Aplicación Web para Gestión de Rutinas y Entrenamientos de Gimnasio

Carlos Perdomo Figueroa

Samuel Vergara Rojas

Ivan Felipe Morales

Arquitectura y Modelamiento de Software

Universidad Cooperativa De Colombia

Campus Villavicencio

2025

1. Introducción

El presente proyecto consiste en el desarrollo de una aplicación web diseñada para optimizar la gestión y seguimiento de rutinas de ejercicio en gimnasios. La aplicación permitirá a los usuarios crear, personalizar y monitorear sus entrenamientos de manera sistemática, proporcionando herramientas para el registro detallado de pesos, repeticiones, técnicas aplicadas y progreso temporal.

La plataforma está estructurada bajo un sistema de roles diferenciados, donde los administradores gestionan la base de datos de ejercicios con contenido multimedia educativo, mientras que los usuarios finales se enfocan en la creación y seguimiento de sus rutinas personalizadas. Este enfoque garantiza tanto la calidad del contenido técnico como la flexibilidad en el uso personal de la aplicación.

2. Planteamiento del problema y/o justificación

En el contexto actual del fitness y el entrenamiento físico, existe una necesidad creciente de herramientas digitales que faciliten el seguimiento sistemático de las rutinas de ejercicio. Los métodos tradicionales de registro, como libretas físicas o aplicaciones genéricas, presentan limitaciones significativas en términos de organización, accesibilidad y análisis del progreso.

Los usuarios principiantes en gimnasios enfrentan desafíos particulares relacionados con la correcta ejecución de ejercicios, la estructuración adecuada de rutinas y el seguimiento consistente de su progreso. Por otro lado, los usuarios más experimentados requieren herramientas que les permitan registrar técnicas avanzadas como dropsets, myo-reps o entrenamientos al fallo, así como mantener un historial detallado de su evolución.

La falta de una plataforma integral que combine la gestión de rutinas, el registro de entrenamientos, el acceso a material educativo y el análisis histórico del rendimiento, justifica el desarrollo de esta aplicación web especializada.

3. Objetivos

3.1. Objetivo general

Desarrollar una aplicación web para la gestión integral de rutinas y entrenamientos de gimnasio mediante una arquitectura basada en microservicios que garantice escalabilidad y mantenibilidad del sistema.

3.2. Objetivos específicos

- Implementar servicios independientes para cada funcionalidad que tendrá la aplicación (auth, exercise, routine, workout).

- Crear funcionalidades de planificación y seguimiento de rutinas personalizadas con bases de datos independientes que aseguren la integridad y autonomía de los datos

- Construir un módulo de historial de entrenamientos que implemente patrones de comunicación entre servicios para garantizar la consistencia de datos temporales

4. Marco Referencial

4.1. Estado del Arte

En el contexto actual del fitness digital, el mercado de aplicaciones para gimnasio y entrenamiento ha experimentado un crecimiento exponencial. Las aplicaciones de ejercicio se colocan como una de las tendencias fitness 2025 más importantes, subiendo al puesto número 7 desde su posición en 2024, lo que demuestra la creciente demanda de soluciones digitales especializadas.

Las tecnologías emergentes están transformando significativamente la experiencia del usuario en el ámbito fitness. La Realidad Virtual (VR) y la Realidad Aumentada (AR) están revolucionando la manera en que las personas experimentan el ejercicio físico en los gimnasios, ofreciendo experiencias inmersivas que trascienden los métodos tradicionales de entrenamiento.

En términos de funcionalidad, las aplicaciones modernas de fitness se enfocan en proporcionar herramientas integrales de seguimiento y motivación. Estas aplicaciones permiten a los miembros del gimnasio realizar un seguimiento de sus entrenamientos, establecer metas y ver su progreso directamente en sus teléfonos, estableciendo un estándar para la experiencia del usuario que prioriza la accesibilidad y el monitoreo continuo.

El panorama competitivo incluye aplicaciones establecidas como Google Fit, Fitbit, Nike Training Club y plataformas especializadas como Alo Moves, que ofrece clases de yoga, fitness, mindfulness y autocuidado dirigidas por instructores de talla mundial. Estas plataformas han establecido las expectativas del mercado en términos de calidad de contenido multimedia y experiencia de usuario.

Las tendencias actuales también indican una evolución hacia la integración de inteligencia artificial y personalización avanzada. Tonal lanzó un sistema de entrenamiento de fuerza inteligente impulsado por la IA, mejorando los entrenamientos domésticos, señalando la dirección futura del sector hacia la automatización y adaptación inteligente de rutinas.

4.2. Marco Teórico

Arquitectura de Software

La arquitectura de software representa la estructura fundamental de un sistema de software, definiendo los componentes principales, sus relaciones y los principios que guían su diseño y evolución. Constituye el blueprint que establece cómo los diferentes elementos del sistema interactúan para cumplir con los requisitos funcionales y no funcionales establecidos. La arquitectura de software es crucial para garantizar la escalabilidad, mantenibilidad, reutilización y evolución de los sistemas de software complejos.

En el contexto del desarrollo de aplicaciones web para gestión de entrenamientos, la arquitectura de software define la organización de módulos como la gestión de usuarios, ejercicios, rutinas, seguimiento de entrenamientos y generación de reportes. Una arquitectura bien definida facilita la separación de responsabilidades, mejora la estabilidad del sistema y permite la implementación de patrones de diseño apropiados para cada funcionalidad específica.

La selección de una arquitectura apropiada impacta directamente en la calidad del software, los costos de desarrollo y mantenimiento, así como en la capacidad del sistema para adaptarse a cambios futuros en los requisitos del negocio.

Diagramas de Componentes

Los diagramas de componentes UML representan las relaciones entre los componentes individuales del sistema mediante una vista de diseño estática, mostrando la estructura de los componentes de la arquitectura de un sistema y cómo están conectados e interactúan. Estos diagramas son fundamentales para visualizar la organización modular del software y las dependencias entre diferentes partes del sistema.

Los diagramas UML de componentes agrupan los componentes en clústeres lógicos y visualizan las relaciones que hay entre ellos. Estos diagramas son perfectos para desglosar

sistemas complejos en sus componentes más pequeños para hacerlos más fáciles de asimilar. En el desarrollo de aplicaciones de fitness, un diagrama de componentes puede mostrar cómo interactúan componentes como el sistema de autenticación, el gestor de ejercicios, el controlador de rutinas y el módulo de reportes.

La implementación efectiva de diagramas de componentes facilita la comprensión de la arquitectura del sistema tanto para desarrolladores como para stakeholders, permitiendo identificar puntos de integración críticos y potenciales cuellos de botella en el sistema.

Diagramas de Secuencia

Los diagramas de secuencia forman parte de los diagramas de comportamiento UML y se utilizan para modelar la interacción entre objetos en el tiempo. Estos diagramas muestran cómo los objetos colaboran para realizar una funcionalidad específica, representando el intercambio de mensajes en orden cronológico.

