desarrollada con Vue.js, construida y servida desde un contenedor Nginx, también configurada y gestionada mediante un Dockerfile, se encuentra expuesta en el puerto 80 siendo accesible desde el navegador con solo ingresar la dirección IP de nuestro servidor.

7.3 Esquema final de la estructura

A continuación podemos ver el diseño final de la estructura utilizada para el despliegue:



8. Conclusiones

8.1 Conclusiones Finales

El desarrollo de esta aplicación concluye de forma satisfactoria, superando incluso mis expectativas iniciales. En un principio, para mí era complicado entender que iba abarcar todo el ciclo de vida de una aplicación web moderna desde su diseño hasta la implementación, despliegue y documentación final dentro del tiempo disponible que teníamos.

Sin embargo, gracias al esfuerzo y la constancia seguida acompañada de recursos formativos (cursos, documentación oficial, foros técnicos, vídeos y artículos), se ha conseguido construir una solución completa, funcional y bien estructurada de pies a cabeza. Además, ha sido fundamental el apoyo de mis compañeros tanto de prácticas como del instituto para compartir conocimientos, fragmentos de código y buenas prácticas a la hora de realizar el desarrollo de aplicaciones. Este proyecto ha sido, en gran parte, posible gracias al profesorado, mi familia y compañeros que me han apoyado constantemente al continuo desarrollo de mi proyecto.

8.2 Desviaciones Temporales

Durante el desarrollo se han producido algunas desviaciones temporales, relacionadas a el aprendizaje y adaptación de las tecnologías seleccionadas. Como el aprendizaje de NodeJS para la realización de la API y herramientas como Vue 3 con Composition API para la realización del cliente, Prisma para la Base de Datos, o el uso avanzado de Tailwind CSS para el estilo y diseño de la aplicación web junto a la gestión de autenticación basada en JWT, estas herramientas presentaron desafíos iniciales debido a ser la primera vez usandolas siguiendo un patrón de trabajo y tecnologías totalmente distinto al seguido en clases.

Pese a estas dificultades, se decidió afrontarlas como un reto, recurriendo a tutoriales y diversos recursos en la web que permitiera adquirir los conocimientos necesarios, siempre con el objetivo de enriquecer la experiencia de aprendizaje.

8.3 Posibles ampliaciones

Como ampliaciones futuras del proyecto, se han contemplado varias ideas que podrían mejorar considerablemente la UX y explotar el potencial de la aplicación así como las funcionalidades ofrecidas:

- **Integración con dispositivos de seguimiento físico**, como pulseras inteligentes, para registrar automáticamente métricas como pasos, calorías o ritmo cardíaco.
- **Sistema de mensajería entre usuarios y entrenadores**, facilitando la comunicación directa, el seguimiento personalizado y la resolución de dudas.
- **Aplicación móvil o de escritorio**, para facilitar el acceso desde dispositivos Desktop/Android/iOS y aumentar la usabilidad en el día a día del usuario permitiéndoles llevar a FitForge a cualquier lado donde lo necesiten.
- **Integración de planes nutricionales**, con el objetivo de ofrecer un enfoque más completo, se plantea la posibilidad de incorporar planes nutricionales personalizados. Esta funcionalidad permitiría orientar al usuario no solo en el ámbito físico, sino también en el alimenticio, recomendando proporciones adecuadas de alimentos en función de sus características y objetivos personales.
- **Sistema de notificaciones**, se propone añadir un sistema de notificaciones que contribuya a mantener la constancia del usuario en sus actividades físicas. Estas alertas podrían utilizarse para recordar entrenamientos, registrar progresos o motivar al usuario a seguir con su rutina.

8.4 Valoración General

Desde una perspectiva personal, este proyecto ha supuesto para mí un aprendizaje profundo permitiendo no solo reforzar mis conocimientos previos, sino también darme la oportunidad de enfrentarme a nuevos retos en ámbitos como la arquitectura backend en Node, el diseño de interfaces modernas con Tailwind, la

autenticación segura o la interacción entre frontend y backend mediante APIs RESTful.

También se ha aprendido a estructurar un proyecto realista, escalable y adaptable a necesidades reales, como lo sería en un entorno profesional siguiendo también su propio despliegue permitiéndonos servir nuestra aplicación a todo el mundo. El resultado es una aplicación funcional que cumple con su propósito: ayudar al usuario a gestionar de forma personalizada su entrenamiento y salud física de forma automática.

Este trabajo no solo ha servido y fortalecido mis conocimientos, sino también me ha ayudado a desarrollar habilidades de organización, resiliencia y resolución de problemas. El proyecto, además, sirve como base o inspiración para futuras aplicaciones similares, ya sea en contextos académicos, personales o empresariales.

