

Benemérita Universidad Autónoma de Puebla

Facultad de Ciencias de la Computación



T E S I S

“Plataforma web multinivel para la gestión automatizada de planes y servicios en gimnasio”

Presenta: *Geovanni Isai Hernández Andrés*

Para obtener el grado de: Licenciatura en Ingeniería en
Tecnologías de la Información

Director: *María Teresa Torrijos Muñoz*

Codirector: *Juan Carlos Conde Ramírez*

Puebla, Pue, Noviembre 2025

Índice

Índice.....	1
Introducción	3
Capítulo 1.....	4
Antecedentes	4
Planteamiento del problema	8
1.1 Objetivo general	9
1.2 Objetivos específicos	9
1.3 Estado del arte	9
1.4 Metodología	10
Capítulo 2.....	13
Requerimientos	13
2.1 Funcionalidades principales de la aplicación	13
2.2 Requisitos de usuario	13
2.3 Información a recopilar y almacenar	16
2.4 Integración con dispositivos y servicios externos	17
2.5 Aspectos de seguridad	18
2.6 Requisitos técnicos:	21
2.6.1 Javascript	21
2.6.2 TypeScript	22
2.6.3 Node.js y Express	23
2.6.4 PostgreSQL.....	25
2.6.5 Sequelize ORM	25
2.6.6 React.....	26
2.6.5 TailwindCSS.....	29
Capítulo 3: Análisis y diseño	30
3.1 Revisión de requerimientos:	30
3.2 Definición de la arquitectura	58
3.2.1 Cliente - Servidor	58
3.2.2 Clean architecture (backend).....	59
3.3 Diseño de interfaces de usuario:	61
3.3.1 Autenticación.....	61
3.3.2 Dashboard - Administrador	63
3.3.3 App Cliente.....	67
3.3.4 App Entrenador	68
3.4 Diseño de la base de datos	69
3.4.1 Tablas	69
3.4.2 Implementación con Sequelize	71
3.6 Planificación del desarrollo (planeación de sprints):	73
Capítulo 4: Construcción	77
4.1 Estructura de Clean Architecture en el Backend	77
4.2 Flujo de backend con clean architecture	78

4.3 Uso de react router	80
4.4 Creación de componentes en React.js	81
4.5 Consumir backend desde frontend	81
Capítulo 5: Pruebas y liberación	82
Entorno de desarrollo (DEV).....	83
Entorno de calidad (QA)	84
Entorno de producción (PRD).....	85
Conclusiones y trabajos futuros	86
Conclusiones	86
Trabajos a futuro	86
Referencias.....	87

Introducción

En los últimos años se ha vuelto común que las personas tomen más conciencia sobre el cuidado en su salud física, mental y estado corporal, ya que la preocupación por las enfermedades no transmisibles como las cardio-vasculopatías, cáncer, diabetes, depresión y ansiedad ha crecido drásticamente durante esta última década. Esto debido al estilo de vida sedentario al que se enfrenta la mayoría de personas alrededor del mundo. (World Health Organization: WHO, 2024)

La activación física se ha consolidado como un pilar fundamental para mejorar la calidad de vida de las personas, no solo por sus beneficios fisiológicos, sino también por su impacto en el bienestar psicológico. Sin embargo, alcanzar este objetivo requiere de un enfoque multidisciplinar que integre: planificación nutricional, entrenamiento personalizado, protocolos de recuperación muscular, etc. Debido a que no todas las personas poseemos conocimientos concretos de cada disciplina necesaria para cuidar nuestra salud los gimnasios han tomado un papel importante ofreciendo una plantilla de trabajadores expertos en estos temas para acompañar a las personas en su proceso de conseguir ese objetivo.

La participación de los gimnasios como Centros de Activación Física (CAF) es vital, ya que dentro existen servicios que los brindan el personal especializado en cada uno de los mismos para alcanzar los objetivos del cliente. Sin embargo, no todas las personas tienen la misma condición física ni la misma meta, para ello es necesario tener un registro de información de cada persona, lo cual ayuda al cliente y los especialistas a llevar un control más de cerca sobre lo que debe hacer, cambiar y lo que está funcionando para cumplir sus objetivos. (Scarpetta et al., 2012, p. 15)

El uso de sistemas de software en dichos establecimientos ha guiado a la creación de software que controlen la gestión de los clientes y personal dentro del gimnasio. La gran mayoría de estos sistemas se centran en operaciones financieras, teniendo como prioridad la ganancia económica de los negocios, pero dejando a un lado lo que realmente importa, el acompañamiento correcto de sus clientes sobre los servicios y objetivos recabando información crítica para formar un perfil al que debe adaptarse cada servicio que se oferta dentro del establecimiento.

Capítulo 1

En este primer capítulo se establecen los fundamentos de esta investigación. Se inicia con los antecedentes y el planteamiento del problema para conocer el contexto de los distintos sistemas usados en gimnasios. Una vez teniendo el contexto se definen los objetivos que guían el trabajo y se describe la metodología que se plantea seguir durante el desarrollo del prototipo.

Antecedentes

La digitalización de los gimnasios inició en la década de 1990 con la llegada de sistemas de software de gimnasios para sustituir la gestión basada en sistemas manuales y registros en papel, pero muchos de estos sistemas solo se enfocan en la gestión de acceso y registro de información personal de los clientes. Sin embargo, fue hasta 2010 que surgieron plataformas como MindBody las cuales incorporan los límites de acceso a servicios según el plan de pago que ha escogido el cliente. Estos sistemas suelen ser eficientes, pero resultan ser costosos para gimnasios independientes lo cual los lleva a optar por soluciones baratas como utilizar otros servicios externos para la gestión de los servicios. (Techfit, 2023)

En el estudio que realizó Foroughi B. sobre centros de fitness en Malasia analiza las intenciones de comportamiento, el deleite y la satisfacción, refiriéndose a la satisfacción como las experiencias acumuladas de los clientes con el gimnasio y sus servicios, aunque también señalan que es la sensación placentera que se produce al comparar las expectativas con el rendimiento del servicio lo que causa la satisfacción. (Baena-Arroyo et al., 2020)

Por lo tanto, la experiencia de servicio se ha convertido en un factor clave en la satisfacción del cliente y el comportamiento que adopte en el futuro, es decir, dependiendo de la calidad del servicio que reciba el cliente mostrará diferente motivación y empeño a cumplir sus objetivos. Se ha hecho notorio que los usuarios abandonan los gimnasios al no ver resultados y esto va de la mano con el mal manejo del progreso o el seguimiento personalizado.

Se realizó una investigación sobre los diferentes softwares disponibles para control de servicios y gestión de usuarios de gimnasios y estos son los principales sistemas existentes hasta el día de hoy.

ABC GLOFOX

Es uno de los softwares más utilizados hoy en día para la gestión de gimnasios, pero solo se enfoca en el negocio para llevar un control de las membresías, ventas y clases que ofrece como

parte de sus servicios. El usuario cliente también dispone de una aplicación móvil la cual dispone las funciones de agendar una visita a clase, pagar membresías o clases y mantener un registro de esas ventas, pero deja de lado el registro del rendimiento o planes personalizados que debe seguir la persona para generar motivación a seguir manteniendo un estilo de vida sano. (*Welcome to Glofox: The #1 Fitness Management Software, 2025*)

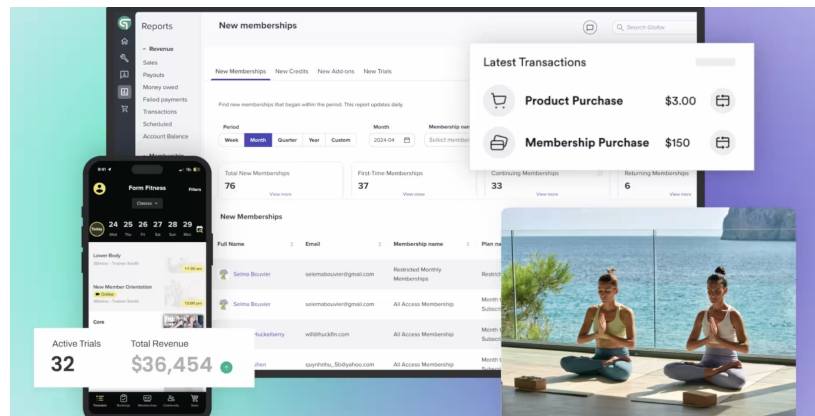


Imagen 1: ABC Glofox

MINDBODY

MindBody es un sistema que automatiza la gestión de reservas de citas en línea para asistir a clases o al gimnasio, al mismo tiempo ofrece herramientas para el staff donde se puede crear una comunidad dentro del sistema para que todos los usuarios se sientan más conectados con la marca y estén informados sobre las clases o áreas de trabajo que se van a impartir en el gimnasio o centro de actividad física ya que igual puede usarse en centros de yoga o alguna otra disciplina. (Playlist, 2025)

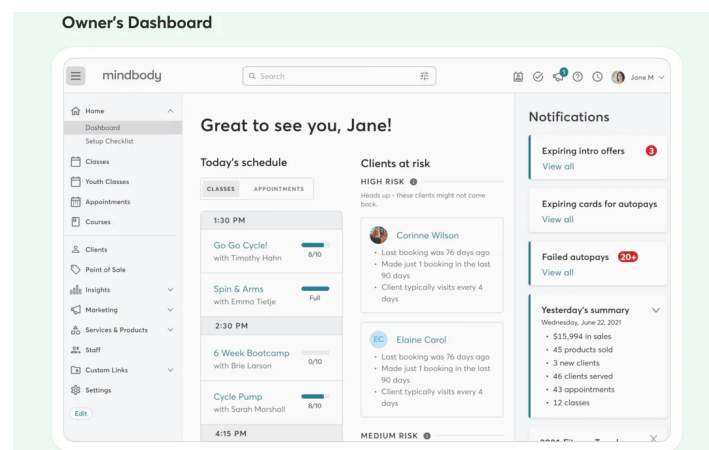


Imagen 2: MindBody

CLUB OS

Este sistema se centra en la retención y ventas del gimnasio con el control de acceso el cual es posible mediante una aplicación móvil la cual sirve también para mantener a los clientes al tanto de anuncios del gimnasio para poner en práctica estrategias de marketing que haga que si un usuario abandonó el gimnasio desee volver y adquirir una membresía desde ahí. (ClubOS, n.d.)



Imagen 3: Club Os

VIRTUAL GYM

A diferencia de otros sistemas este nos ofrece 3 opciones a escoger, cada versión cuenta con funciones diferentes ya que va dirigido a un público específico. Las categorías que existen son: empresas de fitness, bienestar corporativo y uso personal.

- Empresas de fitness: se refiere a un sistema que automatiza la gestión de todas sus operaciones dentro de negocios de fitness como gimnasios, centros de yoga, etc. Pero no se ofrecen servicios tan personalizados a los clientes de los mismos centros de actividad física.
- Bienestar corporativo: va dirigido al personal de las empresas para promover una vida saludable, dando dietas personalizadas y rutinas que son otorgadas por una IA o personal directo de la empresa.
- El sistema de uso personal: es una aplicación que ofrece catálogo de ejercicios, dietas según tus métricas, etc.

Como se puede notar es un sistema muy completo, pero dentro de la gestión de gimnasio aún no contiene el servicio personalizado a los clientes y es muy costoso para que lo implementen negocios pequeños. (Virtuagym, n.d.)

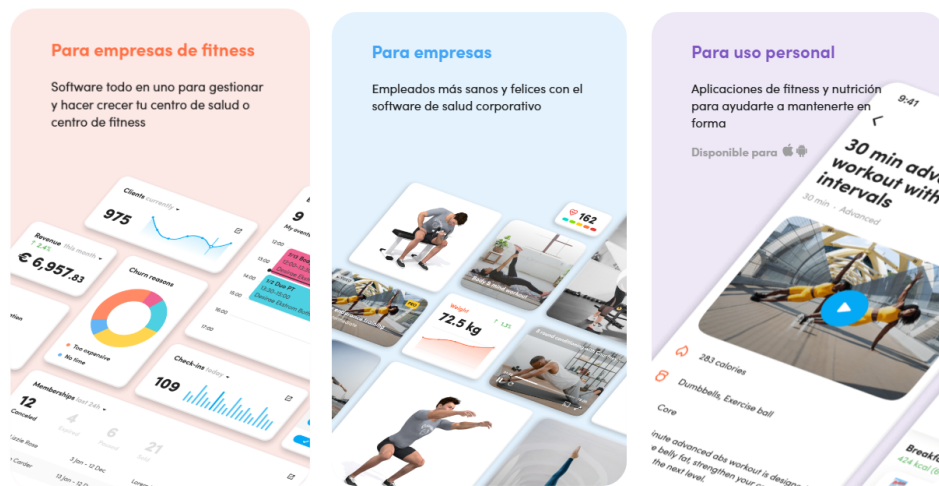


Imagen 4: Virtual Gym

Centrándonos en el mercado de México se encontró que la mayoría de los sistemas que se usan en estos establecimientos son personalizados y sólo abarca la parte de la gestión de las membresías y accesos, algunos otros ofrecen seguimiento de rutinas y dietas, pero no controla estos beneficios desde una suscripción o membresía, por lo tanto, la gestión se vuelve más compleja. (Urbina, 2024)

Esto refuerza la necesidad de desarrollar prototipos escalables que integren seguimiento al desarrollo de nuevas funcionalidades que aborden sus servicios sin elevar costos operativos y evitar la necesidad de adquirir un software comercial de costo elevado.

Planteamiento del problema

Aunque los gimnasios requieren sistemas que puedan vincular suscripciones/membresías con servicios personalizados para retener clientes, casi el 70% de establecimientos utilizan software que no les permite esta integración porque se enfocan más en el acceso y registro de ventas que en los servicios personalizados. Esto genera dos problemáticas: 1) usuarios insatisfechos al no acceder a sus servicios desde una plataforma que centralice todo su entorno de gimnasio, y 2) la gestión operativa ineficiente al depender de software externo u otras plataformas.

Al considerar estos problemas notamos que muchos usuarios abandonan estos centros de activación física al no estar satisfechos con sus resultados y muchos de ellos pierden motivación para seguir con su estilo de vida saludable.

Si bien un estilo de vida saludable ofrece diversos beneficios para la salud, el incumplimiento de los cambios recomendados en el estilo de vida sigue siendo un obstáculo frecuente y complejo para alcanzarlos. La terapia conductual puede mejorar la adherencia al proceso que debe seguir cada persona para lograr su objetivo. (Stonerock & Blumenthal, 2016)

Por lo tanto, las personas consideran los gimnasios como un lugar apto para que los guíe en las medidas que deben seguir con el tiempo para trazar un camino a seguir hábitos saludables y mejorar su calidad de vida. Sin embargo, el desarrollar un plan o rutina apta para cada persona requiere de tener un registro previo que facilite conocer al cliente, ya que todos son diferentes, cada uno tiene objetivos y métricas personales que se diferencian del resto, por lo que la creación global de planes sin tener en cuenta estos factores puede desencadenar problemas físicos al pasar del tiempo.

La mayoría de gimnasios optan por brindar estos servicios de entrenamiento de manera general ya que el personal de staff como los entrenadores normalmente ofrecen estos servicios personalizados a los clientes de manera externa, pero aquí se encuentra una problemática ya que si con el tiempo el cliente desea cambiar de entrenador dentro del establecimiento se pierde el registro de los entrenamientos que ha llevado y los objetivos que se tienen, lo que conlleva a la pérdida de clientes y en el caso de los clientes los hace querer abandonar este estilo de vida al no obtener un progreso continuo o los cambios deseados.

1.1 Objetivo general

El objetivo general es desarrollar una plataforma web en la cual las personas que asisten a un gimnasio para realizar actividad física tengan la oportunidad de elegir un plan o membresía que les permita gestionar sus entrenamientos mediante una aplicación. Dentro de esa aplicación ellos pueden elegir un entrenador que se ajuste a sus necesidades, dicho entrenador tendrá un registro de objetivos, métricas y anotaciones que sean importantes sobre la salud del cliente para posteriormente otorgarle rutinas que se adapten al perfil creado con los datos que dio el cliente.

Dicha plataforma tiene un impacto positivo en el bienestar social actuando como un puente tecnológico que conecta de manera eficiente, segura y accesible a las personas que quieren

mejorar su salud con profesionales que pueden guiarlas. No se limita a digitalizar la gestión de un gimnasio, sino que cumple con los estándares de cuidado personal, promueve salud preventiva y educa a los clientes, creando un ecosistema sano y sostenible dentro de los gimnasios.

1.2 Objetivos específicos

1. Diseñar una arquitectura optimizada para que la lógica sea usada tanto en web como en móvil sin necesidad de rehacer el trabajo.
2. En el servidor se usará una arquitectura clean code para cumplir con los principios SOLID lo cual sirve para que se continúe con el desarrollo después sin someterse a tanta complejidad al momento de hacer cambios.
3. Utilizar tecnologías modernas para un fácil desarrollo y sea incremental.

1.3 Estado del arte

En la última década la industria del fitness ha experimentado una transformación debido a los cambios tecnológicos que están en constante evolución, estos cambios han sido impulsados por la necesidad de optimizar operaciones, mejorar la experiencia de los usuarios y aprovechar tecnologías emergentes.

Con los antecedentes recabados anteriormente de los sistemas existentes se hizo un análisis de las tecnologías aplicadas y tendencias del desarrollo de software para gimnasios, esto nos dio paso a identificar oportunidades de innovación para el sistema propuesto.