En el contexto de una aplicación de gestión de entrenamientos, los diagramas de secuencia son especialmente útiles para modelar procesos como el registro de un nuevo entrenamiento, la creación de una rutina personalizada o la consulta del historial de entrenamientos. Estos diagramas ayudan a identificar la secuencia correcta de operaciones, los puntos de validación necesarios y las interacciones entre diferentes capas del sistema (presentación, negocio y datos).

La utilización de diagramas de secuencia durante el diseño permite identificar tempranamente problemas de diseño, optimizar el flujo de comunicación entre componentes y documentar claramente el comportamiento esperado del sistema.

Diagramas de Despliegue

En UML, un diagrama de implementación muestra el hardware del sistema y el software instalado en el hardware, así como el middleware utilizado para conectarse entre computadoras heterogéneas. Un diagrama de implementación generalmente se considera un diagrama de red o un diagrama de infraestructura que describe la configuración física del sistema en tiempo de ejecución.

Los diagramas de despliegue son esenciales para planificar la implementación de aplicaciones web, ya que muestran cómo los componentes de software se distribuyen en los nodos de hardware disponibles. Para una aplicación de gestión de entrenamientos, estos diagramas pueden representar la distribución entre servidores web, servidores de aplicación, servidores de base de datos y servicios de almacenamiento en la nube.

Estos diagramas facilitan la planificación de la infraestructura necesaria, la identificación de requisitos de conectividad, la estimación de recursos de hardware y la definición de estrategias de escalamiento horizontal y vertical del sistema.

Model Driven Engineering (MDE)

La Ingeniería Dirigida por Modelos (MDE, por sus siglas en inglés) es una metodología de desarrollo de software que se enfoca en crear y explotar modelos de dominio, los cuales son modelos conceptuales de todos los temas relacionados con un problema específico. Por lo tanto, destaca y tiene como objetivo las representaciones abstractas del conocimiento y las actividades involucradas en el dominio del problema que se está resolviendo.

El MDE promueve el uso de modelos como artefactos primarios en el proceso de desarrollo de software, elevando el nivel de abstracción y permitiendo que los desarrolladores se enfoquen en la lógica del dominio en lugar de en detalles de implementación específicos de la plataforma. Esta aproximación mejora la productividad del desarrollo, reduce errores y facilita el mantenimiento del software.

En el desarrollo de aplicaciones de fitness, MDE permite modelar conceptos del dominio como ejercicios, rutinas, entrenamientos y usuarios de manera abstracta, independientemente de la tecnología de implementación específica. Esto facilita la generación automática de código, la validación de modelos y la evolución coherente del sistema.

Model Driven Architecture (MDA)

La Arquitectura Dirigida por Modelos (MDA®, por sus siglas en inglés) es un enfoque para el diseño, desarrollo e implementación de software dirigido por la OMG (Object Management Group). MDA proporciona directrices para estructurar especificaciones, las cuales se expresan como modelos. La arquitectura dirigida por modelos es un enfoque de diseño de software para el desarrollo de sistemas de software que proporciona un conjunto de directrices para la estructuración de especificaciones expresadas como modelos.

La estrategia de Arquitectura Dirigida por Modelos (MDA) de la OMG visualiza un mundo donde los modelos juegan un papel más directo en la producción de software, siendo susceptibles de manipulación y transformación por máquina. La Ingeniería Dirigida por Modelos (MDE) tiene un alcance más amplio que MDA, siendo MDA una implementación específica de los principios MDE promovida por la OMG.

MDA se basa en la separación de la especificación del funcionamiento de un sistema de los detalles de cómo ese sistema utiliza las capacidades de una plataforma particular. Esta separación permite la portabilidad, interoperabilidad y reutilización del software a través de diferentes plataformas tecnológicas.

En el contexto de aplicaciones web para fitness, MDA facilita la creación de modelos independientes de la plataforma (PIM - Platform Independent Model) que pueden transformarse automáticamente en modelos específicos de la plataforma (PSM - Platform Specific Model) para diferentes tecnologías web, bases de datos o arquitecturas de despliegue.

La adopción de MDA en el proyecto permite mantener la coherencia arquitectónica a través de diferentes versiones del sistema, facilita la migración tecnológica y mejora la capacidad de respuesta ante cambios en los requisitos del negocio.

Microservicios

La arquitectura de microservicios es un enfoque para el desarrollo de software que consiste en construir una aplicación como un conjunto de pequeños servicios, los cuales se ejecutan en su propio proceso y se comunican con mecanismos ligeros. Esta arquitectura representa el concepto opuesto al de la arquitectura monolítica, ya que es un método que se basa en una serie de servicios que se pueden implementar de forma independiente.

Con una arquitectura de microservicios, una aplicación se crea con componentes independientes que ejecutan cada proceso de la aplicación como un servicio. Cada microservicio se centra en una tarea específica y se comunica con otros servicios a través de interfaces bien definidas, generalmente utilizando protocolos de comunicación como HTTP/REST o mensajería asíncrona.

En el contexto de una aplicación de gestión de entrenamientos, los microservicios permitirían descomponer el sistema en servicios independientes. Cada uno de estos servicios puede desarrollarse, desplegarse, escalarse y mantenerse de manera independiente, lo que proporciona mayor flexibilidad y robustez al sistema.

Los microservicios ofrecen ventajas significativas en términos de escalabilidad, mantenibilidad y desarrollo distribuido. Permiten que diferentes equipos trabajen de manera independiente en distintos servicios, facilitan la adopción de diferentes tecnologías según las necesidades específicas de cada servicio, y mejoran la resistencia del sistema ya que el fallo de un servicio no necesariamente compromete toda la aplicación.

Sin embargo, también introducen complejidad adicional en términos de gestión de la comunicación entre servicios, consistencia de datos distribuidos y monitoreo del sistema. La

implementación exitosa de una arquitectura de microservicios requiere consideraciones cuidadosas sobre la definición de límites de servicios, estrategias de comunicación y herramientas de orquestación y monitoreo.

4.3. Marco Legal

El desarrollo de aplicaciones web en Colombia está sujeto a un marco legal específico que regula tanto la protección de software como el tratamiento de datos personales y las actividades comerciales digitales.

Protección de Software y Derechos de Autor

La Ley 603 de 2000, conocida como la "Ley del Software", establece el marco legal para la protección, el desarrollo y la comercialización de programas informáticos en el país. Esta ley reconoce el software como una obra protegida por derechos de autor, estableciendo los fundamentos jurídicos para la protección de las aplicaciones desarrolladas.

El registro de software como obra protegida por derechos de autor proporciona seguridad jurídica a los desarrolladores. En Colombia, este registro se realiza a través de la Dirección Nacional de Derecho de Autor, quienes ofrecen el servicio gratuito de registro de obras, garantizando la protección legal de la propiedad intelectual asociada al desarrollo de la aplicación.

Marco Regulatorio TIC

La Ley 1341 de 2009, actual marco general del sector de las Tecnologías de la Información y las Comunicaciones establece los principios generales que rigen el desarrollo y operación de servicios digitales en Colombia. Esta legislación define el contexto regulatorio para el desarrollo de aplicaciones web y servicios digitales.