9. Bibliografía

https://www.youtube.com/playlist?list=PLUofhDlq_38qm2oPOV-IRTTEKyrVBaU7

<https://tailwindcss.com/docs/installation/using-vite>

<https://www.npmjs.com>

<https://vuejs.org/guide/essentials/component-basics.html>

<https://docs.docker.com/compose/>

<https://contabo.com/es/vps/>

<https://www.geeksforgeeks.org/software-engineering/mvc-framework-introduction/>

<https://developer.spotify.com/documentation/web-api>

<https://axios-http.com/es/docs/intro>

<https://www.chartjs.org/docs/latest/general/colors.html>

10. Anexos

10.1 Casos de uso

Caso de Uso	Iniciar sesión del usuario	CU_US_01
Actores	Usuarios	
Tipo	Esencial, Primario	
Referencias	CU_RE1	
Postcondición	El usuario debe haberse registrado anteriormente en la aplicación siguiendo e introduciendo los datos solicitados para su correcto funcionamiento.	
Propósito		
Dar al usuario un token de acceso para que pueda acceder a la aplicación y a los recursos del cliente.		
Resumen		
El sistema recoge el usuario y la credencial de acceso del usuario y verifica los datos con los que se encuentren en la base de datos, si las credenciales coinciden devuelve un Token JWT al usuario para proporcionarle acceso a la aplicación.		

Curso Normal	
1.-	El usuario ingresa sus datos y son validados
2.-	Se busca si el correo se encuentra en la base de datos
3.-	Se comprueba si la contraseña ingresada coincide con la de la base de datos
4.-	Se crea un token JWT con la información mínima del usuario
5.-	El token se le devuelve al usuario proporcionándole acceso al sistema

Cursos con anomalías	
1a.-	Datos ingresados erróneos, se le muestran los errores al usuario, vuelve al punto 1.
2a.-	El usuario no existe en la base de datos devuelve con error al usuario al punto 1
3a.-	La contraseña no coincide devuelve con error al usuario al punto 1

Caso de Uso	Registrar un nuevo usuario	CU_RE1
Actores	Usuario no registrado	
Tipo	Esencial, Primario	
Referencias	Ninguna	
Postcondición	El visitante no debe haberse registrado anteriormente	
Propósito		
Registrar un nuevo usuario en la base de datos con sus datos ingresados		
Resumen		
El visitante ingresará sus datos y luego serán tratados para crear un nuevo usuario en la base de datos con el que podrá acceder a la aplicación.		

Curso Normal	
1.-	El visitante ingresa sus datos de registro
2.-	Se busca si el correo no se encuentra registrado en la base de datos
3.-	Se registra el usuario en la base de datos
4.-	El token se le devuelve al usuario proporcionándole acceso al sistema
5.-	El usuario ahora registrado completa su perfil con el resto de sus datos
6.-	Se guardará toda la información complementaria del perfil registrada del usuario y se le da acceso al sistema

1a.-	Datos ingresados erróneos, se le muestran los errores al usuario, vuelve al punto 1.
2a.-	Ya existe un correo asignado a un usuario en la base de datos se le envia un error y se devuelve al punto 1
5a.-	Los datos complementarios ingresados por el usuario para completar su perfil son erróneos, se le muestran los errores al usuario y vuelve al punto 5.

Caso de Uso	Confirmar dia entrenado	CU_DIA_01
Actores	Usuario iniciado sesión	
Tipo	No Esencial, Secundario	
Referencias	CU_US_01	
Postcondición	El usuario debe haber iniciado sesión anteriormente y debe tener un plan de entrenamiento registrado con al menos un día de entrenamiento que corresponda al dia actual	
Propósito		
Dar por finalizado un dia de entrenamiento en una rutina		
Resumen		
Se le preguntará al usuario si ha entrenado el día de hoy y el usuario podrá confirmar en caso de haberlo hecho		

Curso Normal	
1.-	El sistema busca un plan de entrenamiento que esté activo
2.-	Se ve que hay un dia para entrenar
3.-	El usuario confirma el dia de entrenamiento

1a.-	El sistema no encuentra que haya un plan de entrenamiento activo, muestra al usuario que no hay entrenamiento el dia de hoy y se devuelve al punto 1
2a.-	El sistema encuentra el plan pero no encuentra un día de entrenamiento que corresponda al de hoy, le avisa al usuario y se devuelve al punto 1
3a.-	El usuario no confirma el dia de entrenamiento y pierde la oportunidad de confirmarlo

Caso de Uso	Generar un plan de entrenamiento de forma automática	CU_PLA_A
Actores	Usuario iniciado sesión	
Tipo	Esencial, Secundario	
Referencias	CU_US_01	
Postcondición	El usuario debe haber iniciado sesión anteriormente	
Propósito		
Crear una rutina de entrenamiento con días personalizados basados en grupos musculares y ejercicios variados según los datos del usuario		
Resumen		
El usuario se dirigirá a rutinas e ingresará como desea su plan introduciendo objetivo, nivel y rm en caso de ser necesario y el sistema creará un plan de entrenamiento de forma automática		