Posteriormente al análisis se pudo notar que la mayoría de estos están más enfocados en la gestión de ventas y accesos de los clientes al gimnasio, incluso una gran parte de ellos utiliza sus plataformas de gestión de membresías como marketing para mantener al cliente interesado en seguir pagando los servicios lo cual es muy útil ya que mantiene la parte de mantener a los clientes motivados a seguir asistiendo a estos centros de actividad física.

Cada uno de estos sistemas cuenta con una interfaz intuitiva que mantiene el fácil uso y adaptabilidad de los usuarios lo cual nos permite tener una noción de cómo distribuir nuestro sistema en el área visual para seguir principios UX.

En las limitaciones podemos destacar que no se cuenta con algoritmos para el registro de los objetivos y métricas de cada persona para la trazabilidad de sus rutinas de entrenamiento. Las

plataformas que lo tienen implementado son de alto costo y deja de lado el ecosistema que se tiene dentro de los establecimientos lo cual no permite que los gimnasios locales gocen de ellas o se vuelve difícil de mantener considerando que están aún en desarrollo.

Esta brecha en la falta de sistemas económicos hechos a la medida para el control y registro de servicios integrados nos permite aportar el desarrollo de un sistema que cubra esta necesidad para mejorar la experiencia de estos centros de activación física.

A lo largo de mi estancia en la carrera universitaria tuve la oportunidad de conocer y utilizar las diferentes tecnologías que se usan en el área del desarrollo de software y con las últimas tendencias se pretende aplicar tecnologías modernas las cuales se han consolidados como las más importantes en la actualidad gracias a su rendimiento y escalabilidad para crear desde prototipos a proyectos reales en producción. Esas tecnologías son React.js por su alto rendimiento para el lado del frontend siguiendo principios UX para que sea intuitiva, en el lado del backend se hará uso de Node.js con Express para la creación de API's REST que manejan la comunicación entre la lógica del negocio y la interfaz. En la gestión de datos se pretende usar PostgreSQL que nos otorga beneficios en cuanto a la robustez que ofrece para operaciones críticas de relaciones entre usuarios y sus registros.

1.4 Metodología

La metodología Scrum es un marco de trabajo ágil de procesos que ha sido usado desde principios de los años 90 para gestionar el desarrollo de productos complejos. Scrum no es solo un proceso o técnica que se usa para construir software, es un marco de trabajo en el cual se pueden emplear varias técnicas y procesos. Esto da pie a la eficacia relativa de las prácticas de gestión de producto y las prácticas de desarrollo de modo que se mejoren incrementalmente. (“¿Qué Es La Metodología Scrum?,” 2025)

Scrum es un proceso empírico, donde las decisiones se basan en la observación, experiencia y la experimentación. Tiene tres pilares que son: transparencia, inspección y adaptación, lo cual sustenta el concepto de trabajo iterativo. Empirismo es como trabajar mediante pequeños experimentos, aprender de ese trabajo y adaptar tanto lo que haces como la forma de hacerlo según sea necesario. (*What Is Scrum?*, n.d.)

Una vez conociendo los pilares en los que se basa Scrum se decidió usar esta metodología para realizar entregas parciales y regulares de nuestro producto final, priorizando los beneficios que

aporta el receptor del proyecto, esto debido a que queremos alcanzar la innovación, competitividad, flexibilidad y productividad.

Scrum se compone de diferentes conceptos importantes los cuales se desglosan a continuación.

Roles

Scrum se estructura de 3 roles fundamentales cada uno con responsabilidades claras:

- **Scrum master:** Actúa como facilitador del proceso Scrum, su función es eliminar obstáculos, asegurar que el equipo siga las prácticas ágiles y fomentar una mejora continua.
- **Product Owner:** Representa la voz del cliente, es responsable de gestionar el product backlog priorizando las funcionalidades en función del valor que aportan al negocio, en este trabajo de tesis es representado totalmente por el asesor.
- **Equipo de desarrollo:** Grupo multidisciplinar encargado de construir el producto. (“¿Qué Es La Metodología Scrum?,” 2025)

En este proyecto el rol de Scrum Master y Product Owner recae en el estudiante con supervisión del director de tesis para validar priorizaciones. El equipo de desarrollo solo se conforma del estudiante que posee conocimiento de las habilidades de desarrollo.

Componentes

- **Product backlog:** lista ordenada de tareas y funcionalidades por desarrollar.
- **Sprint backlog:** conjunto de tareas seleccionadas para desarrollarse en sprint.
- **Incremento:** Resultado del trabajo realizado durante un sprint, que debe ser funcional y estar listo para su entrega. (“¿Qué Es La Metodología Scrum?,” 2025)

Eventos

Scrum se organiza en una serie de eventos regulares o ceremonias que estructuran el trabajo y facilitan la comunicación (*Imagen 5*):

- **Sprint planning:** es la reunión inicial de cada sprint, donde el equipo decide qué tareas del Product Backlog se abordarán, se establece un objetivo claro y se estiman esfuerzos requeridos.

- **Daily Scrum - Daily Stand-up:** Reunión diaria de 15 minutos en la que cada miembro responde a tres preguntas claves que tiene como objetivo alinear al equipo y detectar obstáculos de forma temprana:
 - ¿Qué hice hoy?
 - ¿Qué haré hoy?
 - ¿Hay algún impedimento?
- **Sprint review:** al final de cada sprint se realiza una revisión del trabajo realizado. Se presenta el incremento al Product Owner y otras partes interesadas, quienes dan feedback a futuras iteraciones.
- **Sprint retrospective:** reunión final del sprint donde el equipo reflexiona sobre el proceso, identifica lo que ha funcionado bien y proponer mejoras para el siguiente ciclo, fomentando la mejora continua. (“¿Qué Es La Metodología Scrum?,” 2025)

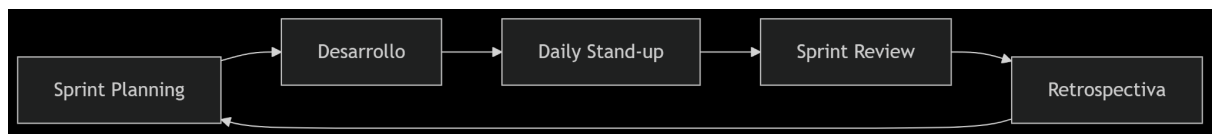


Imagen 5: Proceso scrum

Capítulo 2

En este capítulo se detallan los requerimientos técnicos y funcionales para el desarrollo de la aplicación. Se especifican las funciones principales, requerimientos de usuario y la información que se planea gestionar. De igual manera, se abordan aspectos de seguridad, integración y se justifica el uso del stack tecnológico planeado para llevar a cabo el desarrollo correcto del prototipo.

Requerimientos

2.1 Funcionalidades principales de la aplicación

La funcionalidad principal del sistema es la gestión de rutinas de entrenamiento que serán otorgadas por un especialista dentro del gimnasio, para ello es necesario que el cliente esté registrado en el sistema para conocer sus datos personales y algunas métricas básicas que servirán al entrenador para idear el mejor plan de entrenamiento dependiendo de su condición física.

Cada cliente tiene la posibilidad de crear un perfil previo donde podrá especificar sus objetivos, malestares o notas que necesite tomar en cuenta el entrenador al momento de idear su plan de entrenamiento.

Cada rutina creada debe seguir una estructura apropiada para cada persona evitando el sobre esfuerzo físico por lo que se otorga un límite de ejercicios, repeticiones y series que se pueden asignar en una rutina, de esta manera se procura el bienestar físico y mental de los clientes.

2.2 Requisitos de usuario

La identificación, análisis y gestión de los requerimientos de un sistema en el desarrollo de software es una actividad crucial en el diseño. La gestión de estos requerimientos es una de las causas más frecuentes en el fracaso de proyectos, por lo que es importante entender que son los requerimientos funcionales y no funcionales.

Los **requerimientos funcionales** definen el comportamiento básico del sistema, en esencia son lo que el sistema hace o no debe hacer y pueden utilizarse en términos de cómo responder a las entradas de los usuarios. Definen comportamientos condicionados, incluyen cálculos, procesos de negocio y entradas de datos (*Imagen 6*). La definición de estos requerimientos permite que el sistema funcione según lo previsto ya que se centran en las necesidades del usuario.



Imagen 6: Requisito funcional

A diferencia de otras metodologías de desarrollo las metodologías ágiles como SCRUM establecen un enfoque más flexible sobre lo que es un requerimiento funcional, los proyectos son ejecutados en iteraciones cortas y se fomenta una constante interacción entre usuario y desarrolladores aceptando el cambio en lugar de rechazarlo.

Los **requerimientos no funcionales** no afectan a la funcionalidad básica del sistema, incluso si no se cumplen el sistema podrá seguir realizando su función básica. Definen el comportamiento, las funciones y las características generales del sistema desde la perspectiva de la experiencia de usuario. Estos requerimientos determinan la facilidad de uso del sistema y

se utilizan para evaluar el rendimiento (*Imagen 7*). Un ejemplo es un requerimiento no funcional que especifica la velocidad con la que carga la página web, sin este requerimiento podría verse comprometida la experiencia de usuario y la percepción de la calidad del sistema.

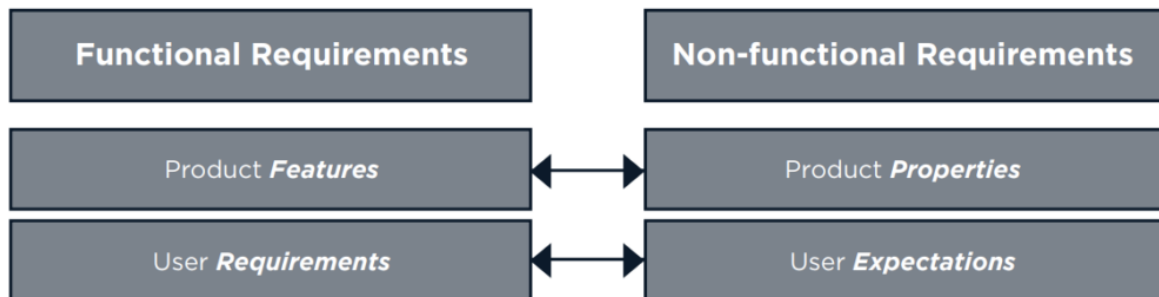


Imagen 7: Requerimientos

Requisitos funcionales del sistema

ID	Requisito	Descripción	Prioridad	Actor
RF1	Autenticación y roles	El sistema debe permitir login/logout y diferenciar permisos según rol (admin, cliente, entrenador).	Media	—
RF2	Crear usuario	El sistema debe permitir crear usuarios dependiendo de un rol (cliente / administrador / entrenador)	Alta	Administrador
RF3	Editar usuario	El sistema debe permitir actualizar datos de usuarios dependiendo de un rol (cliente / administrador / entrenador)	Alta	Administrador
RF4	Eliminar usuarios	El sistema debe permitir eliminar usuarios dependiendo de su rol	Alta	Administrador

1. Gestión de usuarios (*Tabla 1*)

Tabla 1: RF1 Gestión de usuarios

2. Gestión de membresías y suscripciones (*Tabla 2*)

ID	Requisito	Descripción	Prioridad	Actor
RF5	CRUD Planes	El sistema debe permitir crear, leer, actualizar y eliminar planes (ej: básico, premium).	Media	Administrador
RF6	Suscripciones asignar/eliminar	El sistema debe permitir asignar y eliminar suscripciones a clientes.	Media	Administrador

Tabla 2: RF2 Gestión de membresías y suscripciones

ID	Requisito	Descripción	Prioridad	Actor
RF7	Crear perfil de entrenamiento	El sistema permite crear un perfil con información del entrenador que se seleccionara para asignar rutinas de entrenamiento	Media	Cliente
RF7	Editar perfil de entrenamiento	El sistema permite editar un perfil con información para los entrenamientos (entrenador, métricas del cliente, objetivos y notas)	Media	Cliente
RF8	CRUD Rutinas	El sistema debe permitir a los entrenadores y clientes crear, leer, actualizar y eliminar rutinas.	Media	Entrenador/Cliente

3. Gestión de rutinas (Tabla 3)

Tabla 3: RF3 Gestión de rutinas

ID	Requisito	Descripción
RNF1	Seguridad	El sistema debe usar autenticación JWT y encriptación de contraseñas (bcrypt).
RNF2	Rendimiento	El sistema debe responder en menos de 3 segundos en operaciones CRUD
RNF3	Usabilidad	La interfaz debe ser intuitiva.
RNF4	Compatibilidad	El sistema debe funcionar en Chrome, Firefox y Safari (versiones recientes).

Requisitos no funcionales (Tabla 4)

Tabla 4: Requisitos no funcionales

2.3 Información a recopilar y almacenar

El sistema necesita gestionar los datos de usuarios, entrenadores, administradores, planes, rutinas, progresos, etc. Por lo que se entiende que se necesitan muchas relaciones dentro de los datos para al final ser consultados a través de ellas, por ello se utilizará como motor de bases de datos una base de datos relacional. Dichas bases de datos se caracterizan por almacenar y proporcionar acceso a puntos de datos relacionados entre sí. Se basan en el modelo relacional que es una forma directa e intuitiva de representar datos en tablas. (Oracle, 2021)

Los datos que van a ser requeridos para manejar las funcionalidades de la aplicación se dividen de la siguiente manera:

1. Datos de usuarios

Información básica:

Nombre, apellidos, teléfono, correo, contraseña

Perfil de Cliente:

Fecha de nacimiento, género.

Atributos físicos: peso actual, altura, objetivo, medidas (opcional)

Perfil de entrenador:

Biografía, especialidades o habilidades, redes sociales, horarios disponibles.

2. Planes

Nombre, descripción, beneficios, precio, duración en días.

Con esa información y la de los usuarios con el perfil de cliente se pretende crear suscripciones que actúen como puntos de partida para determinar si ese usuario tiene acceso a algún servicio del gimnasio. En el prototipo desarrollado en esta tesis sólo se abordará el servicio que da acceso a una aplicación web para el registro de rutinas otorgadas por un entrenador que el cliente elija como su instructor.

3. Suscripciones

Fecha inicio, fecha fin, cliente, plan

4. Rutinas de entrenamiento

Nombre, descripción, nivel, ejercicios (*Código 1*)

```
{  
  
  "nombre": "Sentadillas",  
  
  "series": 4,  
  
  "rep_meta": 12,  
  
  "nota": "Mantén la espalda recta"  
  
}
```

Código 1: JSON ejercicios

Estas rutinas están relacionadas con los entrenadores que son quienes las crean y a los clientes a quienes se les asignan. El dato de los ejercicios se especifica como JSON que es una notación en formato basado en texto estándar para representar datos

estructurados en la sintaxis de objetos de JavaScript lo cual nos da más control sobre los datos que se manejan para las rutinas al permitirnos manejar datos en clave - valor.

2.4 Integración con dispositivos y servicios externos

Envío de emails

Para medidas de seguridad e informes a los clientes se hizo uso de un servicio externo para el envío de emails, se utilizó el servicio de MailTrap, el cual es una plataforma de entrega y prueba de correo electrónico que actúa como servidor SMTP ficticio capturando los correos sin ser enviados a los destinatarios reales. También nos ofrece envíos de email reales dependiendo de la cantidad de emails que se enviaran al mes es cómo se elige el plan que se desea utilizar, hay desde planes gratuitos con 1,000 envíos de email al mes hasta planes para empresas grandes con 1,500,000 emails mensuales con un costo de 750 dólares al mes.

2.5 Aspectos de seguridad

Seguridad en el backend

Autenticación y autorización: para este apartado se implementaron diferentes estrategias para abordar conceptos claves de seguridad a los que se enfrentan los servidores con API Rest y al mismo tiempo se aborda seguridad dentro de la base de datos.

- Confirmar cuenta mediante tokens únicos:

Se generan tokens dentro del servidor al crear una cuenta, esta estrategia evita que la base de datos se sature de usuarios por cuentas creadas con correos inexistentes o cuentas que utilizan un correo que no les pertenece. Dichos tokens serán enviados mediante un correo para después ingresarlos en una petición a la API Rest que confirma la cuenta para poder hacer uso de ella, en caso de no confirmar la cuenta no se puede ingresar a su cuenta y evitará realizar ciertas operaciones sobre ella.

- Uso de JWT (JSON Web Token) y middlewares:

En la vida cotidiana para realizar trámites; por ejemplo, abrir una cuenta bancaria, es necesario una contraseña y datos personales. De esta manera esos datos se convierten en una firma digital que hará que la persona se identifique de manera única para ese banco y cada vez que se acceda al banco e ingrese la contraseña y datos personales, el banco comprenderá que tú eres el usuario que

inicio sesión permitiendo realizar operaciones que son privadas como movimientos con tu dinero.

Un JWT es un estándar para la autenticación e intercambio de información mediante un token que almacena objetos JSON de forma segura y compacta en código Base64. (*¿Qué Es JSON Web Token?* | Alura Cursos Online, n.d.)

La estructura de un JWT está compuesta por encabezado, carga útil y firma las cuales se separan por puntos **“header.payload.signature”**

- **Header:** encabezados del token donde se asignan el algoritmo para crear la firma “alg” y el tipo de token “typ”

```
{  
  "alg": "HS256",  
  "typ": "JWT"  
}
```

- **Payload:** carga útil es el componente donde se encuentran datos de autenticación como correos electrónicos, nombre, rol; pero es importante entender que en este componente no deben ir datos sensibles como contraseñas o cuentas bancarias. (*¿Qué Es JSON Web Token?* | Alura Cursos Online, n.d.)

```
{  
  "email" : "nombre@correo.com"  
  "rol" : "member"  
}
```

- **Signature:** la firma del token está compuesta por la codificación del encabezado y el payload más una clave secreta y es generada por un algoritmo especificado en el encabezado. (*¿Qué Es JSON Web Token?* | Alura Cursos Online, n.d.)

```
HS256SHA256(base64UrlEncode(header) + "." +  
base64UrlEncode(payload), secret_key)
```

El resultado de esas tres cadenas nos da algo así
“eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.

eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0Ijox
NTE2MjM5MDIyfQ.

SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c” dicha cadena nos permite manejar las rutas a las que se puede acceder dentro del cliente y las rutas

de la API a la que se puede acceder por lo que nos da paso a proteger rutas sensibles con el uso de middlewares.

Los middlewares son funciones que se insertan entre la solicitud y la respuesta de una aplicación, estos se usan para dar autorización a rutas críticas o privadas de cada tipo de rol, por ejemplo; los administradores son los únicos que pueden registrar a los entrenadores, por lo que es necesario implementar un middleware que verifique si el usuario cumple con el rol específico, para ello en el encabezado de la solicitud se inserta el token del usuario autenticado para en el middleware extraerlo, después ejecutar la consulta al tipo de usuario que está haciendo la solicitud y así decidir si tiene acceso o no a esa ruta.

- Validación de datos

Es necesario validar los datos que se reciben en las rutas de la aplicación, esto nos previene de código malicioso que algún atacante desee ingresar, para ello en JavaScript existen diferentes librerías, pero en este caso se hizo uso de express-validator (librería para validar datos de entrada) que verifica que los datos recibidos en el cuerpo de las solicitudes o en los parámetros de rutas cumplan con un formato específico, en caso de no hacerlo se retornará un error y no permite seguir con la operación de la ruta. Esta medida de seguridad evita inyecciones a la base de datos y XSS (cross-site scripting)

- Seguridad en la base de datos, ORM y encriptación de datos:

Para evitar utilizar queries crudos de SQL y que sea una vulnerabilidad que los atacantes pueden explotar para ingresar a datos de la aplicación se utilizó un ORM (Object Relational Mapping), esto una técnica de programación que permite interactuar con la base de datos utilizando objetos del lenguaje de programación en lugar de escribir las queries manualmente. Esta capa de abstracción sanitiza la entrada de los datos automáticamente lo que añade seguridad extra para los datos que ingresamos a la base de datos.

Se encriptan las contraseñas para que sean seguras, nunca se guardan en texto plano.

- Protección del servidor:

Se controla el límite de cantidad de solicitudes que se pueden realizar a las APIs haciendo uso de una técnica llamada rate limiting la cual previene que se ponga en riesgo la disponibilidad de los recursos y la seguridad de la API, detectando a bots maliciosos que aumentan la presión en el servidor o usuarios que estén

haciendo mal uso de las peticiones desde una misma IP y los bloquea para que no siga haciendo solicitudes cuando llegue al rango límite. Permitiendo así que el servidor esté disponible en todo momento. (Expósito, 2023)

Seguridad en el cliente

En el lado del cliente que es el frontend creado con React se gestionan rutas privadas las cuales validan el tipo de usuario con su rol para renderizar el contenido que le corresponde según su rol dentro del sistema, permitiendo así la separación de tareas para que no sean públicas a todos los usuarios, esto va de la mano con la autorización a las rutas de la API del servidor.

Las llamadas a la API contienen los tokens en la cabecera de la petición con Axios en formato Bearer para mantener seguros esos tokens en la comunicación con el servidor.

2.6 Requisitos técnicos:

2.6.1 Javascript

Es un lenguaje de programación utilizado en gran mayoría para hacer páginas web interactivas, desde actualizar las fuentes de texto hasta mostrar animaciones o mapas interactivos. Funciona mediante la traducción de sintaxis similar a la del inglés a código máquina, que después el navegador o motor de ejecución se encarga de ejecutarlo. JavaScript se clasifica como un lenguaje de scripting o interpretado, es decir, directamente traducido a código de lenguaje máquina en tiempo de ejecución. (*¿Qué Es JavaScript? - Explicación De JavaScript (JS) - AWS*, n.d.)

Javascript está orientado a objetos y es multiplataforma, lo que lo convierte en uno de los lenguajes más utilizados al tener una gran aceptación por los desarrolladores al ser utilizado en móvil, web, API Rest, aplicaciones de escritorio, IA, blockchain, etc.

- Javascript del lado del cliente:

En el navegador funciona mediante el motor de JavaScript que ya se incluye en cada navegador, la manera en que se utiliza en el navegador es asociando funciones creadas con JS a eventos del navegador como hacer clic o situar el ratón sobre algún elemento.

El flujo de cómo se asocian los eventos a las funciones es de la siguiente manera:

1. El navegador carga la página web cuando el usuario la visita
2. Durante la carga el navegador convierte la página con todos sus elementos como botones, etiquetas y cuadros desplegables en una estructura de datos denominada *Document Object Model (DOM)*.

3. El motor de JavaScript del navegador convierte el código JavaScript en código intermedio.
 4. Los diferentes eventos como hacer clic desencadenan la ejecución de bloques de código asociado para posteriormente el motor interpretar el código y realizar los cambios en el DOM.
 5. El navegador muestra el nuevo DOM
- JavaScript del lado del servidor:

En el lado del backend también se puede usar JavaScript, en este caso el motor de JavaScript se encuentra directamente en el servidor. Las funciones que se pueden implementar del lado del servidor son acceder a la base de datos, realizar operaciones lógicas y responder a varios eventos desencadenados por el sistema del servidor.

Node.js ha alcanzado gran popularidad por la comunidad de desarrolladores para escribir código en el backend en los últimos años. Una prueba de su popularidad es que las principales plataformas de streaming y video han sido codificadas en Node.js.
 - Beneficios de JavaScript:
 - La sintaxis se basa en el lenguaje de programación Java y es fácil de aprender. Es utilizado en muchos sitios web y aplicaciones móviles para scripting del lado del cliente.
 - Reduce la carga del servidor y congestión de la red ya que permite ejecutar operaciones lógicas y realizar gran parte del trabajo del servidor en el propio cliente como la validación de formularios, siendo así que se pueden ejecutar funciones en el cliente que revisen que los tipos de datos que se enviaran cumplan patrones como cierta cantidad de caracteres, tipos de datos específicos e incluso realizar operaciones lógicas como suma de precios para enviar el total de una venta directamente a las solicitudes del backend y reduciendo las operaciones en él.
 - La creación de sitios web o aplicaciones sofisticadas es posible gracias a la variedad de bibliotecas y frameworks existentes. Debido a que JavaScript es muy utilizado hoy en día, existe una gran comunidad que constantemente está creando herramientas para crear sitios web dinámicos, que cumpla con estándares necesarios y que sean optimizados en todo momento, algunas de esas herramientas son React, Svelte, Vue, Angular y React Native.

2.6.2 TypeScript

Typescript es un lenguaje de programación open-source mantenido y desarrollado por Microsoft. Su función es añadir sintaxis adicional a JavaScript para otorgar una integración más amplia con el editor de código. Nos permite identificar errores a tiempo en el momento de desarrollar o en el pipeline de CI/CD y escribir código más mantenible.

Con TypeScript se introduce el tipado estático opcional que permite a los desarrolladores especificar el tipo de variables, parámetros y valores de retorno, detectando errores relacionados con el tipo durante el desarrollo. (*Node.js — Introduction to TypeScript*, n.d.)

```
1 type User = {  
2   name: string;  
3   age: number;  
4 };  
5  
6 function isAdult(user: User): boolean {  
7   return user.age >= 18;  
8 }  
9  
10 const justine = {  
11   name: 'Justine',  
12   age: 23,  
13 } satisfies User;  
14  
15 const isJustineAnAdult = isAdult(justine);
```

Imagen 8: TypeScript Sintaxis

Para entender mejor cómo funciona TypeScript es necesario ver la imagen 8 donde la palabra reservada “**type**” es responsable de declarar un tipo User que funciona como contrato que se debe cumplir para un objeto, en este caso ese contrato se utiliza en la función **isAdult**, dicha función sólo admite como parámetro un objeto de tipo User, esto quiere decir que los datos especificados dentro del type User serán los únicos recibidos por la función como parámetro para realizar operaciones con esos datos dentro de ella.

La manera en que trabaja TypeScript es proporcionando esta capa de seguridad de especificar el tipo de dato que viajará a través de las operaciones del sistema o script donde se esté utilizando, para que el editor de código nos otorgue autocompletado más preciso y nos haga saber en qué parte del código estamos fallando con los contratos de tipos de las variables y funciones que están siendo creadas en el momento de desarrollo.

Algo importante de mencionar es que TypeScript no es el lenguaje en el que se ejecutarán los scripts en producción, solo funciona en desarrollo ya que posteriormente para ser soportado por los motores de JavaScript debe compilarse a lenguaje JavaScript para ser compatible en cualquier lugar donde se ejecute dicho lenguaje.

2.6.3 Node.js y Express

Node.js es un entorno que trabaja en tiempo de ejecución, es multiplataforma y permite a los desarrolladores crear toda clase de herramientas del lado del servidor y aplicaciones en JavaScript. La ejecución en tiempo real es para usarse fuera del contexto de un navegador web lo que permite que se ejecute directamente en una computadora o el sistema operativo del servidor. El entorno omite las APIs que permiten trabajar con el navegador web y añade soporte a APIs del sistema operativo más tradicionales como HTTP y biblioteca de sistema de ficheros. Una aplicación de Node.js se ejecuta en un solo proceso, sin crear nuevos hilos para cada solicitud. Node.js proporciona un conjunto de operaciones E/S (entrada/salida) asíncronas en su biblioteca estándar que evitan el bloqueo de código JavaScript.

Cuando se realiza una operación de E/S como leer la red, acceder a una base de datos o sistema de archivos, en lugar de bloquear el hilo y perder ciclos de CPU en espera, Node.js reanuda las operaciones al recibir la respuesta. Esto permite gestionar miles de conexiones simultáneas con un solo servidor sin la carga de gestionar la concurrencia de hilos, lo que podría ser una fuente importante de errores. (*Node.js — Introduction to Node.js*, n.d.)

Ventajas de Node.js

- Gran rendimiento: fue diseñado para optimizar el rendimiento y la escalabilidad en aplicaciones web, es muy utilizado problemas comunes de desarrollo web como aplicaciones en tiempo real.
- Los desarrolladores de frontend pueden crear servidores utilizando el mismo lenguaje del lado del cliente que es JavaScript por lo que puede crear aplicaciones completas sin tener que lidiar con aprender un nuevo lenguaje. (*Node.js — Introduction to Node.js*, n.d.)
- Es portable, con versiones que funcionan en Microsoft Windows, OS X, Linux, Solaris, FreeBSD, OpenBSD, WebOS, y NonStop OS. Además, está bien soportado por muchos de los proveedores de hospedaje web, que proporcionan infraestructura específica y documentación para hospedaje de sitios Node.
- Tiene un ecosistema y comunidad de desarrolladores muy activa, por lo que es fácil encontrar recursos en la web que nos ayuden a resolver problemas en proyectos reales.

Express

Es el framework web más popular de Node ya que proporciona mecanismos para:

- Creación de manejadores de peticiones con diferentes verbos HTTP en diferentes rutas (GET, POST, SET, etc.) para la creación de API Rest.

- Integración con motores de renderización de vistas para generar respuestas mediante la introducción de datos en plantillas.
- Añadir procesamiento de peticiones middleware adicional en cualquier punto dentro del manejo de la solicitud (*Node.js — Introduction to Node.js*, n.d.)

2.6.4 PostgreSQL

Una base de datos relacional es una colección de elementos de datos con relaciones predefinidas. Estos elementos se organizan como un conjunto de tablas con columnas y filas. PostgreSQL es un sistema de base de datos relacionales de código abierto altamente estable, que proporciona soporte a funciones de SQL como, claves foráneas, subconsultas, disparadores y tipos o funciones definidas por el usuario. Aumenta el beneficio del lenguaje SQL ofreciendo funciones que escalan y reservan meticulosamente las cargas de trabajo de datos, y tiene soporte a datos JSON. (*PostgreSQL 18.0 Documentation*, 2025)

Como servidor de bases de datos, su función principal es almacenar datos de forma segura y conforme a las mejores prácticas, y recuperarlos posteriormente, según lo soliciten otras aplicaciones de software, ya sean las que se encuentren en el mismo equipo o las que se ejecuten en otro equipo a través de una red (incluida Internet).

Se utiliza principalmente para almacenar datos de aplicaciones móviles, web, geoespaciales y de análisis. (*PostgreSQL 18.0 Documentation*, 2025)

Posee un diseño orientado a objetos con lo cual los programadores pueden comunicarse con los servidores de bases de datos mediante objetos en su código, definir herencia o relaciones padre-hijo entre tablas.

2.6.5 Sequelize ORM

Un ORM (Object-Relational Mapping) es una técnica de programación que permite a los desarrolladores interactuar con una base de datos relacional utilizando objetos de programación en lugar de consultas SQL directas. El objetivo principal de un ORM es proporcionar una capa de abstracción que facilite la comunicación y la manipulación de datos entre el código de la aplicación y la base de datos. (Chill C., 2025)

En un sistema de base de datos, estos se almacenan en tablas y cuando se necesita acceder a esos datos desde una aplicación se hacen consultas con SQL para interactuar con la base de datos. Sin embargo, trabajar con SQL puede ser complejo para algunos desarrolladores.

El ORM de sequelize resuelve este problema mapeando las tablas de la base de datos en clases y los registros de las tablas a objetos dependiendo del lenguaje que se utilice, puede ser JavaScript, Python, Java, C#, etc. Se puede ver si flujo en la imagen 9 (Chill C., 2025)

Algunos de sus beneficios son:

- Ofrece operaciones de alto nivel para CRUDs, las cuales se ejecutan a través de métodos proporcionados por el ORM lo que facilita el acceso y manipulación de los datos.
- Gestión de transacciones para garantizar la integridad de los datos y la consistencia en caso de que ocurran errores durante las operaciones.
- Puede ayudar a prevenir ataques de inyección SQL, ya que los datos se manejan de manera segura a través de objetos y métodos. El ORM se encarga de limpiar los datos automáticamente lo que protege las aplicaciones contra vulnerabilidades.

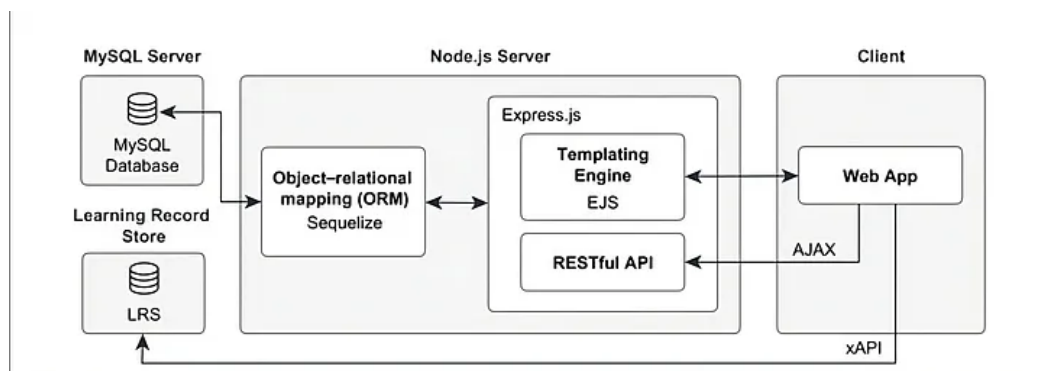


Imagen 9: Funcionamiento Sequelize [21]

2.6.6 React

React es una biblioteca para construir interfaces de usuario. React no es un framework y tampoco se limita a la web. React es utilizado con otras bibliotecas para renderizar en ciertos entornos. Por ejemplo, React Native puede usarse para desarrollar aplicaciones móviles.

El objetivo principal de React es minimizar los errores que ocurren cuando los desarrolladores construyen interfaces de usuario. Esto lo hace mediante el uso de componentes o piezas de código lógicas y auto-contenidas que describen una parte de la interfaz del usuario. Estos componentes se pueden juntar como si fueran LEGOs para crear una interfaz de usuario completa, y React abstrae la mayor parte del trabajo de renderizado, permitiéndote enfocarse en el diseño de la interfaz.

A diferencia de otros frameworks, al ser una biblioteca, React no impone reglas estrictas sobre convenciones de código u organización de archivos lo cual permite establecer a los equipos las convenciones que funcionen mejor para ellos y usar React de la manera que se desee.