La normatividad TIC colombiana también incluye consideraciones específicas para el desarrollo de software empresarial y aplicaciones comerciales, estableciendo requisitos de calidad, seguridad y accesibilidad que deben considerarse en el proceso de desarrollo.

Consideraciones de Licenciamiento

la Ley 603 de 2000 obliga a las empresas a declarar si los problemas de software son o no legales, estableciendo responsabilidades específicas para el uso y distribución de software. Este marco legal requiere que los desarrolladores consideren aspectos de licenciamiento tanto para las herramientas de desarrollo utilizadas como para los componentes de terceros integrados en la aplicación.

El cumplimiento de las normativas de licenciamiento es fundamental para evitar conflictos legales y garantizar la operación legítima de la aplicación desarrollada, especialmente cuando se integran librerías, frameworks o servicios de terceros**.**

5. Metodología

Para el desarrollo de este proyecto de Sistema de gestión y control de rutinas de entrenamiento, se decidió utilizar la metodología Scrum. Esta metodología ágil facilita el trabajo en equipo, permite obtener feedback continuo y ofrece flexibilidad para adaptarse a cambios en los requerimientos de gestión de entrenamientos. Scrum se centra en la entrega incremental de valor a los usuarios del sistema, logrando mejoras constantes en cada iteración a través de sprints enfocados en funcionalidades específicas como autenticación, gestión de espacios, sistema de reservas y panel administrativo.

La implementación con tecnologías modernas como React para la interfaz de usuario, TypeScript para mayor robustez del código, postgreSQL y prisma para la gestión de base de datos y autenticación, junto con Tailwind CSS para el diseño responsivo, permite un desarrollo ágil y escalable que se alinea perfectamente con los principios de Scrum, facilitando la colaboración, transparencia y adaptación continua del sistema a las necesidades de la comunidad.

5.1. Composición de la Metodología

• **Planificación del Sprint**: En cada sprint, el equipo realiza una planificación inicial donde se eligen las historias de usuario prioritarias que se desarrollarán. Se establecen objetivos claros para el sprint, alineados con las necesidades del cliente.

• **Reuniones Diarias (Daily Scrum):** El equipo se reúne brevemente cada día para revisar el progreso, compartir bloqueos y asegurar que todos estén alineados con los objetivos del sprint.

• **Revisión del Sprint:** Al finalizar cada sprint, se realiza una revisión donde se presenta el incremento del producto a los stakeholders, quienes ofrecen feedback sobre las funcionalidades implementadas.

• **Retrospectiva del Sprint:** Tras la revisión del sprint, el equipo evalúa el proceso de trabajo, identificando áreas de mejora para implementar en el próximo sprint, mejorando continuamente la eficiencia y la colaboración.

5.2. Planificación de roles

La planificación de roles en Scrum permite distribuir responsabilidades y asegurar que cada miembro del equipo contribuya al éxito del proyecto. Los roles clave son los siguientes:

- **Product Owner:** Responsable de definir las historias de usuario y priorizarlas en el backlog del producto, asegurando que el equipo trabaje en los elementos que entregan más valor al cliente. Este rol también recibe el feedback y lo traduce en nuevos requerimientos o ajustes.
- **Scrum Máster:** Facilita el proceso Scrum, asegura la eliminación de impedimentos y se enfoca en mantener el equipo alineado con las prácticas ágiles. Ayuda a optimizar la colaboración y guía al equipo hacia una mejora continua.
- **Equipo de Desarrollo - Samuel Vergara, Iván Morales, Carlos Perdomo:** Son responsables de implementar las historias de usuario en código, asegurando que cumplan con los requerimientos funcionales y técnicos.
- **Pruebas (Testing) - Samuel Vergara, Iván Morales, Carlos Perdomo.** Ambos miembros son responsables de realizar pruebas de calidad en cada incremento, asegurando que las funcionalidades cumplan con los estándares de calidad y funcionen como se espera.
- **Documentación - Samuel Vergara, Iván Morales, Carlos Perdomo:** También son responsables de crear y mantener la documentación actualizada y clara sobre el progreso y los entregables del proyecto.

5.3 Product Backlog

Historia de Usuario	Prioridad
HU - 1. Como visitante sin cuenta, **quiero** registrarme con un nombre de usuario y contraseña válidos, **para** acceder al contenido personalizado de la aplicación	Alta
HU - 2. Como usuario registrado, **quiero** iniciar sesión y mantener mi sesión activa, **para** no tener que introducir mis credenciales en cada visita	Alta
HU - 7. Como administrador, **quiero** mantener la biblioteca de ejercicios con descripciones, alias y videos, **para** ofrecer referencias fiables al resto de usuarios	Alta
HU - 3. Como usuario registrado, **quiero** crear, buscar, actualizar, duplicar o eliminar las rutinas, **para** organizar mis entrenamientos según mis objetivos	Alta
HU - 4. Como usuario en sesión, **quiero** iniciar una rutina, controlar descansos y registrar cada serie, **para** guardar el progreso de mis entrenamientos	Media
HU - 5. Como usuario activo, **quiero** revisar mis entrenamientos anteriores y profundizar en su detalle, **para** analizar mi progreso.	Media
HU - 6. Como usuario activo, **quiero** ver un resumen de mis métricas recientes y accesos rápidos, **para** monitorear mi avance y volver a entrenar rápidamente	Baja

5.4. Sprint Backlog

Sprint 1 (Semanas 1-2) – Autenticación y perfiles

Objetivo: Habilitar el ingreso a la plataforma y los flujos básicos de gestión de usuarios.

Historias de usuario:
HU1 (registro de cuenta), HU2 (inicio de sesión y sesión persistente).

Sprint 2 (Semanas 3-4) – Catálogo de ejercicios

Objetivo: Desarrollar el CRUD completo de ejercicios con soporte multimedia y panel administrativo.

Historia de usuario:
HU3 (gestionar biblioteca de ejercicios).

Sprint 3 (Semanas 5-6) – Gestión de rutinas

Objetivo: Permitir a los usuarios construir y mantener rutinas basadas en el catálogo de ejercicios.

Historia de usuario:
HU4 (crear, consultar, editar, duplicar y eliminar rutinas).

Sprint 4 (Semanas 7-8) – Registro de entrenamientos

Objetivo: Capturar la ejecución de rutinas, registrar series y gestionar el historial de entrenamientos.

Historias de usuario:
HU5 (iniciar rutina y registrar series), HU6 (registrar entrenamientos recientes).

Sprint 5 (Semanas 9-10) – Estadísticas y reportes

Objetivo: Proveer visualizaciones e indicadores que muestren el progreso del usuario y consoliden la información registrada.

Historia de usuario:
HU7 (consultar métricas y accesos rápidos al progreso).

6. Resultados

6.1. Descripción de Requerimientos

Requerimientos Funcionales

Nombre	**RF1.** Registro de Usuarios
Resumen	Permite a los usuarios registrarse mediante nombre de usuario y contraseña
Entradas	Nombre de usuario, Contraseña
Resultados	Usuario persistido en la base de datos y sesión guardada.