Curso Normal	
1.-	El usuario ingresa su nivel y objetivo
2.-	Si el nivel es avanzado el usuario ingresará su rm en caso de no tenerlo registrado
3.-	El usuario ingresará los datos restantes y enviará la solicitud para crear el plan de entrenamiento.
4.-	Si el nivel es avanzado se ajusta el número de repeticiones y peso para algunos ejercicios
5.-	Si el objetivo es fuerza se ajustará nuevamente las repeticiones y el peso para los ejercicios
5.5.-	Si el usuario no tiene RM y el plan es avanzado lo guardará en la base de datos en caso contrario tomará unos valores por defecto
6.-	Se generarán los días de entrenamiento y sus ejercicios y se le devolverá al usuario el nuevo plan de entrenamiento

3a.-	Los datos son erróneos por lo que le indica al usuario los errores devolviéndole al punto 1
5.5a.-	El usuario no fue encontrado por lo que no se puede actualizar su rm

Caso de Uso	Crear un nuevo ejercicio	CU_EJ_01		
Actores	Administrado o entrenador iniciado sesión			
Tipo	No Esencial, Secundario			
Referencias	CU_US_02			
Postcondición	El administrador o entrenador debe haber iniciado sesión anteriormente en la aplicación			
Propósito				
Crear un nuevo ejercicio que se encontrará disponible en la implementación de nuevos ejercicios para un entrenamiento				
Resumen				
El entrenador o el administrador introducirá el tipo de ejercicio que desea crear junto a los valores por defecto que va a tener cada uno				

Curso Normal	
1.-	El administrador/entrenador introduce los datos que tendrá el nuevo ejercicio que desea crear
2.-	Se comprueba que ese ejercicio no existe
3.-	Se crea el ejercicio con los valores ingresado por el administrador/entrenador

1a.-	Los datos son erróneos, se le indica al usuario y le devuelve al punto 1
2a.-	El ejercicio existe, se le indica al usuario y le devuelve al punto 1

No se han desarrollado todas las tablas correspondientes a los casos de uso debido a la extensión y complejidad del proyecto, por ende se ha priorizado aquellos casos que representaban funcionalidades esenciales para el funcionamiento adecuado de la aplicación.

Se ha optado por documentar los casos de uso más representativos y relevantes, ya que estos nos permiten comprender el flujo de trabajo y las interacciones del sistema. Esta decisión se tomó con el objetivo de optimizar el tiempo disponible y centrar los esfuerzos en los apartados más significativos desde el punto de vista funcional y técnico.

10.2 Guía de estilos

LOGO



Se han creado dos logos diferentes para usarlos dependiendo de la página y el espacio que haya en cada sitio

TIPOGRAFÍA

Durante el diseño de la aplicación se han usado múltiples tipografías que representan el estilo de vida de la página, entre ellas tenemos la siguientes:

REVAMPED: Esta fuente importada directamente desde el cliente es una fuente con estilo militar que destaca el estilo de la aplicación esta es la fuente principal utilizada en el título de la aplicación y en el logo.



Esta fuente se carga desde un fichero tipo *.woff* de la siguiente manera:

```
@font-face {  
    font-family: "Revamped";  
    src: url("/Revamped.woff") format("woff");  
    font-weight: normal;  
    font-style: normal;  
}  
  
.revamped {  
    font-family: "Revamped";  
    font-weight: 100;  
}
```

ANTON: Esta fuente es utilizada para los títulos y enlaces de la página principal se escogió por su formalidad y seriedad que aportan un enfoque moderno y casual a la página.

¿QUE ES FITFORGE?

Esta fuente es importada desde google font por lo que solo definimos el uso de la fuente como una clase más.

```
.anton {  
    font-family: "Anton", sans-serif;  
    font-weight: 100;  
}
```

SANS-SERIF: Se ha decidido en usar esta fuente como fuente principal para la interfaz del dashboard y párrafos debido a su alta legibilidad y claridad visual en entornos digitales. Además permite leer perfectamente los párrafos y textos en cada una de las pantallas sin problema alguno aportando modernidad al texto.

Registrar Información

Registrar tu información nos permite darte un seguimiento más preciso de tus estadísticas. Solo necesitas actualizar tus progresos cada semana, y nosotros nos encargamos del resto.

Esta fuente ya se encuentra importada en nuestro sistema y se encuentra activa por defecto al usar tailwind como hoja de estilos. No hace falta llamarla ni asignando ninguna clase al elemento HTML

ICONOS

Los iconos utilizados en la aplicación son importados de la librería disponible para vue llamada Vue-Lucide, lucide es un paquete de iconos modernos utilizados en el diseño de páginas web que aportan minimalismo y sencillez a cada página.