React utiliza características modernas de JavaScript para muchos de sus patrones. El punto donde más se aleja de JavaScript se refleja en el uso de la sintaxis JSX, la que, a su vez, amplía la sintaxis de JavaScript para que código similar a HTML pueda formar parte del mismo, un ejemplo de la sintaxis es la imagen 10.

```
export default function MyApp() {  
  return (  
    <div>  
      <h1>Bienvenido a mi aplicación</h1>  
      <MyButton />  
    </div>  
  );  
}
```

Imagen 10: Sintaxis .jsx

Para que React sea dinámico contiene “hooks” que permiten usar diferentes características de React en los componentes, también se pueden crear hooks personalizados con lógica que nosotros incorporamos para integrarlos en la interfaz, unos de los ya existentes son:

- Hooks de estado: Permite al componente recordar información como el valor de un input.
 - **useState**: declara un estado de variable que puedes actualizar directamente.
const [state, setState] = useState(initialState)
 - **useReducer**: declara un estado variable con la lógica de actualización dentro de una función llamada reducer. Los reducers son una manera diferente a la de useState de manejar estados, en vez de decirle a React “qué hacer” al asignar un estado, especifica que acaba de hacer el usuario despachando acciones desde los controladores de eventos. (*Hooks Integrados En React – React*, n.d.)
const [state, dispatch] = useReducer(reducer, initialArg, init)
- Hooks de contexto: Permite a un componente recibir información desde un componente padre distante sin pasarlas como props. Un uso cotidiano es al pasar el tema de la interfaz de usuario actual a todos los componentes hijo sin importar que sean hijos muy lejanos.
 - **useContext**: permite leer y suscribirse a un contexto desde un componente. No almacena esa información, sólo representa el tipo de información que se puede

proporcionar o leer desde los componentes. (*Hooks Integrados En React – React*, n.d.)

```
const value = useContext(some)
```

- Hooks Ref: permite a un componente conservar algo de información que no es usada para el renderizado como un nodo del DOM o el ID de un timeout. La diferencia con un estado es que actualizar un ref no vuelve a renderizar el componente.

- **useRef**: declara un ref para conservar cualquier valor en él, se utiliza comúnmente para conservar nodos del DOM.

```
const inputRef = useRef(null);
```

- **useImperativeHandler**: permite personalizar el ref expuesto por el componente.
- Hooks de efecto: permiten a un componente conectarse y sincronizarse con sistemas externos lo que incluye; lidiar con la red, el DOM del navegador, animaciones, widgets y otro código que no sea React.

- **useEffect**: un hook que recibe como parámetro una función que se ejecutará cada vez que nuestro componente se renderice, ya sea por un cambio de estado, por recibir props nuevas o, y esto es importante, porque es la primera vez que se monta. (*Hooks Integrados En React – React*, n.d.)

```
useEffect(() => {
```

```
    // Funciones a ejecutar
```

```
  }, [dependencias])
```

- Hooks de rendimiento: Una forma común de optimizar el rendimiento del re-renderizado es saltarse el trabajo innecesario. Por ejemplo, puedes decirle a React que reutilice cálculos que están en la caché o que se salte un re-renderizado si los datos no han cambiado desde el renderizado anterior.
- **useMemo**: permite almacenar en caché el resultado de cálculos costosos.
 - **useCallback**: permite almacenar en caché la definición de una función antes de pasarla a un componente optimizado

2.6.5 TailwindCSS

Es un framework de CSS que, a diferencia de los tradicionales como Bootstrap, no crea componentes con una sola clase, sino que tiene algo llamado Utility Classes, que son clases específicas para cada cosa. Por ejemplo, una clase para los textos, otra clase para las sobras, una para el color, entre otros. El objetivo es que se pueda personalizar a fondo.

Tailwind también resuelve por detrás todos los fantasmas de CSS como herencia, especificidad, conflictos, cascadas, etc.

Este enfoque tiene varios beneficios:

- **Desarrollo más rápido:** Las clases predefinidas ayudan a los desarrolladores a construir e iterar rápidamente en diseños sin tener que cambiar constantemente entre archivos HTML y CSS.
- **Estilo consistente:** Tailwind proporciona un conjunto estandarizado de clases, lo que ayuda a mantener la consistencia en el diseño en todo el proyecto.
- **Archivos CSS más pequeños:** Debido a que los estilos se aplican directamente en el HTML, no es necesario tener archivos CSS grandes y personalizados.
- **Mantenimiento más fácil:** Con los estilos definidos en el HTML, es más fácil ver cómo los cambios afectarán a un elemento específico sin tener que buscar en archivos CSS separados.

```
<img class="size-48 otros-estilos" alt="" src="" />
```

Capítulo 3: Análisis y diseño

En este capítulo se hace el análisis de los requerimientos que se describieron en el capítulo anterior para realizar el diseño del sistema, desde la UX hasta las funcionalidades plasmadas en casos de uso y diagramas de secuencia para su mejor desarrollo.

3.1 Revisión de requerimientos:

Login

Caso de uso

Caso de uso	Login	
Actores	Admin/Miembro/Entrenador	
Descripción	Inicio de sesión al sistema, depende del rol le dará acceso a contenido específico (dashboard o aplicación).	
Precondición	Tener una cuenta creada y confirmada vía correo.	
Flujo principal	Paso	Acción
	1	Ingresa a la pantalla de login.
	2	Ingresa credenciales (email, password) y da click a iniciar sesión.
	3	El sistema valida las credenciales.
	4	Si son correctas las credenciales ingresa al sistema.
Excepciones	Paso	Acción
	1	Las credenciales no son correctas, regresa al paso 2 del flujo principal.
	2	Si no recuerda contraseña va al caso de uso de recuperar contraseña.
	3	Si recibe mensaje de error ir al caso de uso confirmar cuenta.
	4	La cuenta no existe. Solicitar al administrador crear una.

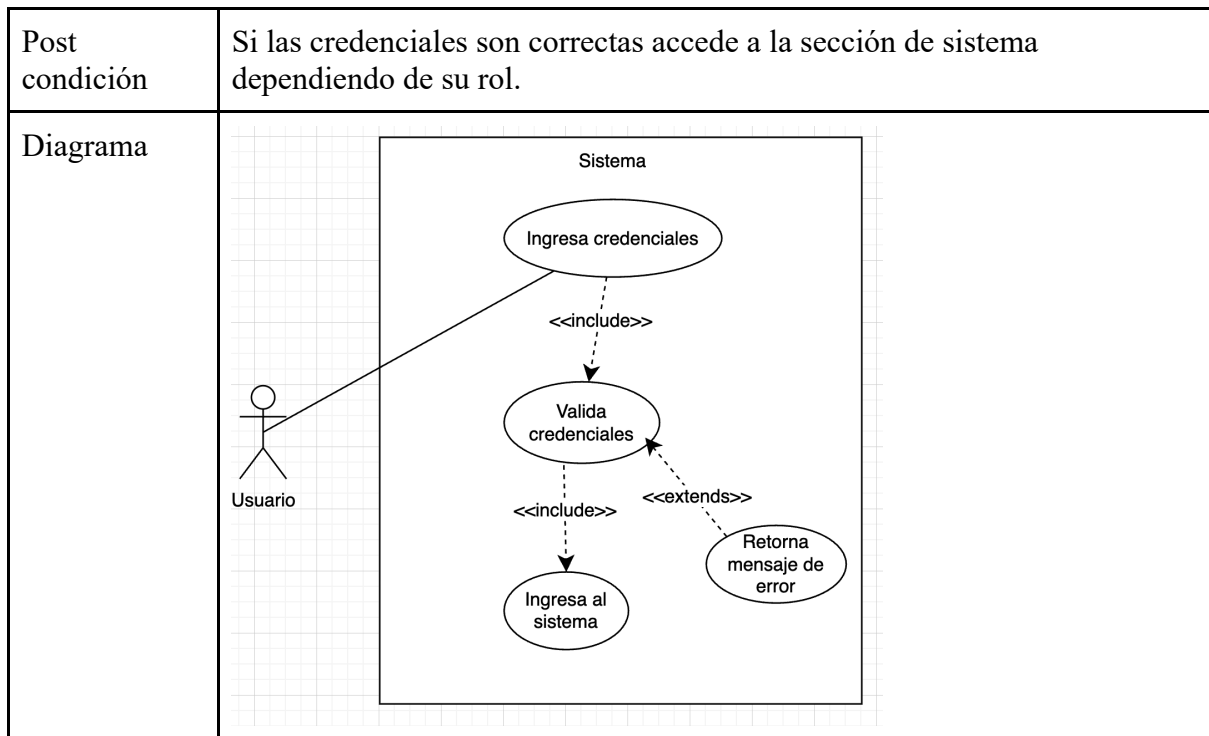
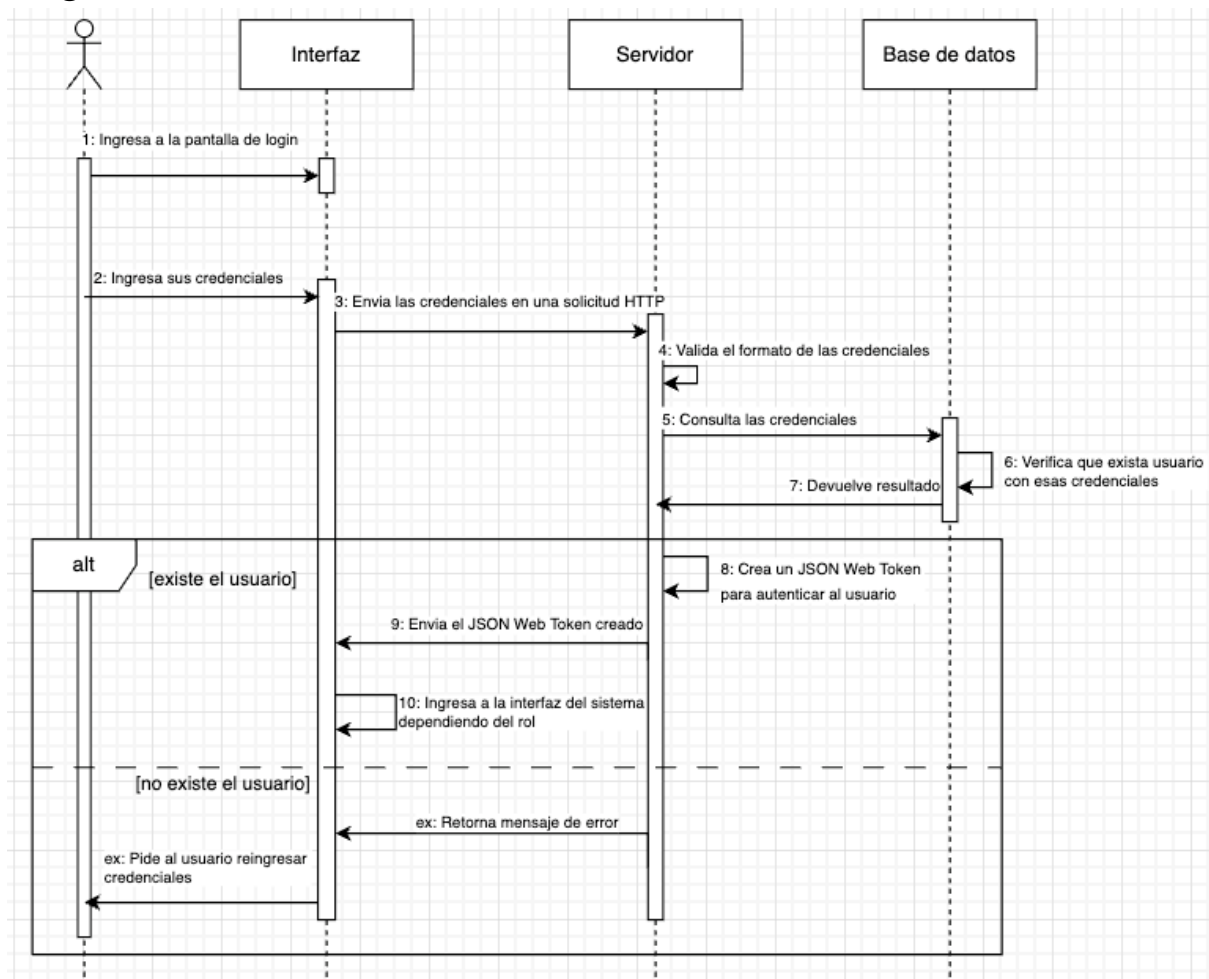


Diagrama de secuencia



Confirmar cuenta

Caso de uso

Caso de uso	Confirmar cuenta	
Actores	Admin / Entrenador / Cliente	
Descripción	El usuario confirmará su cuenta ingresando al enlace que se le comparte por correo.	
Precondición	Tener cuenta creada esperando la confirmación.	
Flujo principal	Paso	Acción
	1	Ingresa a la URL que llega por correo electrónico para confirmar la cuenta.
	2	El sistema envía la solicitud de confirmación con un token que va dentro de la URL.
	3	El sistema valida el token.
	4	Retorna mensaje de éxito.
Excepciones	Paso	Acción
	1	Si el token ya se usó o no existe, el sistema retorna un mensaje indicando “token expirado”.
Post condición	—	

Diagrama

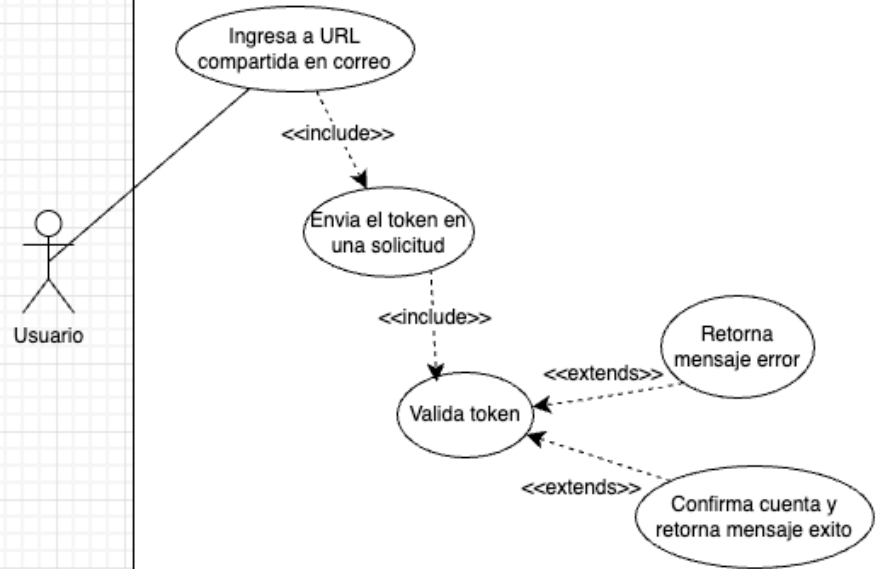


Diagrama de secuencia

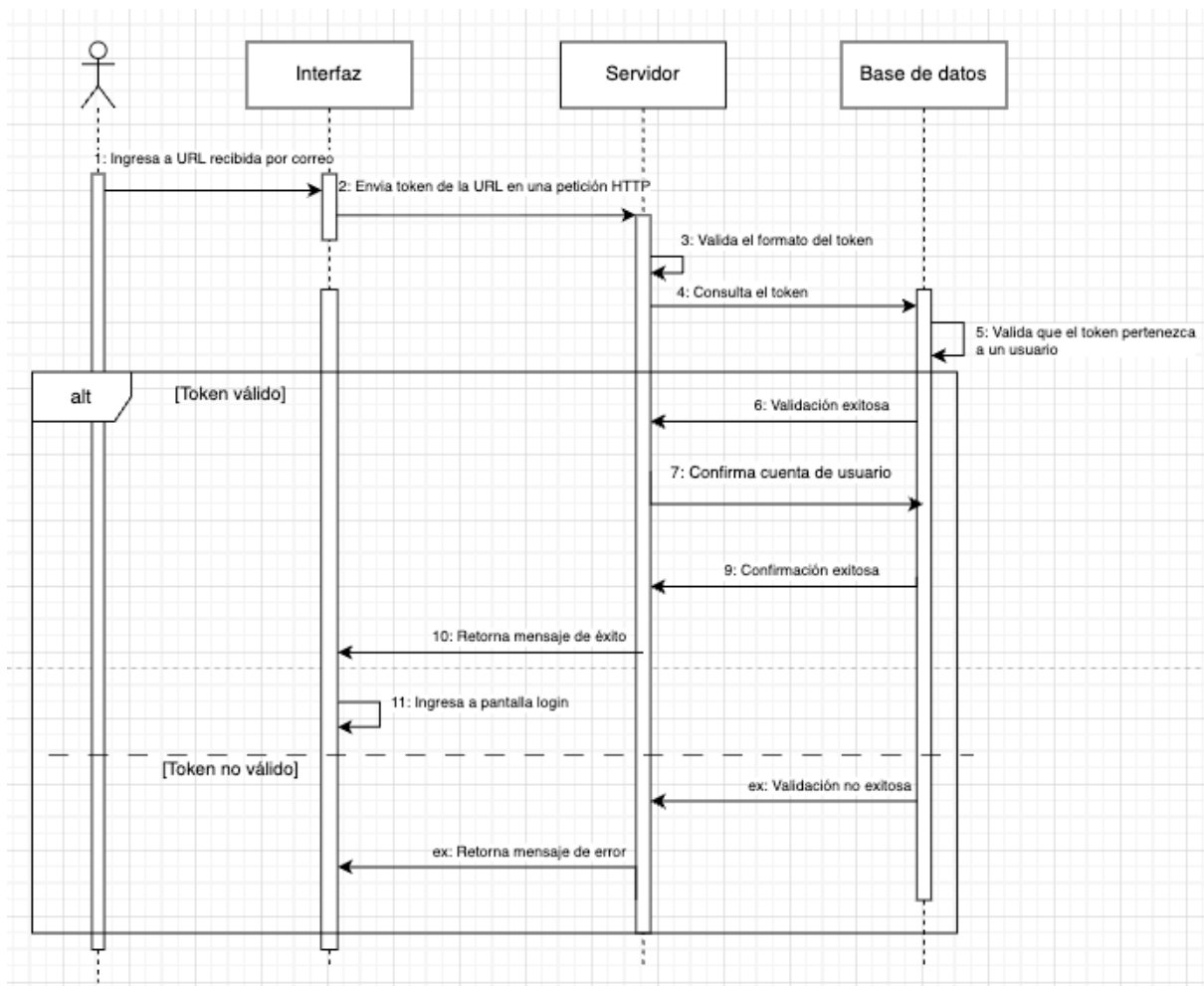
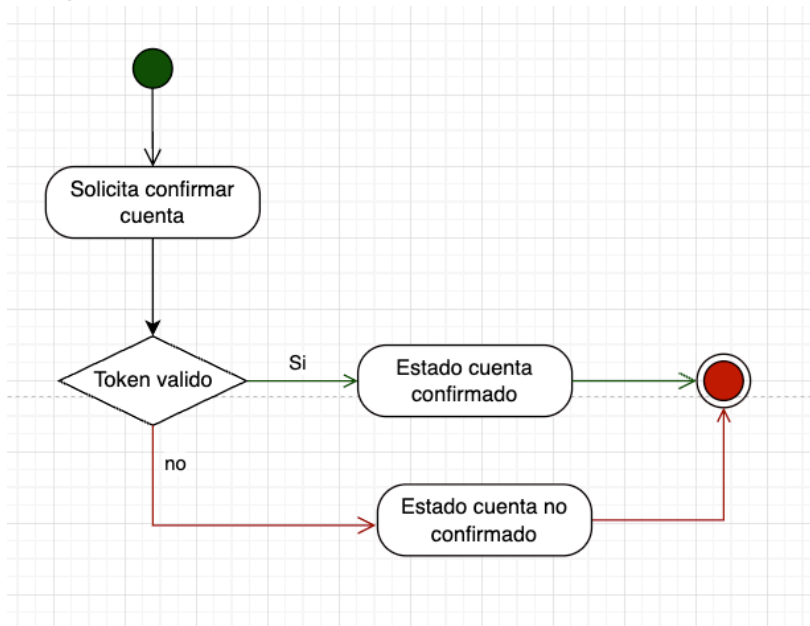


Diagrama de estado



Restablecer contraseña

Caso de uso

Caso de uso	Restablecer contraseña	
Actores	Admin / Entrenador / Cliente	
Descripción	El usuario puede restablecer su contraseña si la ha olvidado ingresando una solicitud con su correo.	
Precondición	Tener cuenta creada.	
Flujo principal	Paso	Acciones
	1	El usuario ingresa a la pantalla de olvidé contraseña.
	2	Ingresa su correo al formulario y da clic a solicitar restablecer.
	3	El sistema envía un correo con la URL que contiene un token para dirigir al usuario a la pantalla para ingresar la nueva contraseña.
	4	El usuario ingresa a la URL recibida por correo.
	4	El usuario ingresa una nueva contraseña y envía un formulario.