Nombre	**RF2.** Inicio y cierre de sesión
Resumen	Permite a los usuarios iniciar sesión y navegar en la aplicación
Entradas	Nombre de usuario, Contraseña
Resultados	Sesión activa con perfil cargado al iniciar

Nombre	**RF3.** Gestión de rutinas
Resumen	Listar, buscar, crear, editar, duplicar y eliminar rutinas por usuario autenticado
Entradas	Nombre de rutina, lista de ejercicios con sus parámetros (reps, series, etc.)
Resultados	Rutina Creada, lista de rutinas actualizada

Nombre	**RF4.** Inicio de entrenamientos
Resumen	Los usuarios deben poder registrar entrenamientos completados, incluyendo todos los sets realizados con peso y repeticiones.
Entradas	Fecha/hora de inicio y finalización, series, pesos, reps, etc.
Resultados	Entrenamiento registrado

Nombre	**R5.** Historial y detalle de entrenamientos
Resumen	Los usuarios deben poder consultar su historial completo de entrenamientos con filtros y búsqueda.
Entradas	Fecha/hora de inicio y fin
Resultados	Lista de entrenamientos del usuario ordenados cronológicamente,

Nombre	**RF6.** Panel de control
Resumen	Los usuarios deben poder ver métricas agregadas y accesos rápidos en el home
Entradas	Estadísticas, accesos directos
Resultados	Tarjetas con totales/medias, listados de recientes y enlaces directos para crear rutinas, entrenar y revisar historial.

Nombre	**RF7.** Administración de ejercicios
Resumen	Ofrecer a administradores la gestión completa del catálogo de ejercicios
Entradas	Datos del ejercicio (nombre, descripción, alias), archivos de video o URL, parámetros de búsqueda.
Resultados	Ejercicios creados/actualizados, videos almacenados

Nombre	**RF8.** Búsqueda de Ejercicios
Resumen	El sistema debe permitir buscar ejercicios por nombre o cualquiera de sus alias
Entradas	Término de la búsqueda
Resultados	Lista de coincidencias de ejercicios

Nombre	**RF9.** Reproducción de Videos
Resumen	El sistema debe permitir la reproducción de videos durante el entrenamiento

Entradas	
Resultados	Vista previa del video

Requerimientos no funcionales

RNF – 1 Seguridad: Contraseñas Encriptadas**,** vistas protegidas para adminis

RNF – 2 Escalabilidad

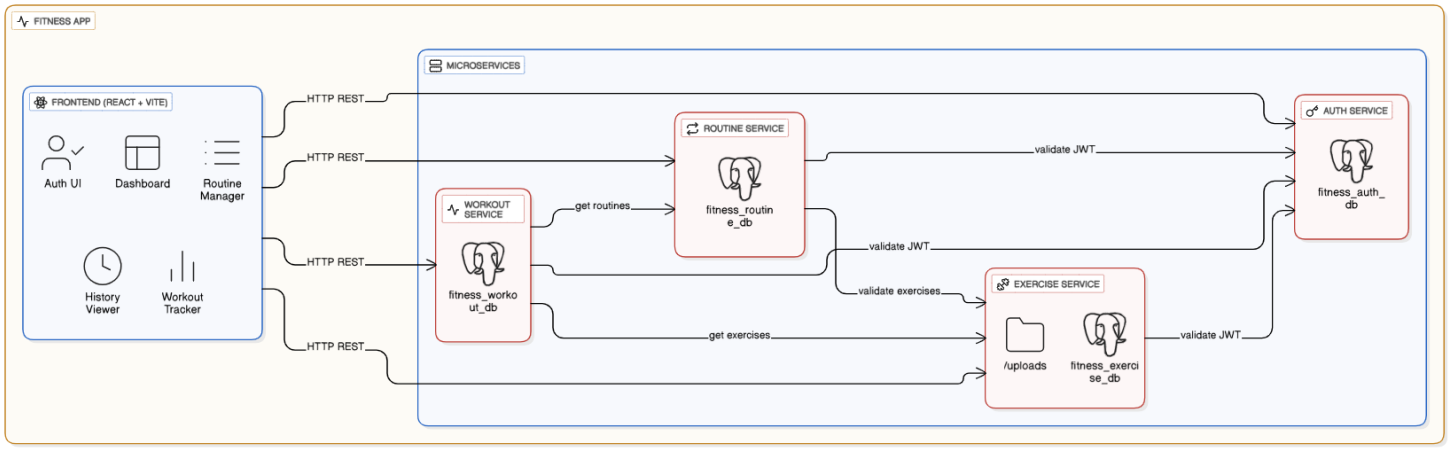
RNF – 3 Usabilidad: Interfaz responsive, manejo de errores, navegación intuitiva