<https://lucide.dev/icons/>

PALETA DE COLORES

Para el diseño de la interfaz del usuario se ha optado por una paleta de colores que refleje un equilibrio entre seriedad, claridad y dinamismo. Aportando un buen contraste entre los colores.



La selección de estos colores no solo se basa en criterios estéticos, sino también en principios de accesibilidad y contraste, asegurando una experiencia de usuario clara y agradable, especialmente en entornos con poca luz o en dispositivos móviles.

Primer Color: #151419: un tono muy oscuro, casi negro, es utilizado como color base para los elementos principales de la interfaz, como fondos, encabezados o áreas destacadas. Transmite elegancia, seriedad y contraste visual.

Segundo Color: #fbfbfb: un tono blanco cálido, empleado para textos o fondos secundarios. Mejora la legibilidad y proporciona equilibrio frente al color principal oscuro.

Tercer Color: #f56e0f: un color naranja vibrante utilizado para los elementos interactivos (botones, enlaces activos, íconos destacados). Representa energía, acción y dinamismo, alineándose con el enfoque activo y motivador de una aplicación relacionada con salud o entrenamiento.

Cuarto Color: #262626: un gris oscuro que funciona como color auxiliar para fondos secundarios, separadores o menús. Añade profundidad sin perder coherencia con la estética general.

Quinto Color: #878787: un gris medio utilizado para textos secundarios, descripciones o elementos deshabilitados. Aporta jerarquía visual sin robar protagonismo a la información principal.

BOTONES

Los botones utilizados en la aplicación siguen una línea de diseño que nos permite mantener una identidad visual clara en toda la interfaz. Aunque sus funcionalidades son totalmente distintas, la mayoría de estos comparten la misma estructura visual y se clasifican principalmente en dos tipos: primarios y secundarios.

Los botones primarios se utilizan para acciones principales. Se caracterizan por un fondo naranja, texto blanco en negrita y un ícono que refuerza la acción, todo con un estilo llamativo que destaca visualmente.



Los botones secundarios, en cambio, se reservan para acciones complementarias. Tienen fondo blanco, texto naranja y una sombra sutil que les da profundidad, manteniendo un diseño más discreto pero aún visible.



10.3 Lógica Rutinas Automáticas

Lógica de generación automática de planes de entrenamiento

La generación de planes de entrenamiento se basa en tres variables clave: **nivel del usuario** (básico o avanzado), **objetivo** (hipertrofia o fuerza), y en el caso avanzado, su **Repetición Máxima (RM)**.

- **RM:** representa el mayor peso que un usuario puede levantar en una sola repetición. Para usuarios básicos que no declaran su RM, se establece un valor por defecto (50 kg en hombres, 30 kg en mujeres). En ejercicios de brazos y hombros, se usa el 30 % del RM para reducir la carga.

Distribución de días y grupos musculares

Se generan rutinas de 6 días a la semana, distribuidos según el sexo:

- **Hombres:** Lunes y jueves (pecho), martes y viernes (piernas), miércoles y sábado (espalda).
- **Mujeres:** Lunes, miércoles y viernes (piernas), martes y sábado (espalda), jueves (full body).

Volumen e intensidad por nivel

Nivel	Ejercicios/grupo	Series (Hipertrofia / Fuerza)	Descanso (Hipertrofia / Fuerza)
Básico	10	3 / 5	90 s / 5 min
Avanzado	16	5 / 6	2 min / 5 min

Intensidad diaria y progresión

La intensidad se organiza cíclicamente por día según el objetivo:

- **Hipertrofia:**

- Cargas: 60–70 % RM (12 repeticiones) y 70–80 % RM (8 repeticiones).
- Secuencia semanal: 60 %, 60 %, 80 %, 60 %, 80 %, 80 %, 60 %, 80 %.

- **Fuerza:**

- Cargas: 85–95 % RM (3 repeticiones).
- Secuencia semanal: 85 %, 85 %, 95 %, 85 %, 95 %, 95 %, 85 %, 95 %.

Esta progresión se repite cíclicamente y permite una sobrecarga progresiva sin generar fatiga excesiva.

Cálculo del peso

Para cada ejercicio y día, el peso se calcula dinámicamente como:

$$\text{Peso} = \text{RM} \times \text{Peso} = \text{RM} \times \% \text{ intensidad diaria}$$

Por ejemplo, si el RM es 100 kg y la intensidad del día es 60 %, se asigna un peso de 60 kg.

10.4 Enlace a la aplicación

El enlace a nuestra aplicación es el siguiente: <http://185.203.216.150>