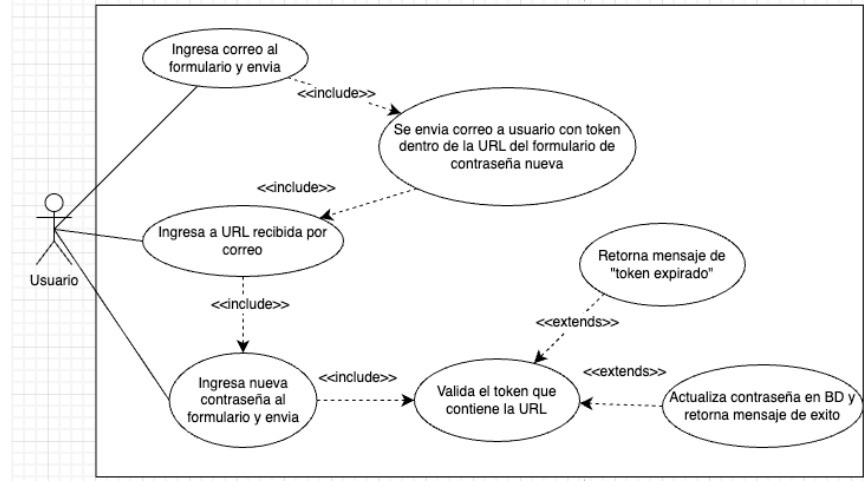
	5	El sistema valida el token de restablecer contraseña.
	6	Se guarda la contraseña nueva y retorna el mensaje de éxito.
Excepciones	Paso	Acciones
	1	Si el correo ingresado al formulario no existe, el sistema lanza un mensaje indicando “la cuenta no existe” y regresa el usuario al paso 2.
	2	Si en la validación del token ya fue usado lanza mensaje indicando “token expirado” y el usuario regresa al paso 2.
Post condición	—	
Diagrama	 <pre> sequenceDiagram actor Usuario Usuario->>U1: Ingresar correo al formulario y enviar U1-->>U2: <<include>> Se envía correo a usuario con token dentro de la URL del formulario de contraseña nueva U2->>U3: Ingresar a URL recibida por correo U3-->>U4: <<include>> Ingresar nueva contraseña al formulario y enviar U4-->>U5: <<include>> Validar el token que contiene la URL U5-->>U6: <<extends>> Retornar mensaje de "token expirado" U5-->>U7: <<extends>> Actualizar contraseña en BD y retornar mensaje de éxito </pre> The diagram illustrates the password reset process. It starts with a user (Usuario) who enters an email and sends it. This triggers an include relationship to a step where a token is sent to the user via email. The user then clicks the received URL, which leads to another include relationship for entering a new password. This is followed by an include relationship for validating the token. From the validation step, two extend relationships branch out: one for returning a 'token expired' message and another for updating the password in the database and returning a success message.	

Diagrama de secuencia

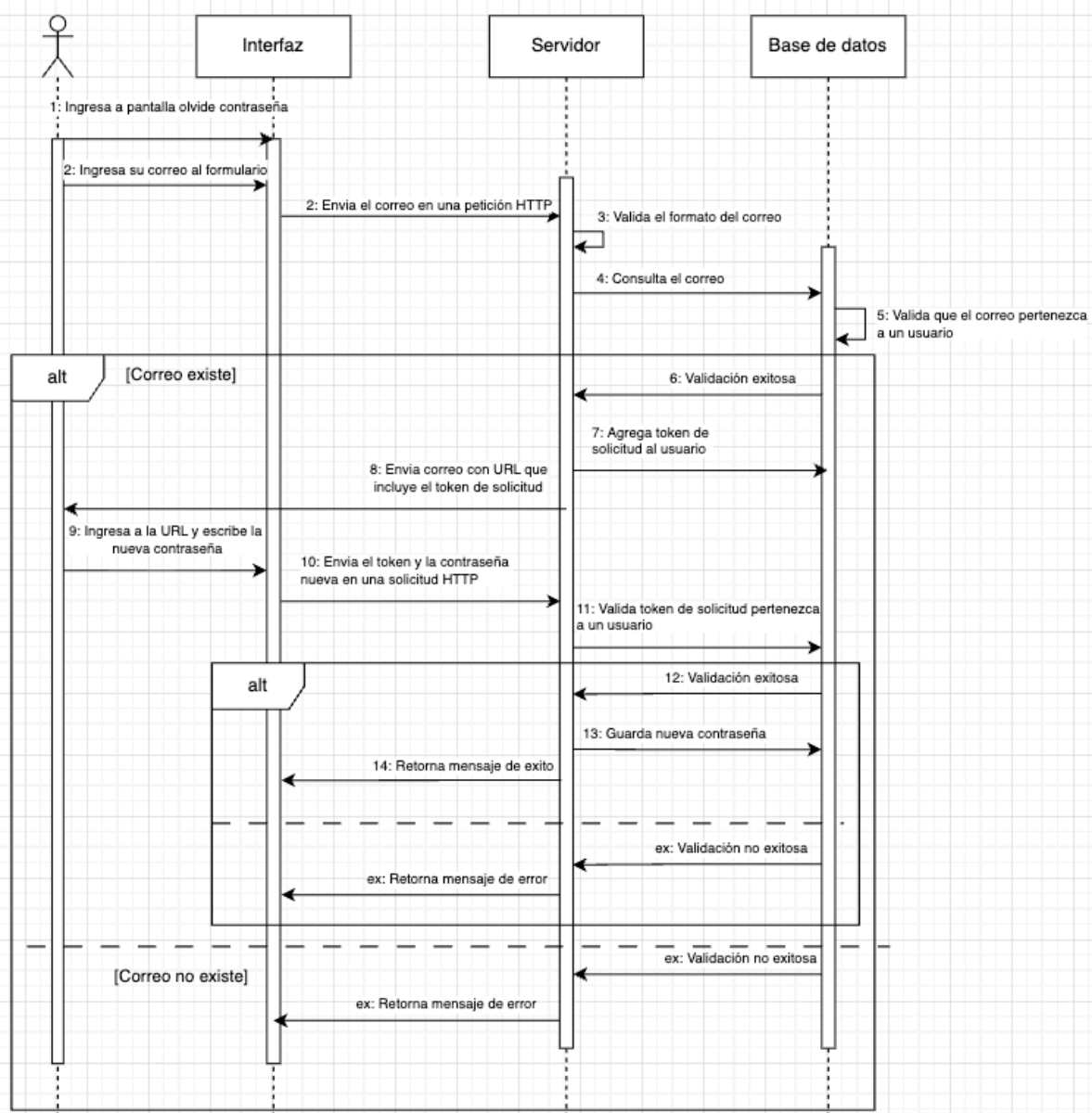
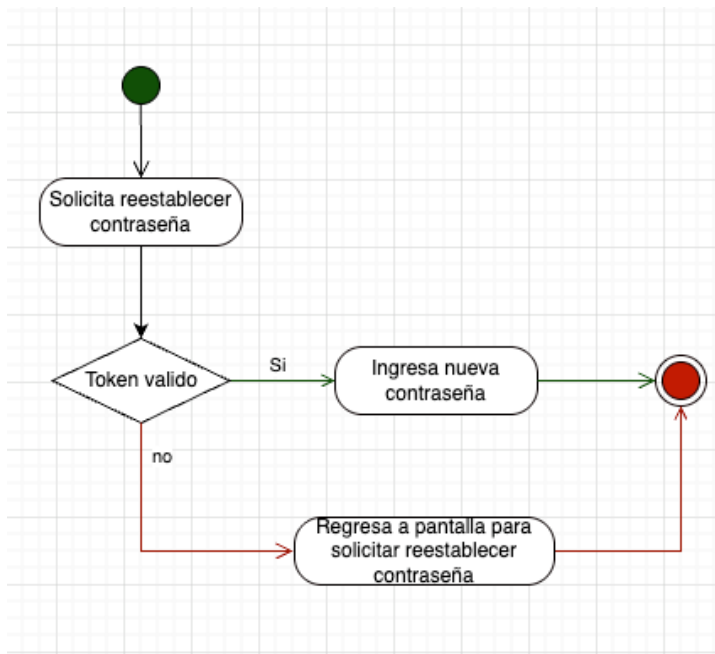


Diagrama de estado



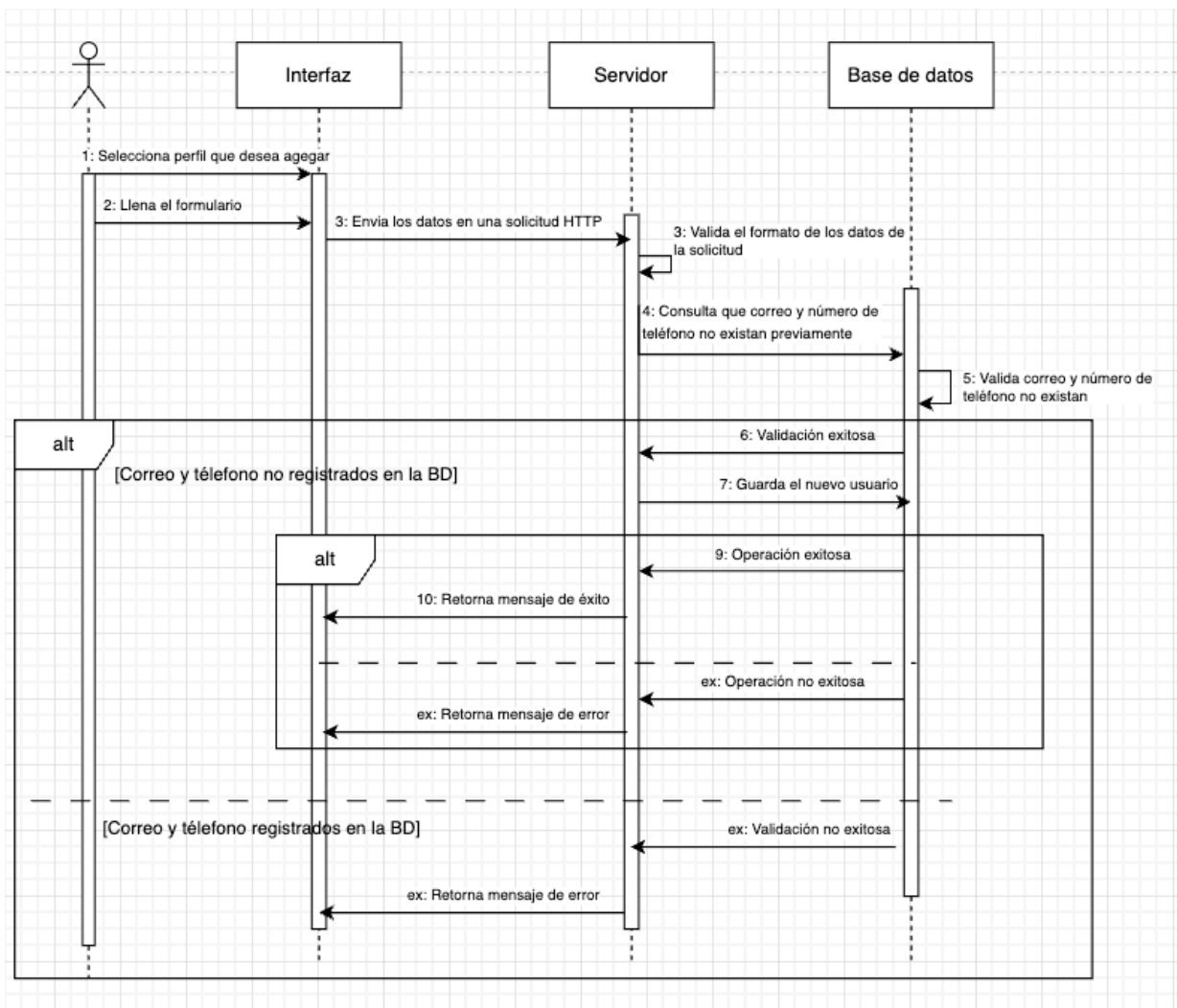
Crear usuario

Caso de uso

Caso de uso	Crear usuario	
Actores	Admin	
Descripción	El administrador ingresa a dashboard en la pantalla de agregar usuario y selecciona el perfil que desea agregar dependiendo del rol (cliente, entrenador, administrador).	
Precondición	Haber iniciado sesión al sistema con una cuenta de administrador.	
Flujo principal	Paso	Acción
	1	Selecciona el perfil que se desee crear.
	2	Ingresa los datos del formulario y da clic a enviar formulario.
	3	El sistema valida los datos.
	4	El usuario se guarda en la base de datos.
	5	El sistema envía correo al usuario para confirmar la cuenta.
	6	El sistema retorna el mensaje de éxito o error.

Excepciones	Paso	Acción
	1	Los datos del formulario no son correctos. Repite paso 2.
	2	Si el correo y teléfono ya están en el sistema retorna mensaje de "cuenta existente" y repite paso 2.
Post condición	El usuario debe confirmar su cuenta mediante la URL que se le proporciona en el correo para que posteriormente le permita el ingreso a la plataforma.	
Diagrama	<pre> sequenceDiagram actor Admin as Administrador Admin->>U1: Ingresar a pantalla agregar usuario U1->>U2: <<include>> U2->>U3: <<include>> U3->>U4: <<include>> U4->>U5: Validación de datos U5-->>U6: <<extends>> U6-->>U5: <<extends>> U5-->>U7: <<extends>> U7-->>U5: <<extends>> U5->>U8: <<include>> U8->>U9: Envía correo para confirmación </pre> The diagram illustrates the user registration process. It begins with an actor labeled 'Administrador' initiating the process by entering the 'Ingresar a pantalla agregar usuario' screen. This is followed by a sequence of steps: 'Selecciona perfil' (selected via include), 'Llena el formulario y envía' (selected via include), and 'Validación de datos' (selected via include). From 'Validación de datos', there are two parallel paths: one for error handling ('Devuelve mensaje de error') and another for successful registration ('Guarda el usuario en la BD'), both connected to the main flow via 'extends' relationships. The process concludes with 'Envía correo para confirmación', which is also included in the main flow.	