RNF – 4 Concurrencia: Múltiples usuarios simultáneos sin conflictos

RNF – 5: Tolerancia a fallos: Manejo de excepciones y mensajes de error claros

6.2. Modelado

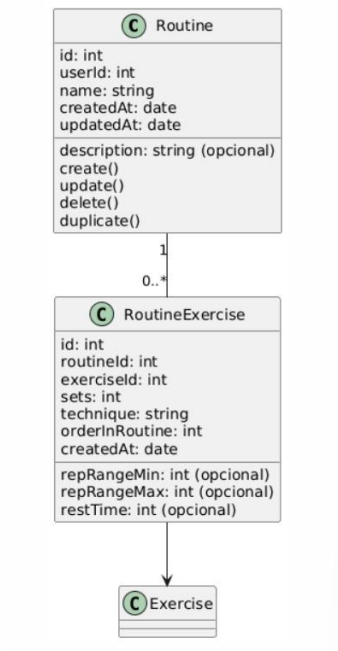
Diagrama de Arquitectura y Componentes



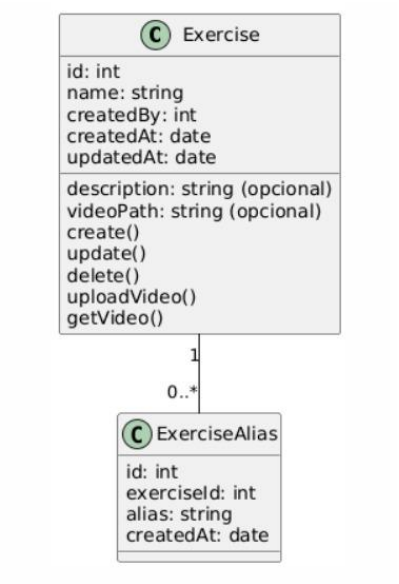
Diagramas de Clases

Auth-Service

Routine-Service



Exercise-Service



Workout-Service

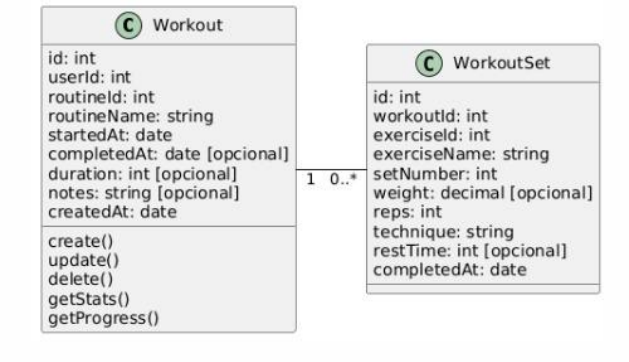
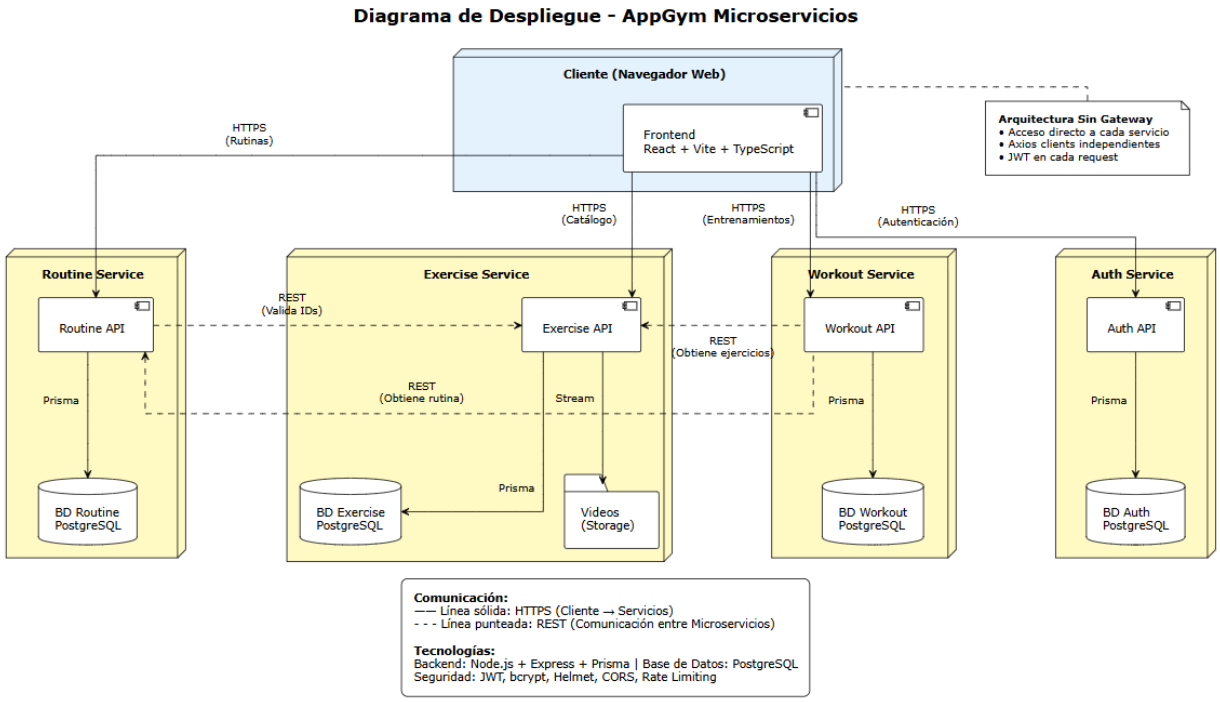
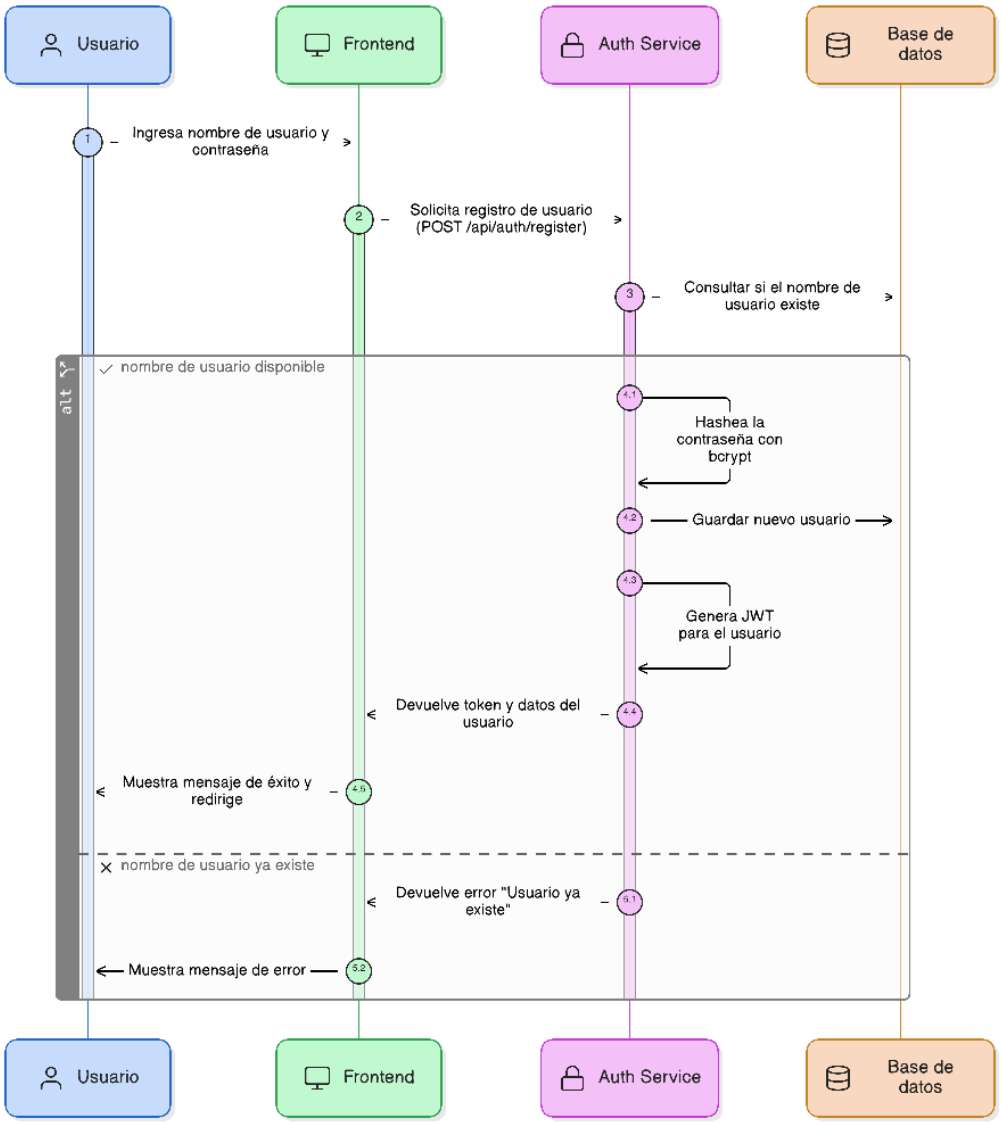


Diagrama de Despliegue



Diagramas de secuencia

Registro de usuario



Crear Rutina

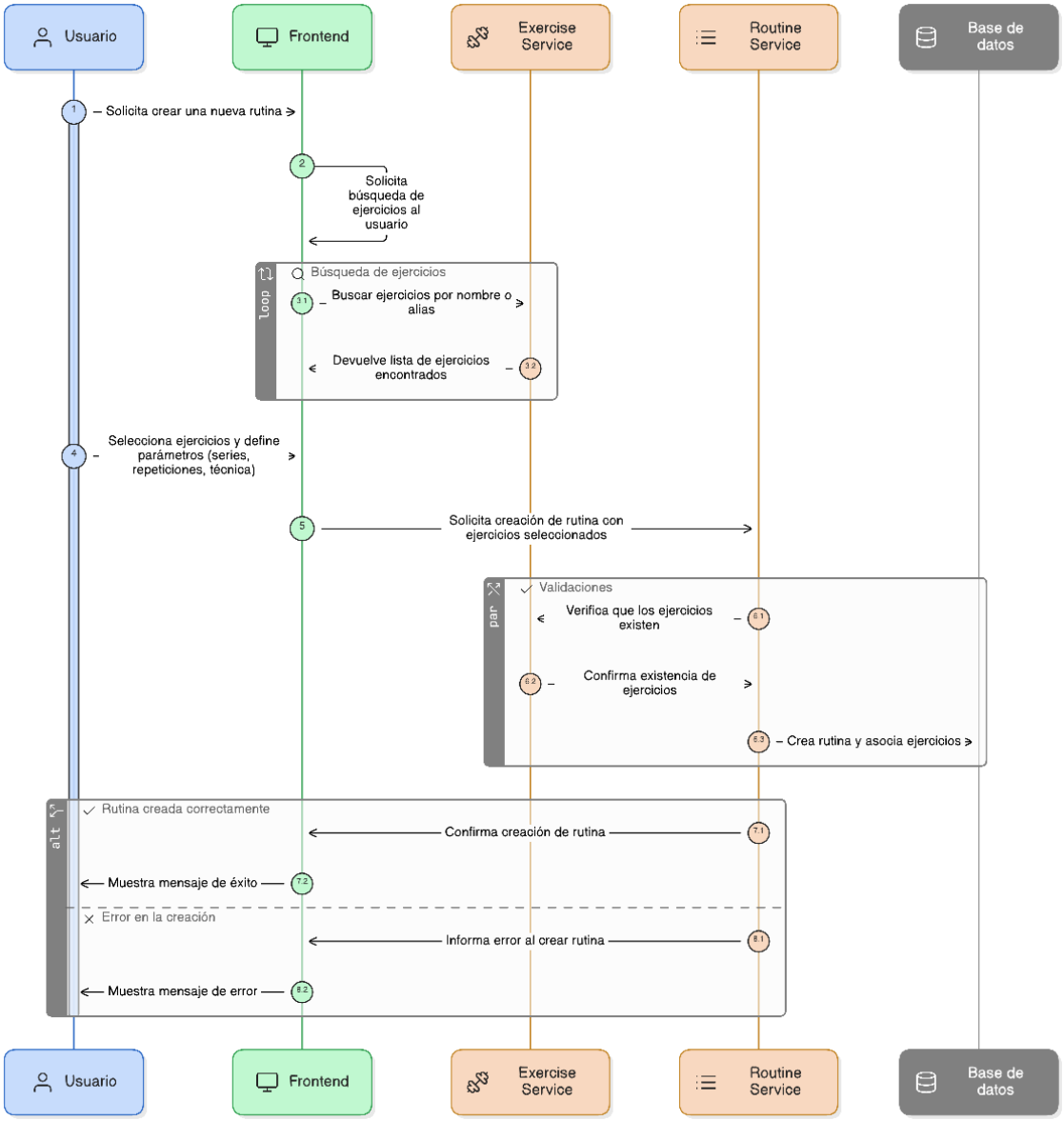
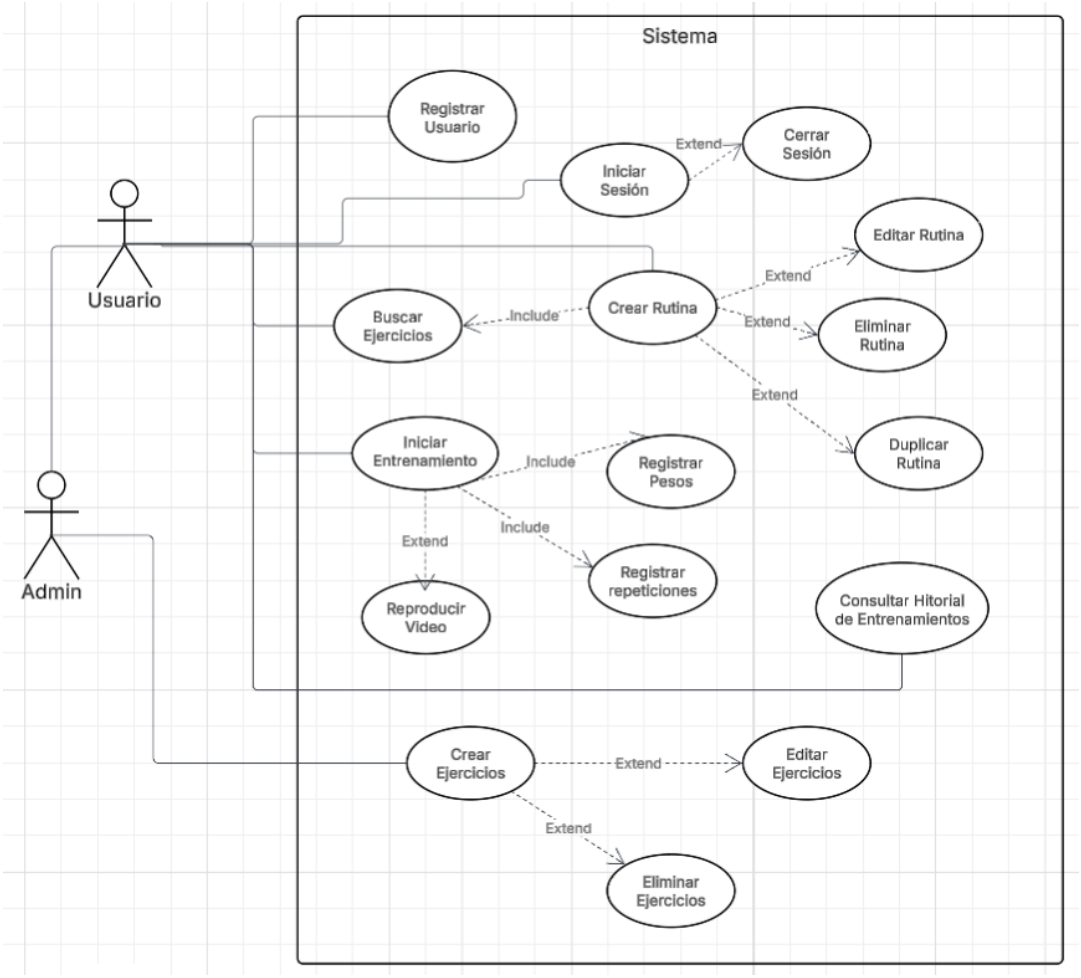


Diagrama de Casos de Uso



Descripción de los Casos de Uso

Caso de Uso	**Registrar Usuario**
Autores	Usuarios
Descripción	El sistema se debe comportar como se describe en el siguiente caso de uso cuando un usuario decida registrase a la app
Precondición	El usuario debe disponer Información para crear una cuenta y poder acceder al sistema.