Diagrama de secuencia



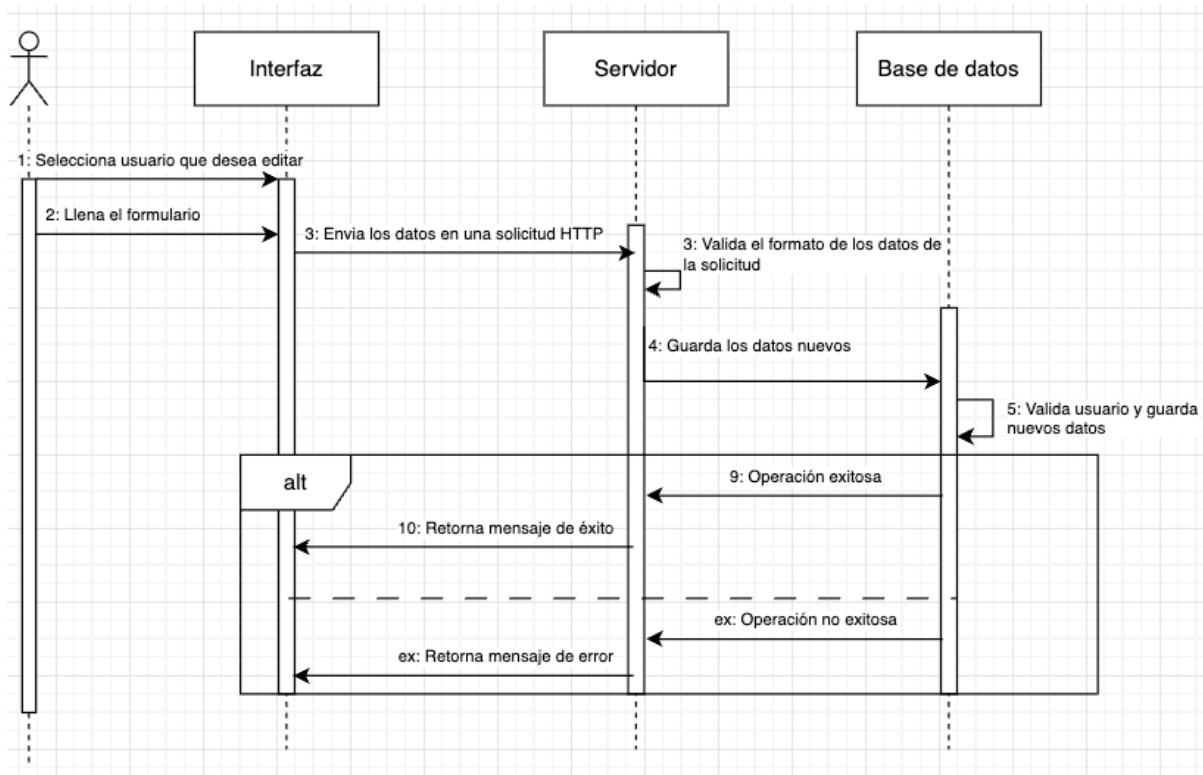
Editar usuario

Caso de uso

Caso de uso	Editar usuario	
Actores	Admin	
Descripción	El administrador ingresa a dashboard en la pantalla de gestión del tipo de usuario que desee editar (cliente, entrenador, administrador).	
Precondición	Haber iniciado sesión al sistema con una cuenta de administrador y que exista el usuario a editar.	
Flujo principal	Paso	Acción
	1	En el usuario a editar selecciona opción de editar.
	2	Ingresa los datos del formulario y da clic a enviar formulario.

	3	El sistema valida los datos.
	4	El usuario se guarda en la base de datos.
	5	El sistema retorna el mensaje de éxito.
Excepciones	Paso	Acción
	1	Si los datos del formulario no son correctos. Repite paso 2.
Post condición	—	
Diagrama	<pre> sequenceDiagram actor Admin as Administrador participant System Note over System: Ingresa a pantalla gestión de tipo de usuario Note over System: <<include>> Note over System: Selecciona usuario y opción editar Note over System: <<include>> Note over System: Llena el formulario y envía Note over System: <<include>> Note over System: Validación de datos Note over System: <<extends>> Devuelve mensaje de error Note over System: <<extends>> Guarda los datos nuevos en la BD Note over System: <<include>> Retorna mensaje de ok </pre>	

Diagrama de secuencia



Eliminar usuario

Caso de uso

Caso de uso	Eliminar usuario	
Actores	Admin	
Descripción	El administrador elimina un usuario	
Precondición	Haber iniciado sesión al sistema con una cuenta de administrador	
Flujo principal	Paso	Acción
	1	Ingresa a la pantalla de todos los usuarios dependiendo el tipo de usuario que quiere eliminar.
	2	Selecciona al usuario y da clic en la opción de eliminar.
	3	El sistema realiza eliminación lógica (soft delete) sobre el usuario en la base de datos.
	4	El sistema retorna el mensaje de éxito.
Excepciones	Paso	Acción

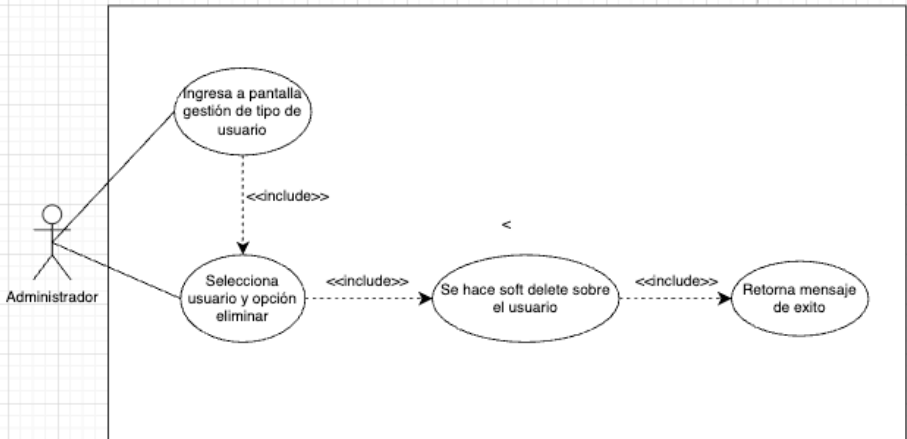
	1	En caso de error al eliminar y procesar información retorna mensaje de error y se repite paso 2.
Post condición	—	
Diagrama	 <pre> graph TD Admin[Administrador] --> UC1((Ingresa a pantalla gestión de tipo de usuario)) UC1 -.-> <<include>> UC2((Selecciona usuario y opción eliminar)) UC2 -.-> <<include>> UC3((Se hace soft delete sobre el usuario)) UC3 -.-> <<include>> UC4((Retorna mensaje de éxito)) </pre>	

Diagrama de secuencia

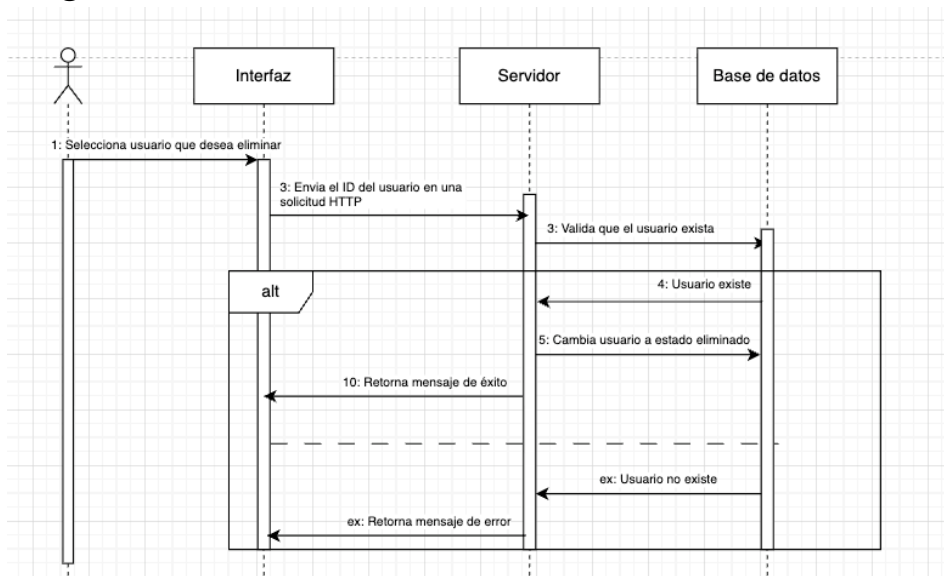
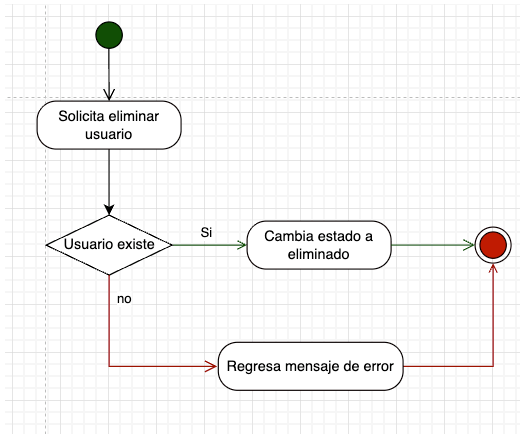


Diagrama de estado



Crear plan

Caso de uso

Caso de uso	Crear plan	
Actores	Admin	
Descripción	El administrador agrega un plan al sistema.	
Precondición	Haber iniciado sesión al sistema con una cuenta de administrador.	
Flujo principal	Paso	Acción
	1	Ingresa a pantalla para agregar plan.
	2	El administrador llena el formulario para agregar un plan nuevo y envía solicitud.
	3	El sistema valida los datos.
	4	El sistema guarda en base de datos el plan y retorna el mensaje de éxito.
Excepciones	Paso	Acción
	1	Si algún campo del formulario es erróneo lanza mensaje error y repite paso 2.
Post condición	---	

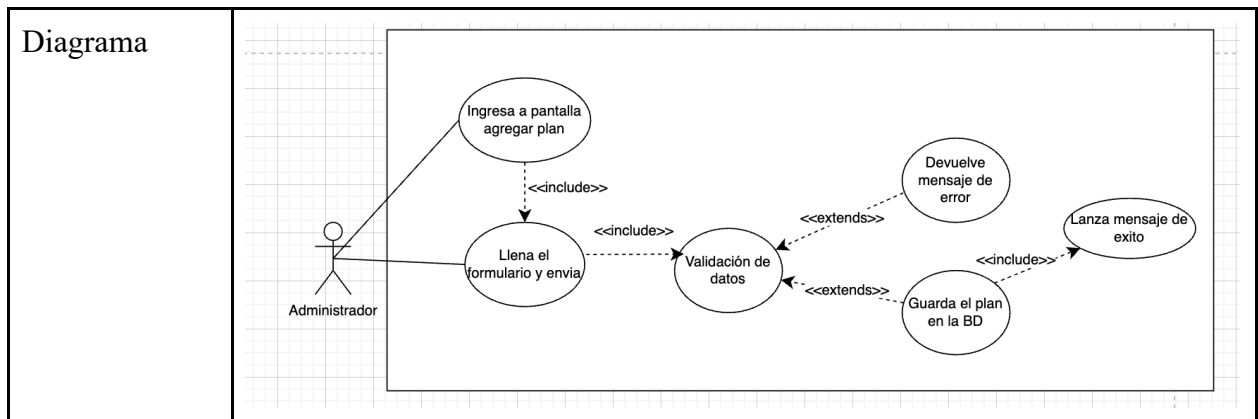
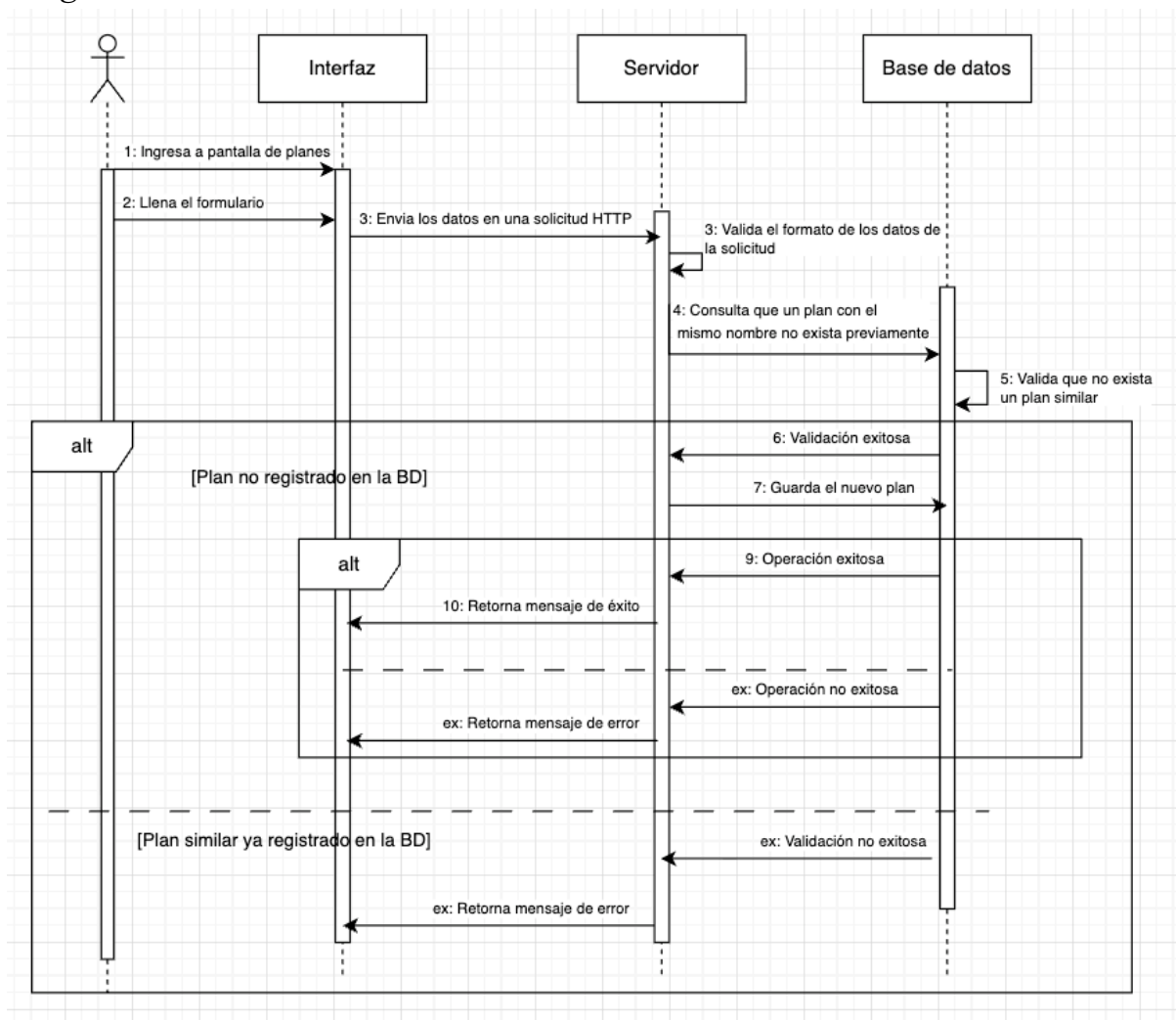


Diagrama de secuencia



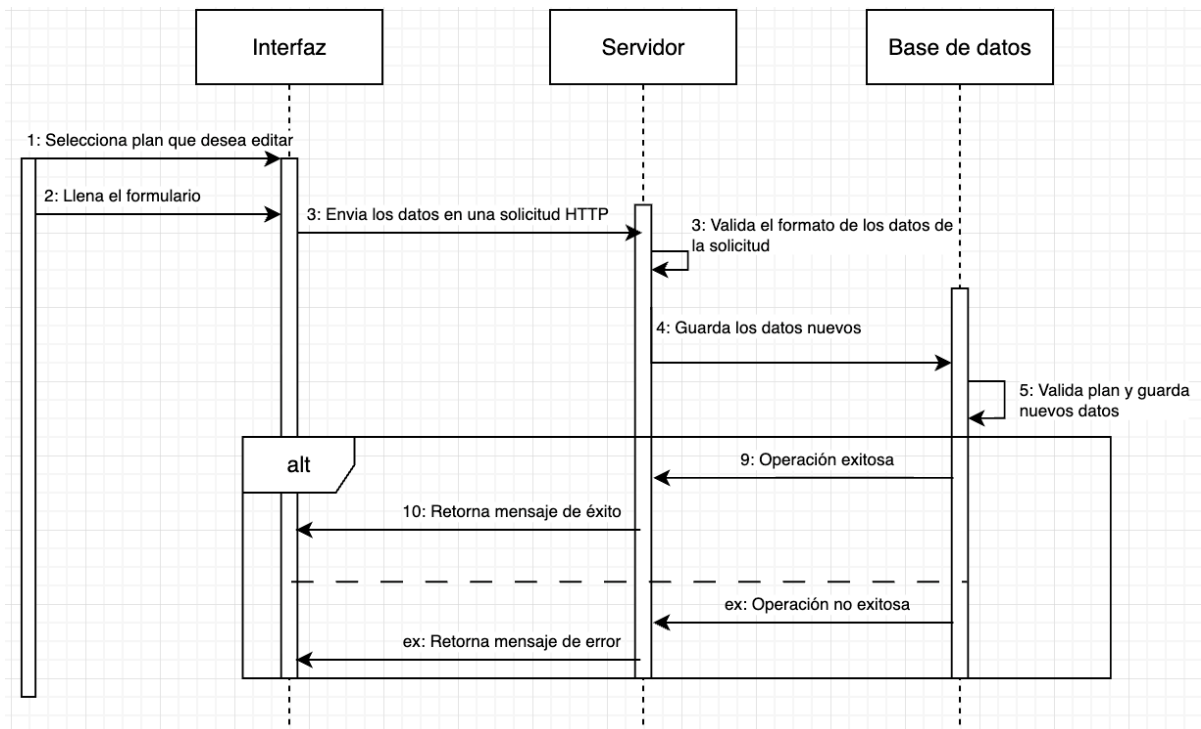
Editar plan

Caso de uso

Caso de uso	Editar plan
-------------	-------------

Actores	Admin	
Descripción	El administrador edita un plan del sistema.	
Precondición	Haber iniciado sesión al sistema con una cuenta de administrador y que el plan exista.	
Flujo principal	Paso	Acción
	1	Ingresa a pantalla para agregar plan.
	2	El administrador llena el formulario para agregar un plan nuevo y envía solicitud.
	3	El sistema valida los datos.
	4	El sistema guarda en base de datos el plan y retorna el mensaje de éxito.
Excepciones	Paso	Acción
	1	Si algún campo es erróneo lanza mensaje error y repite paso 2.
Post condición	—	
Diagrama	<pre> sequenceDiagram actor Admin as Administrador participant System Admin->>System: Ingresa a pantalla agregar plan activate System System->>System: <<include>> Llena el formulario y envia activate System System->>System: <<include>> Validación de datos activate System System-->>System: <<extends>> Devuelve mensaje de error System-->>System: <<extends>> Actualiza el plan en la BD System->>System: <<include>> Lanza mensaje de exito deactivate System </pre>	

Diagrama de secuencia



Eliminar plan

Caso de uso

Caso de uso	Eliminar plan	
Actores	Admin	
Descripción	El administrador elimina un plan del sistema.	
Precondición	Haber iniciado sesión al sistema con una cuenta de administrador y que el plan exista.	
Flujo principal	Paso	Acción
	1	Ingresa a la pantalla de todos los planes.
	2	Selecciona el plan y da clic en opción de eliminar.
	3	El sistema realiza eliminación lógica (soft delete) en la base de datos.