Secuencia normal	**Paso**	**Acción**
	1	El Usuario busca en la web la app
	2	El sistema le solicita que ingrese la información de registro
	3	El usuario ingresa los datos solicitados
	4	El sistema guarda los datos ingresados
	5	El sistema lo redirige a la página de inicio de la app
Excepciones	**Paso**	**Acción**
	5	Si alguno de los datos no cumple las especificaciones o las contraseñas no coinciden, el sistema muestra un mensaje de error. **Ir al paso 2**
Postcondición	Si la información ingresada cumple las condiciones, el usuario queda registrado y accede a la pantalla de inicio de la app	
Importancia	Vital	

Caso de Uso	**Iniciar Sesión**	
Autores	Administradores y Usuarios	
Descripción	El sistema permite a los usuarios autenticarse en la aplicación mediante el ingreso de sus credenciales de acceso (nombre de usuario y contraseña).	
Precondición	- El usuario debe estar registrado en el sistema - El usuario debe conocer su nombre de usuario y contraseña	
Secuencia normal	**Paso**	**Acción**
	1	El usurario solicita entrar en la aplicación
	2	El sistema le solicita que ingrese la Username y Contraseña
	3	El usuario ingresa los el Username y la Contraseña

	4	El sistema comprueba los datos ingresados
	5	Si los datos son correctos el sistema lo dirige a la página de inicio de la aplicación
Excepciones	**Paso**	**Acción**
	5	Si el username o la contraseña no son correcto, el sistema mostrara un mensaje de error. **Ir al paso 2**
Postcondición	Si la información ingresada es correcta, el usuario accede con la sesión iniciada a la aplicación	
Importancia	Vital	

Caso de Uso	**Crear Ejercicio**	
Autores	Administradores	
Descripción	El sistema se debe comportar como se describe en el siguiente caso de uso cuando un Admin registre un nuevo ejercicio.	
Precondición	- El Admin debe estar registrado en el sistema - El Admin debe tener sesión iniciada	
Secuencia normal	**Paso**	**Acción**
	1	El admin inicia sesión en el sistema
	2	El adimin se dirige a la sección “Panel Admin”
	3	El admin escoge la opción “Crear Ejercicio”
	4	El sistema dispone formulario para ingresar info del nuevo ejercicio
	5	El admin ingresa la info del ejercicio
	6	El sistema comprueba y guarda la información
Excepciones	**Paso**	**Acción**

	6	Si algun dato ingresado no cumple condiciones, se muestra mensaje de error. **Ir al paso 5**
Postcondición	Si la información ingresada es correcta se crea un nuevo ejercicio disponible para buscar y seleccionar	
Importancia	Vital	

Caso de Uso	**Crear Rutina**	
Autores	Todos los Usuarios	
Descripción	El sistema se debe comportar como se describe en el siguiente caso de uso cuando un Admin registre un nuevo lugar de adopción.	
Precondición	- Deben estar registrados en el sistema - Deben tener sesión iniciada - Debe haber ejercicios creados	
Secuencia normal	**Paso**	**Acción**
	1	Se inicia sesión en el sistema
	2	Se dirige a la sección “Rutinas”
	3	Se escoge la opción “Crear Rutina”
	4	El sistema dispone formulario para ingresar info de la nueva rutina
	5	Se ingresa la info de la rutina
	6	El sistema comprueba y guarda la información
Excepciones	**Paso**	**Acción**
	6	Si algún dato ingresado no cumple condiciones, se muestra mensaje de error. **Ir al paso 5**

Postcondición	Si la información ingresada es correcta se crea una nueva rutina disponible para ver y seleccionar
Importancia	Vital

Caso de Uso	**Iniciar Entrenamiento**	
Autores	Todos los usuarios	
Descripción	El sistema se debe comportar como se describe en el siguiente caso de uso cuando un Usuario inicie un entrenamiento.	
Precondición	- El usuario debe estar registrado en el sistema - El usuario debe tener sesión iniciada	
Secuencia normal	**Paso**	**Acción**
	1	El usuario inicia sesión en el sistema
	2	El usuario se dirige a la sección “Rutinas”
	3	El usuario escoge la rutina a realizar
	4	El usuario da click en iniciar entrenamiento
	5	El usuario registra la info durante su entrenamiento
	6	El usuario guarda su entrenamiento con los registros
Excepciones	**Paso**	**Acción**
	6	Si algun dato ingresado no cumple condiciones, se muestra mensaje de error. **Ir al paso 5**
Postcondición	Si la información ingresada es correcta se guarda un nuevo entrenamiento disponible para consultar en el historial	
Importancia	Vital	

6.3. Diseño de Interfaz

6.4. Descripción Técnica del Sistema

Arquitectura general: La aplicación adopta una arquitectura de microservicios compuesta por un frontend React y cuatro servicios independientes (autenticación, ejercicios, rutinas y entrenamientos), cada uno publicado en su propio puerto y respaldado por una base de datos PostgreSQL. Esta estructura se apoya en Node.js/Express en el backend y Vite + React + TypeScript + Tailwind CSS en el frontend.

Microservicios backend: Cada servicio Express aplica middleware de seguridad, expone un endpoint de health check y arranca tras establecer conexión con PostgreSQL mediante Prisma. Las dependencias comunes incluyen express, cors, dotenv y @prisma/client, con scripts de inicio y nodemon para desarrollo.

- **Auth Service:** Gestiona registro, login y verificación de tokens JWT, usando bcrypt para proteger contraseñas y Prisma para persistir usuarios
- **Exercise Service:** Implementa CRUD de ejercicios con alias, búsquedas y subida de vídeos, manteniendo los datos en PostgreSQL.
- **Routine Service:** Orquesta rutinas del usuario, valida la existencia de ejercicios mediante llamadas HTTP al Exercise Service y asegura la consistencia de parámetros (series, rangos, orden).
- **Workout Service:** Registra sesiones, series y métricas agregadas, ofreciendo estadísticas por ejercicio, para validar rutinas y ejercicios se comunica con los servicios respectivos.

Persistencia y datos: Cada microservicio define su propio esquema Prisma apuntando a PostgreSQL, lo que permite migraciones aisladas y relaciones con eliminación en cascada (por ejemplo, ejercicios y alias, rutinas y ejercicios asociados, entrenamientos y series)

Frontend web: Se construye con Vite + React + TypeScript, incorporando React Router para navegación y Tailwind para estilos, además de utilidades como date-fns para manejar las fechas y lucide-react para el uso de iconos. La jerarquía de rutas centraliza la protección de pantallas mediante un AuthProvider contextual y un componente ProtectedRoute que aplica redirecciones y control de rol administrador.

Integración y comunicaciones. El frontend consume directamente los endpoints de cada microservicio, enviando el JWT emitido por el Auth Service, internamente servicios como Routine y Workout consultan a Exercise (y viceversa) mediante clientes Axios (llamadas HTTP) configurables por URL, posibilitando validaciones cruzadas sin un API Gateway centralizado.

7. Análisis y Discusión

7.1. Dificultades Encontradas y Soluciones

Problema: Cuando un usuario completa un entrenamiento, necesitamos guardar información sobre la rutina y los ejercicios que hizo. El problema es: ¿qué pasa si después el usuario cambia el nombre de una rutina o elimina un ejercicio? No queremos que su historial de entrenamientos se vea afectado o pierda información.

Solución: Se decidió guardar copias de la información importante (como nombres de rutinas y ejercicios) directamente en el workout-service. Así, aunque el usuario modifique o elimine rutinas originales, su historial queda intacto con los nombres que tenían en ese momento. Además, cuando se crea un nuevo entrenamiento, el workout-service verifica con los otros servicios que los ejercicios y rutinas existen, usando peticiones HTTP simples con axios.

7.2. Efectividad Herramientas Utilizadas

Prisma ORM: Fue muy útil para manejar las cuatro bases de datos PostgreSQL separadas. Nos permitió definir las tablas y sus relaciones de forma clara. Las migraciones automáticas facilitaron mucho mantener las bases de datos actualizadas.

Express.js: Perfecto para crear las APIs de cada servicio. Usamos varios plugins que nos dieron funcionalidades importantes con poco código: helmet para seguridad básica, cors para permitir peticiones desde el frontend y express-validator para verificar que los datos que llegan sean correctos.

Axios para la comunicación entre servicios: Funcionó bien para hacer peticiones HTTP entre servicios. Por ejemplo, cuando el routine-service necesita verificar que un ejercicio existe, simplemente hace una petición al exercise-service.

React con TypeScript y Tailwind CSS: Esta combinación fue muy productiva para el frontend. TypeScript nos ayudó a evitar errores al trabajar con los datos de cada servicio. Tailwind CSS hizo que diseñar la interfaz fuera rápido sin escribir mucho CSS personalizado. Usamos react-router-dom para las rutas protegidas y el Context API para manejar la autenticación de forma centralizada.

7.3. Aspectos No Terminados o No Implementados

Gateway API centralizada: El frontend se conecta directamente a cada servicio en su puerto específico (3001, 3002, 3003, 3004).

8. Conclusiones

Se logró implementar exitosamente cuatro servicios independientes (auth-service, exercise-service, routine-service y workout-service), cada uno con su propia base de datos PostgreSQL y lógica de negocio aislada. Esta arquitectura permitió que cada servicio se desarrollara, desplegara y mantuviera de forma autónoma, cumpliendo con el principio de separación de responsabilidades.

Las funcionalidades de planificación y seguimiento de rutinas personalizadas se implementaron exitosamente con bases de datos independientes para cada microservicio, asegurando la integridad y autonomía de los datos. El uso de Prisma como ORM permitió definir esquemas específicos con relaciones claras y migraciones aisladas, facilitando la evolución independiente de cada servicio

El módulo de historial de entrenamientos (workout-service) implementó exitosamente patrones de comunicación entre servicios que garantizan la consistencia de datos temporales. La estrategia de almacenar copias de información crítica (nombres de rutinas y ejercicios) directamente en el workout-service demostró ser una solución efectiva para preservar la integridad histórica de los entrenamientos, incluso cuando los usuarios modifican o eliminan rutinas originales

REFERENCIAS

Atlassian. (s.f.). Arquitectura de microservicios. *Atlassian*. https://www.atlassian.com/es/microservices/microservices-architecture

Atlassian. (s.f.). Cinco ventajas de los microservicios (y algunas desventajas). *Atlassian*. https://www.atlassian.com/es/microservices/cloud-computing/advantages-of-microservices

Altova. (s.f.). Diagramas UML. En *UModel 2024*. https://www.altova.com/manual/es/UModel/umodelbasic/uml_diagrams.html

Amazon Web Services. (2025). Microservicios. *AWS*. https://aws.amazon.com/es/microservices/

Decide Soluciones. (2022, 20 de abril). Arquitectura de microservicios: qué es, ventajas y desventajas. *Decide Soluciones*. https://decidesoluciones.es/arquitectura-de-microservicios/

Intel. (s.f.). Qué son los microservicios y la arquitectura de microservicios? *Intel*. https://www.intel.la/content/www/xl/es/cloud-computing/microservices.html

IONOS. (2020, 23 de septiembre). Diagrama de componentes: modelado eficiente de sistemas con módulos de software. *IONOS*. https://www.ionos.com/es-us/digitalguide/paginas-web/desarrollo-web/diagrama-de-componentes/

Lucidchart. (2020, 13 de mayo). Presentación de los tipos de diagramas UML. *Blog de Lucidchart*. https://www.lucidchart.com/blog/es/tipos-de-diagramas-uml

Miro. (s.f.). Diagrama de componentes UML: Qué es y cómo hacerlo. *Miro*. https://miro.com/es/diagrama/que-es-diagrama-componentes-uml/

Modeling Languages. (2020, 7 de noviembre). Clarifying concepts: MBE vs MDE vs MDD vs MDA. *Modeling Languages*. https://modeling-languages.com/clarifying-concepts-mbe-vs-mde-vs-mdd-vs-mda/

Object Management Group. (s.f.). Model Driven Architecture (MDA). *OMG*. https://www.omg.org/mda/

Pahl, C. (2009, 1 de abril). Model-Driven Engineering (MDE) and Model-Driven Architecture (MDA) applied to the Modeling and Deployment of Technology Enhanced Learning (TEL) Systems: promises, challenges and issues. *HAL Open Science*. https://hal.science/hal-00372442v1

Paradigma Digital. (s.f.). Patrones de arquitectura de microservicios, ¿qué son y qué ventajas…. *Paradigma*. https://www.paradigmadigital.com/dev/patrones-arquitectura-microservicios-que-son-ventajas/

ProcessOn. (s.f.). ¿Qué es un "diagrama de implementación" UML? Tutoriales y casos adjuntos. *ProcessOn*. https://www.processon.io/es/blog/uml-deployment-diagram-tutorials

Red Hat. (s.f.). ¿Qué son y para qué sirven los microservicios? Beneficios de los microservicios. *Red Hat*. https://www.redhat.com/es/topics/microservices

Springer. (s.f.). Model Driven Engineering. En *SpringerLink*. https://link.springer.com/chapter/10.1007/3-540-47884-1_16

Verified Market Reports. (2025, 28 de febrero). Tamaño del mercado de la arquitectura de microservicio, expansión del mercado, tendencias y pronóstico 2032. *Verified Market Reports*. https://www.verifiedmarketreports.com/es/product/microservice-architecture-market/