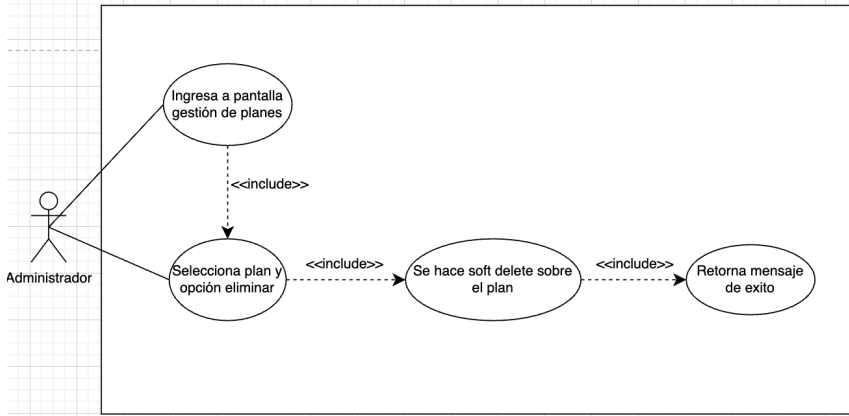
	4	Retorna mensaje de error.
Excepciones	Paso	Acción
	1	En caso de error al eliminar y procesar información retorna mensaje de error y se repite paso 2.
Post condición	—	
Diagrama	 <pre> graph TD Admin[Administrador] --> UC1((Ingresa a pantalla gestión de planes)) UC1 -.-> <<include>> UC2((Selecciona plan y opción eliminar)) UC2 -.-> <<include>> UC3((Se hace soft delete sobre el plan)) UC3 -.-> <<include>> UC4((Retorna mensaje de éxito)) </pre>	

Diagrama de secuencia

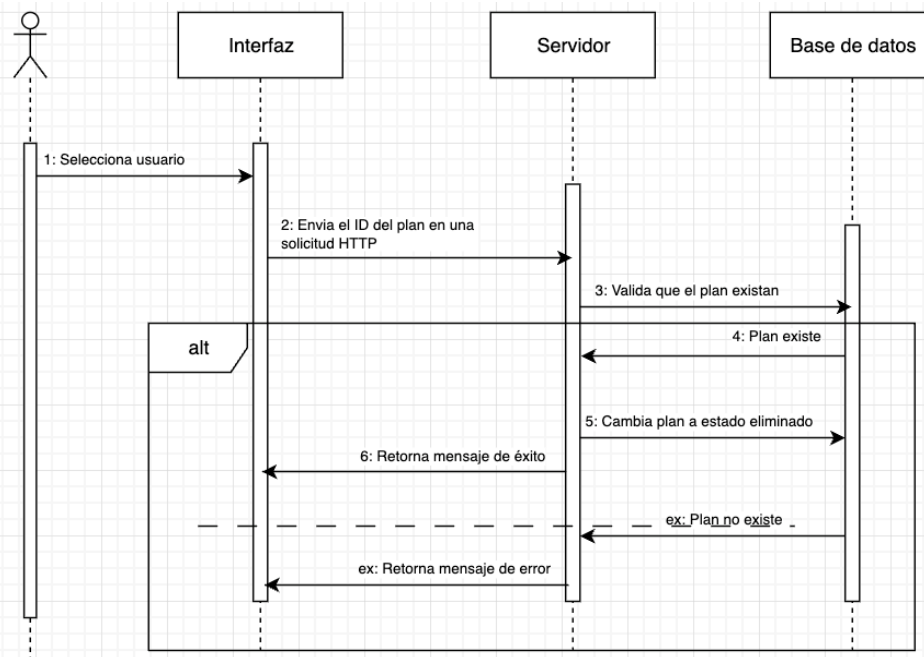
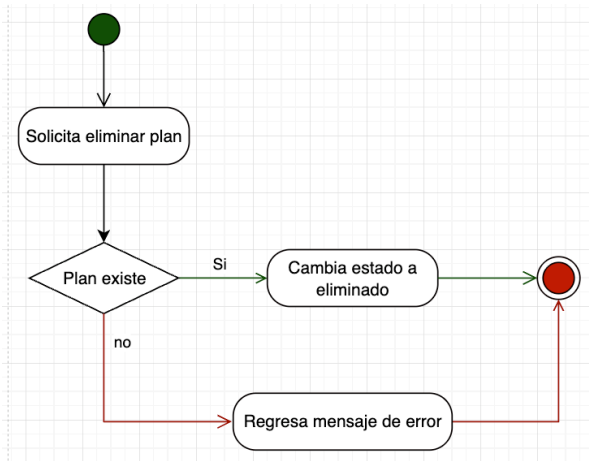


Diagrama de estado



Asignar suscripción

Caso de uso

Caso de uso	Asignar suscripción	
Actores	Admin	
Descripción	El administrador asigna suscripción a cliente seleccionando un plan	
Precondición	Haber iniciado sesión al sistema con una cuenta de administrador	
Flujo principal	Paso	Acción
	1	Ingresa a pantalla de clientes
	2	Selecciona cliente e ingresar a sus suscripciones

	3	Da clic en asignar suscripción
	4	Selecciona plan deseado y envía solicitud
	5	El sistema valida si tiene suscripción activa
	6	El sistema guarda la suscripción en la base de datos y retorna mensaje éxito
Excepciones	Paso	Acción
	1	Si ya tiene suscripción activa retorna el mensaje indicando “ya tiene una suscripción activa” y sale del caso de uso.
Post condición	—	
Diagrama		

Diagrama de secuencia

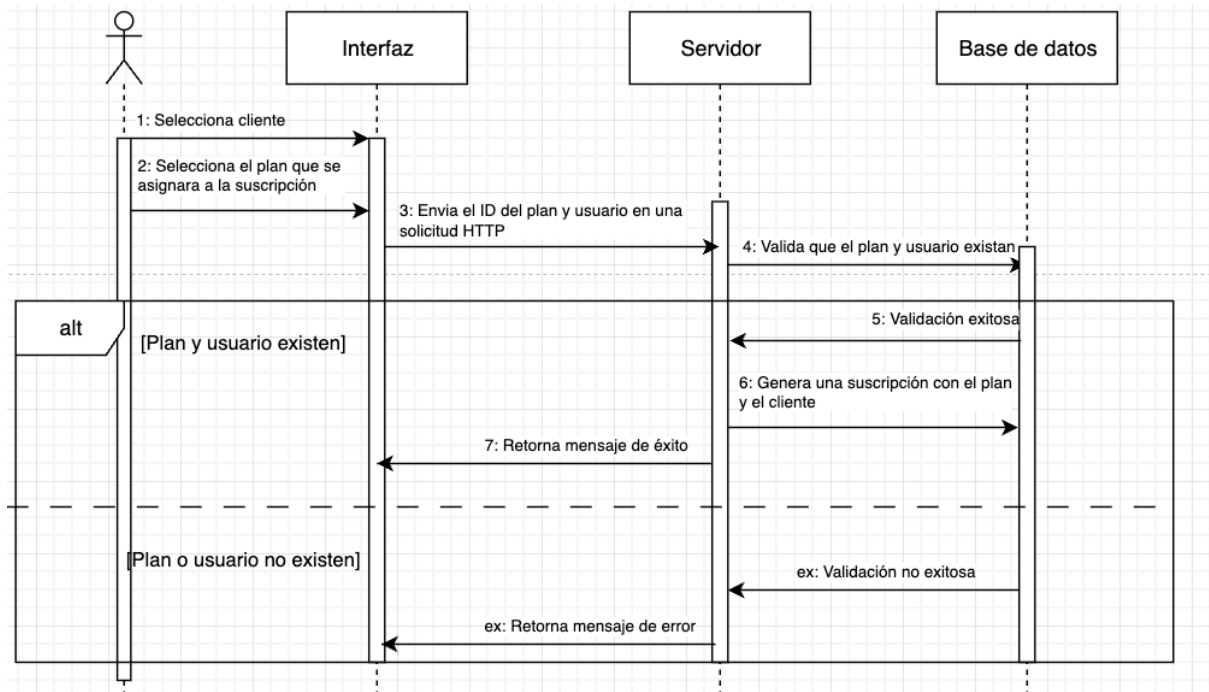
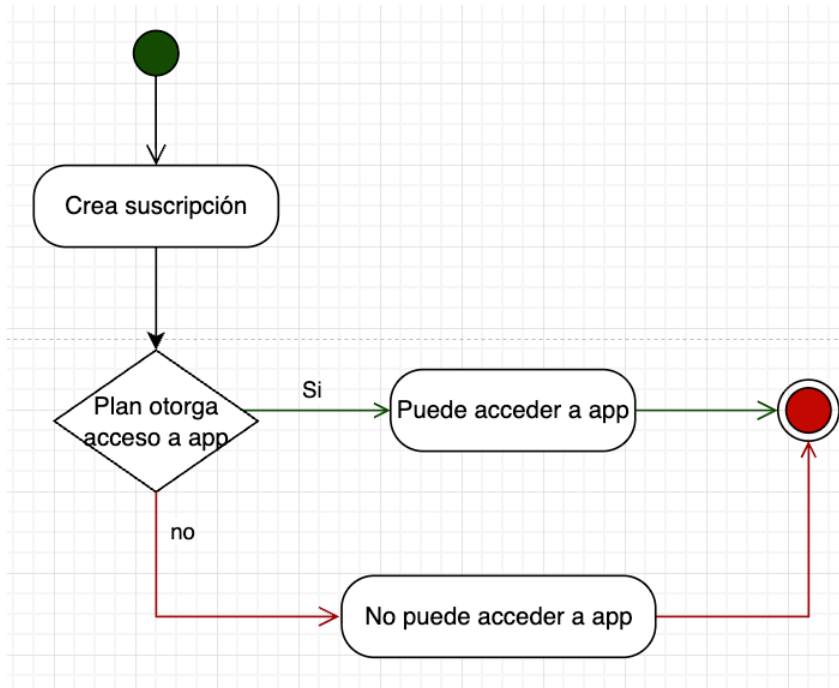


Diagrama de estado



Eliminar suscripción

Caso de uso

Caso de uso	Eliminar suscripción	
Actores	Admin	
Descripción	El administrador podrá eliminar la suscripción del cliente que haya sido asignada el mismo día en caso de errores al asignar la suscripción	
Precondición	Haber iniciado sesión al sistema con una cuenta de administrador, usuario y plan existente. Y la suscripción haya sido asignada en el día que se desea eliminar.	
Flujo principal	Paso	Acción
	1	Ingresa a pantalla de clientes
	2	Selecciona cliente e ingresar a sus suscripciones
	3	Da clic en la opción de eliminar suscripción
	4	El sistema valida si la suscripción fue asignada el mismo día que se intenta eliminar.
	5	Elimina de la base de datos la suscripción y retorna el mensaje de éxito.
Excepciones	Paso	Acción
	1	Si la suscripción fue creada días anteriores a la fecha actual retorna mensaje de error “No se puede eliminar una suscripción que no sea de hoy” y sale del caso de uso.
Post condición	—	

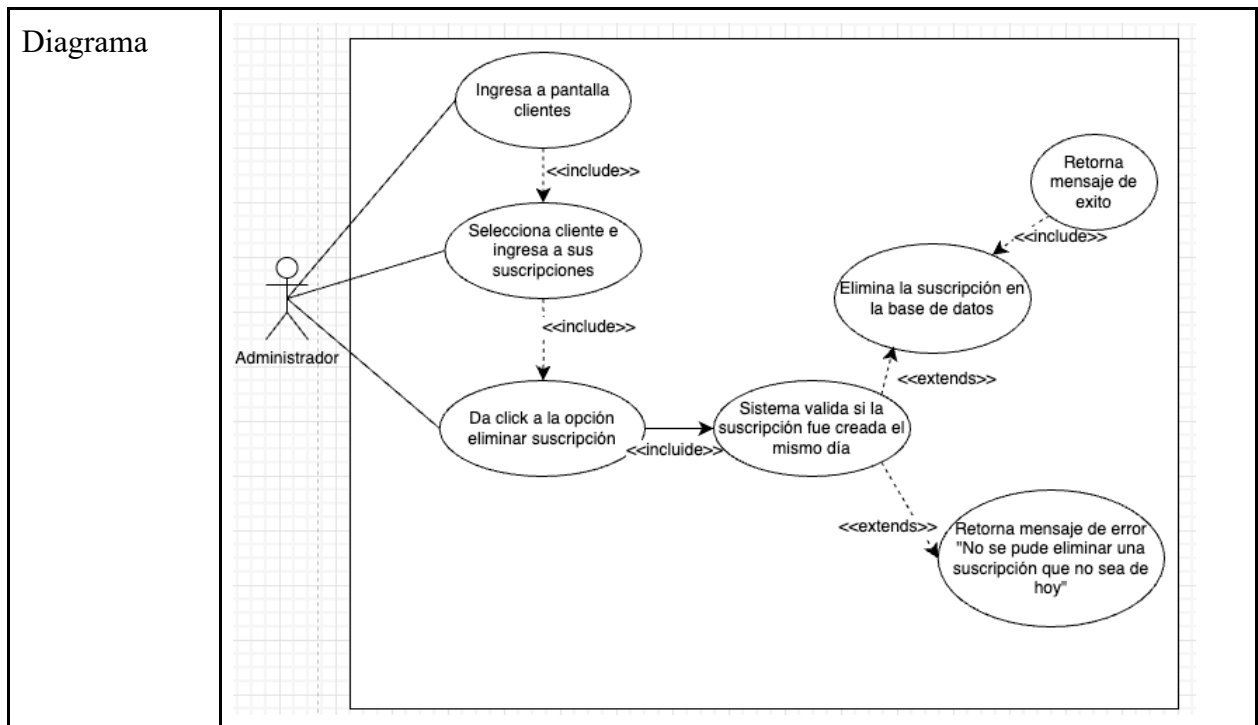
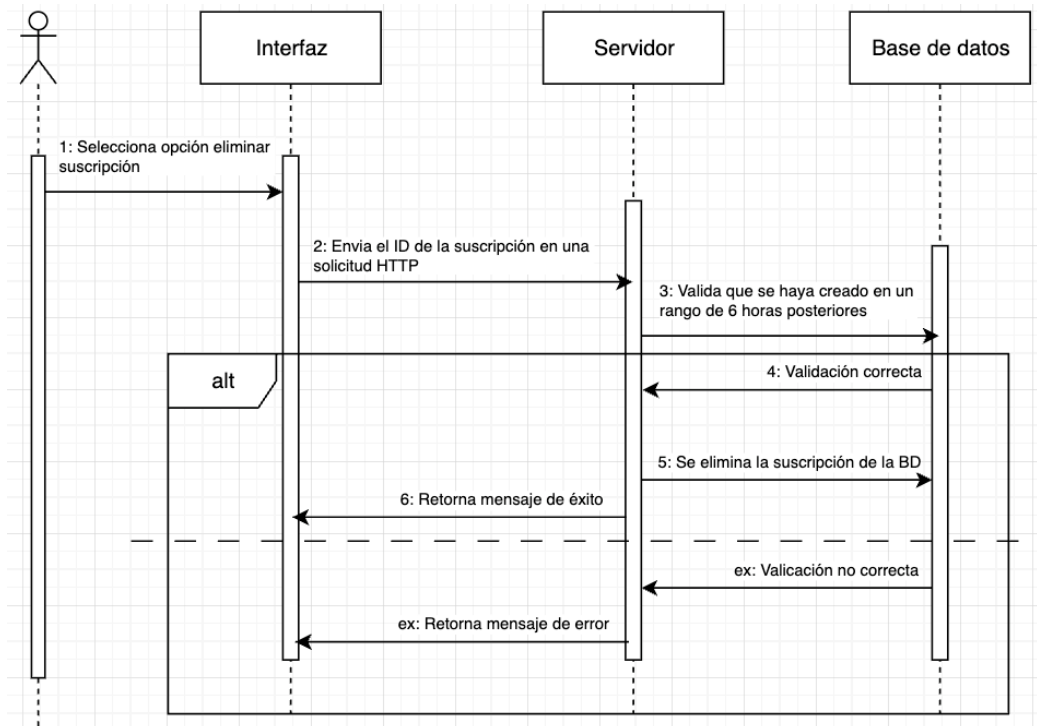


Diagrama de secuencia

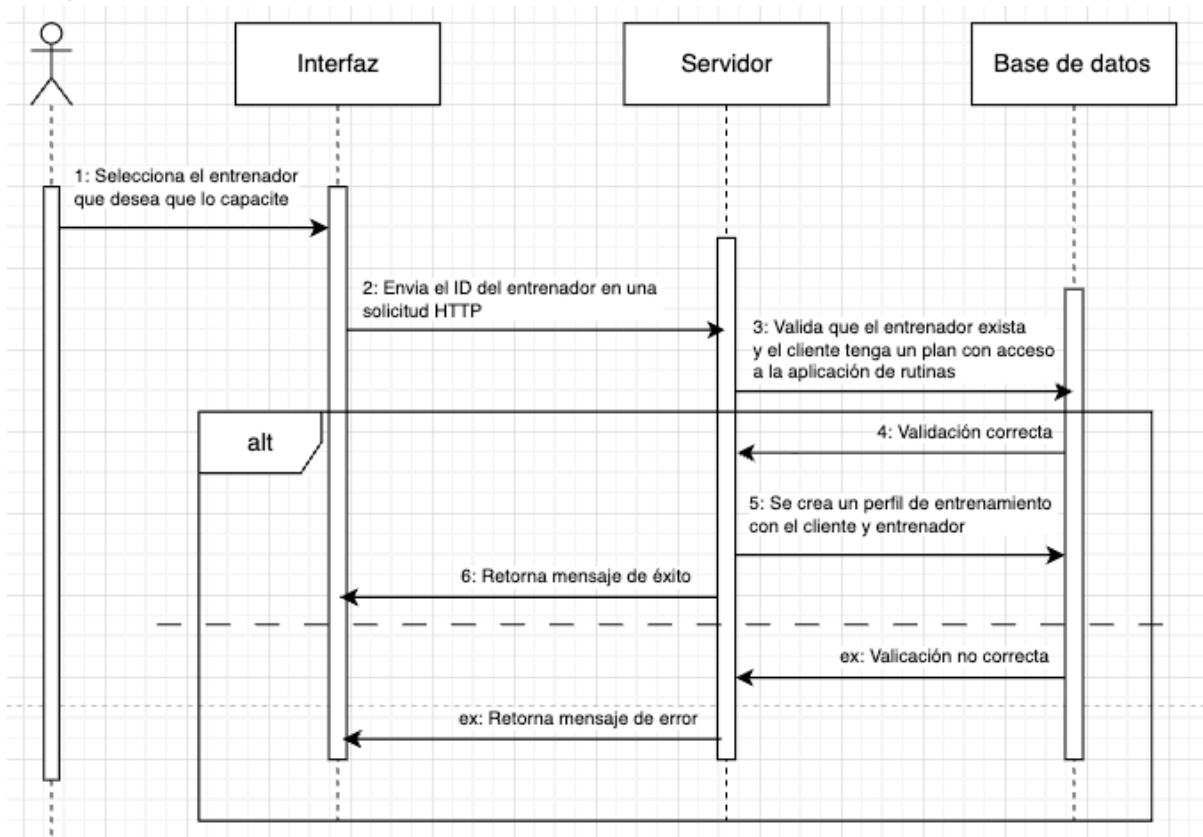


Crear perfil de entrenamiento

Caso de uso

Caso de uso	Crear perfil de entrenamiento	
Actores	Cliente	
Descripción	El cliente podrá crear un perfil de entrenamiento seleccionando a un entrenador que será el que le asigne las rutinas de entrenamiento. Solo se puede crear un perfil por cliente.	
Precondición	Haber iniciado sesión al sistema con una cuenta de cliente y tener una suscripción con un plan que permita la gestión de rutinas.	
Flujo principal	Paso	Acción
	1	Ingresa a pantalla rutinas.
	2	Selecciona al entrenador que desea tener como instructor.
	3	Da clic en asignar entrenador.
	4	El sistema guarda la información en un perfil de relación entre entrenador y cliente en la base de datos.
	5	El sistema retorna el mensaje de éxito.
Excepciones	Paso	Acción
	1	En caso de error al almacenar y procesar información retorna el mensaje de error y regresa al paso 2.
Post condición	Editar el perfil de entrenamiento para completar la información de objetivo, notas y métricas como el peso corporal.	
Diagrama	<pre> graph TD Cliente((Cliente)) U1([Ingresa a pantalla rutinas]) U2([Selecciona al entrenador que desee tener como instructor]) U3([Da click en asignar entrenador]) U4([El sistema procesa y guarda la información en la base de datos.]) U5([Retorna mensaje de éxito]) U6([Retorna mensaje error]) Cliente --> U1 U1 -.-> <<include>> U2 U2 -.-> <<include>> U3 U3 -.-> <<include>> U4 U4 -.-> <<extends>> U5 U4 -.-> <<extends>> U6 U6 -.-> U2 </pre> The diagram illustrates the process of creating a training profile. It starts with the 'Cliente' actor initiating the process by entering the routines screen. This leads to selecting a trainer, which then involves clicking to assign the trainer. The system then processes and saves the information in the database. Depending on the outcome, it either returns a success message or an error message, which loops back to the selection step.	

Diagrama de secuencia

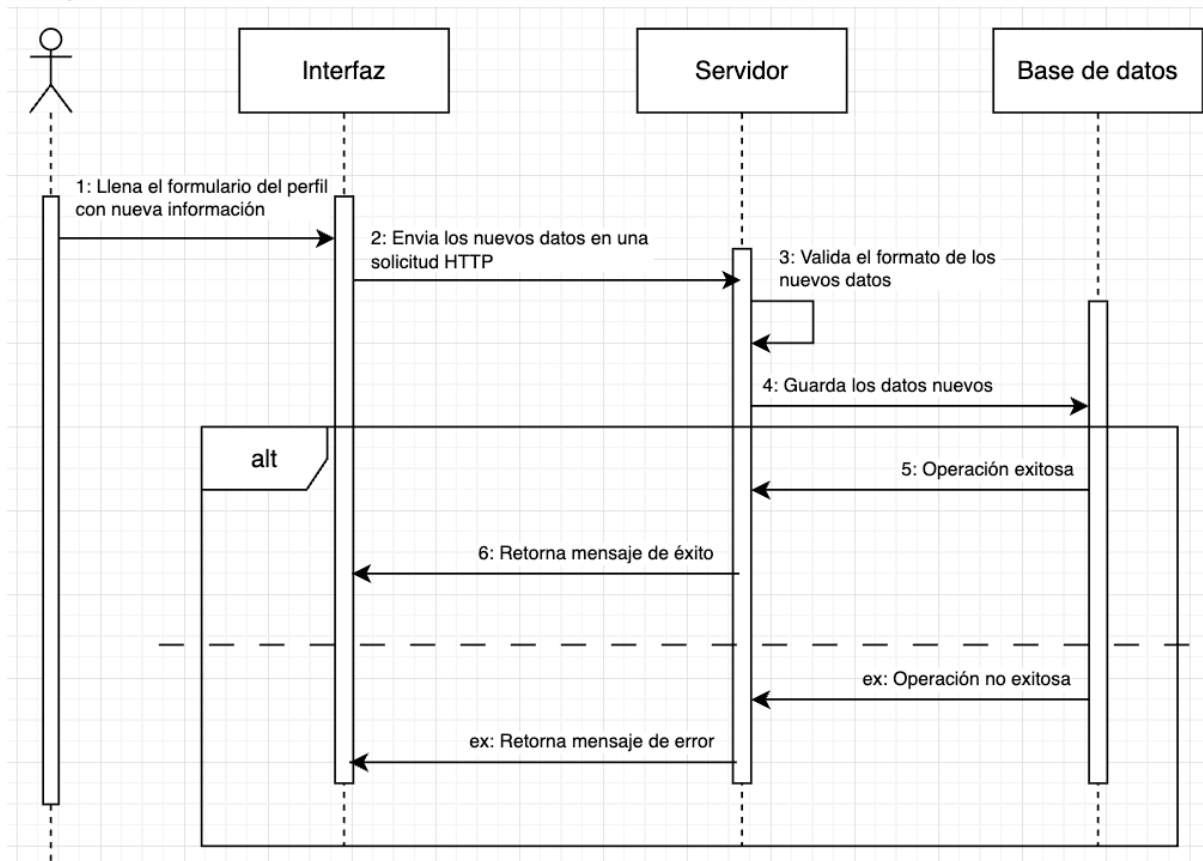


Editar perfil de entrenamiento

Caso de uso

Caso de uso	Editar perfil de entrenamiento	
Actores	Cliente	
Descripción	El cliente podrá editar el perfil de entrenamiento para ajustar la información que necesita conocer el entrenador para planificar sus entrenamientos de manera correcta.	
Precondición	Haber iniciado sesión al sistema con una cuenta de cliente, tener una suscripción con un plan que permita la gestión de rutinas y tener un perfil de entrenamiento creado.	
Flujo principal	Paso	Acción
	1	Ingresa a pantalla rutinas.
	2	Da clic en editar perfil entrenamiento.
	3	Completa el formulario con su información y guarda cambios.
	4	El sistema actualiza la información del perfil de entrenamiento en la base de datos.
	5	El sistema retorna el mensaje de éxito.
Excepciones	Paso	Acción
	1	En caso de error al almacenar y procesar información retorna el mensaje de error y regresa al paso 3.
Post condición	—	
Diagrama	<pre> graph TD Cliente((Cliente)) UC1([Ingresa a pantalla rutinas]) UC2([Da click en editar perfil de entrenamiento]) UC3([Completa formulario y da click en guardar cambios]) UC4([El sistema procesa y guarda la información en la base de datos.]) UC5([Retorna mensaje de éxito]) UC6([Retorna mensaje error]) Cliente --- UC1 Cliente --- UC2 Cliente --- UC3 UC1 -.-> <<include>> UC2 UC2 -.-> <<include>> UC3 UC3 -.-> <<include>> UC4 UC4 -.-> <<extends>> UC5 UC4 -.-> <<extends>> UC6 </pre> The diagram illustrates the process of editing a training profile. It starts with the 'Cliente' actor interacting with three use cases: 'Ingresa a pantalla rutinas', 'Da click en editar perfil de entrenamiento', and 'Completa formulario y da click en guardar cambios'. These steps are linked sequentially with 'include' relationships. The final step, 'Completa formulario y da click en guardar cambios', leads to 'El sistema procesa y guarda la información en la base de datos.', which then branches into two 'extends' paths: 'Retorna mensaje de éxito' and 'Retorna mensaje error'.	

Diagrama de secuencia



Crear rutina

Caso de uso

Caso de uso	Crear rutina	
Actores	Entrenador	
Descripción	El entrenador podrá crear rutinas para los clientes con quienes comparten una relación en el perfil de entrenamiento.	
Precondición	Haber iniciado sesión al sistema con una cuenta de entrenador, tener un perfil de entrenamiento con algún cliente.	
Flujo principal	Paso	Acción
	1	Ingresa a pantalla asesorados.
	2	Da clic en cliente que se le asignará una rutina.
	3	Da clic en agregar rutina.
	4	Completa el formulario y da clic en guardar rutina.
	5	El sistema valida los datos.
	6	El sistema guarda la rutina en la base de datos y retorna el mensaje de éxito.
Excepciones	Paso	Acción
	1	En caso de error al almacenar y procesar información retorna mensaje de error y se repite paso 4.
Post condición	—	

Diagrama

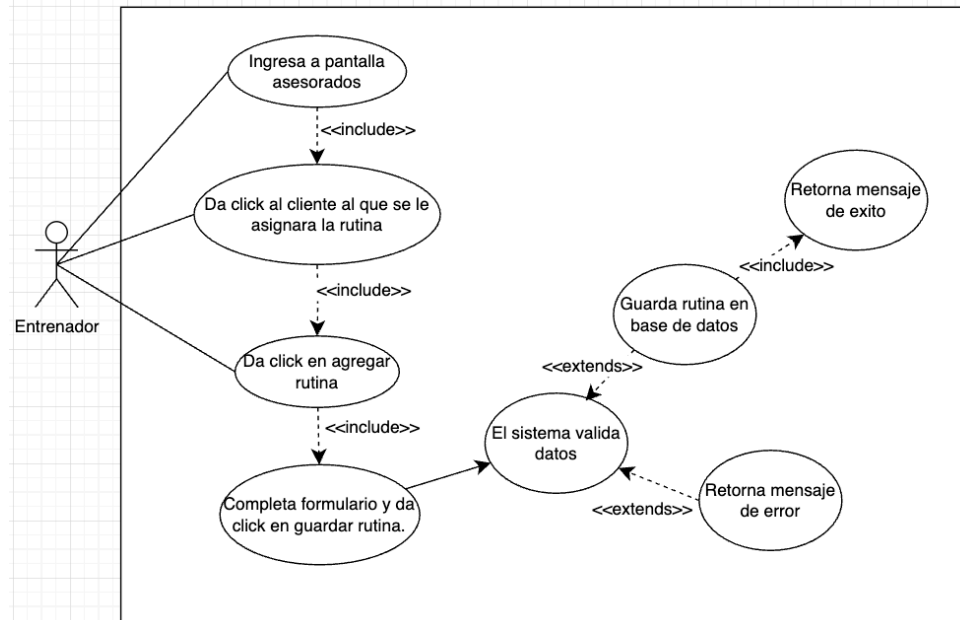
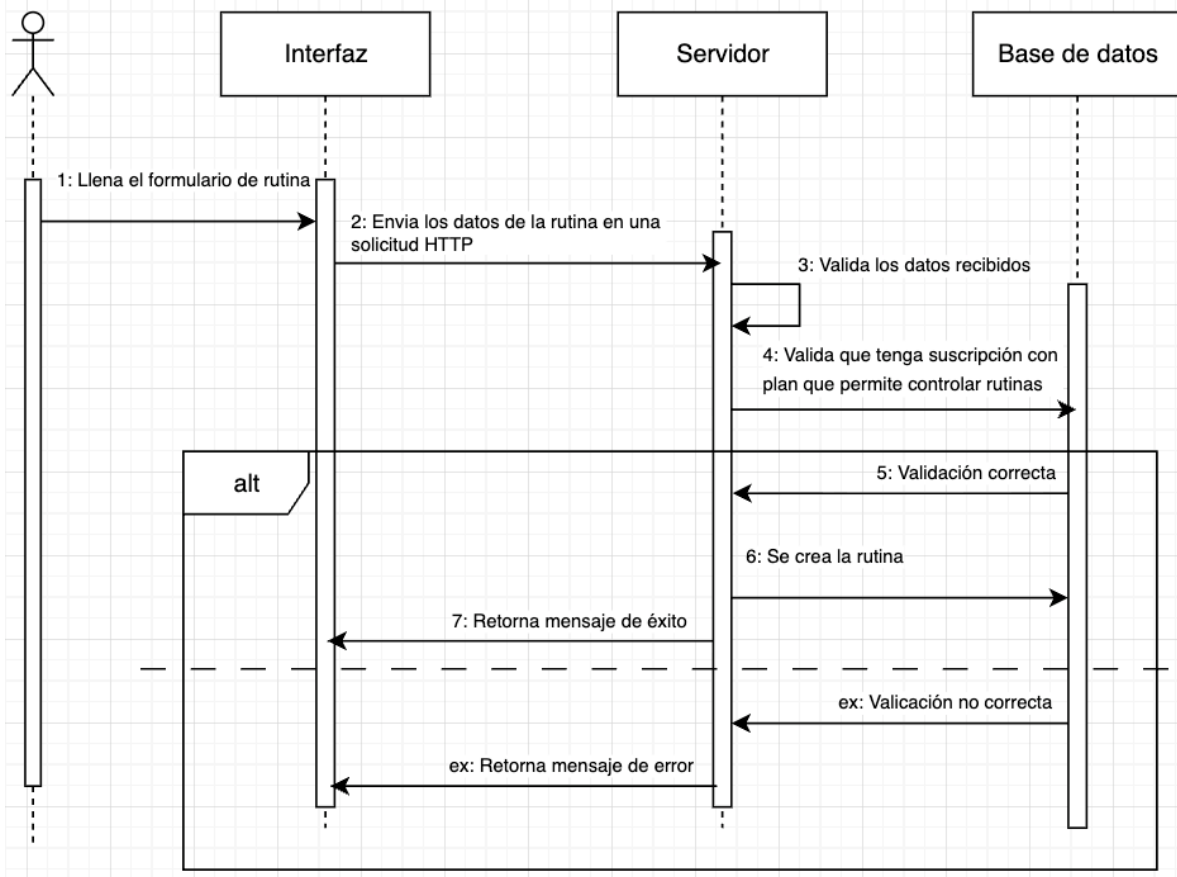


Diagrama de secuencia



Editar rutina

Caso de uso

Caso de uso	Editar rutina	
Actores	Entrenador	
Descripción	El entrenador podrá editar las rutinas para los clientes con quienes comparten una relación en el perfil de entrenamiento.	
Precondición	Haber iniciado sesión al sistema con una cuenta de entrenador, tener un perfil de entrenamiento con algún cliente.	
Flujo principal	Paso	Acción
	1	Ingresa a pantalla asesorados.
	2	Da clic en el cliente y el sistema muestra todas las rutinas del cliente.
	3	Da clic en la rutina que se desee editar.
	4	Da clic en activar edición.
	5	Hace cambios en el formulario y da clic en guardar rutina.
	6	El sistema valida los datos.
	7	El sistema actualiza la rutina en la base de datos y retorna el mensaje de éxito.
Excepciones	Paso	Acción
	1	En caso de error al almacenar y procesar información retorna mensaje de error y se repite paso 5.
Post condición	—	

Diagrama

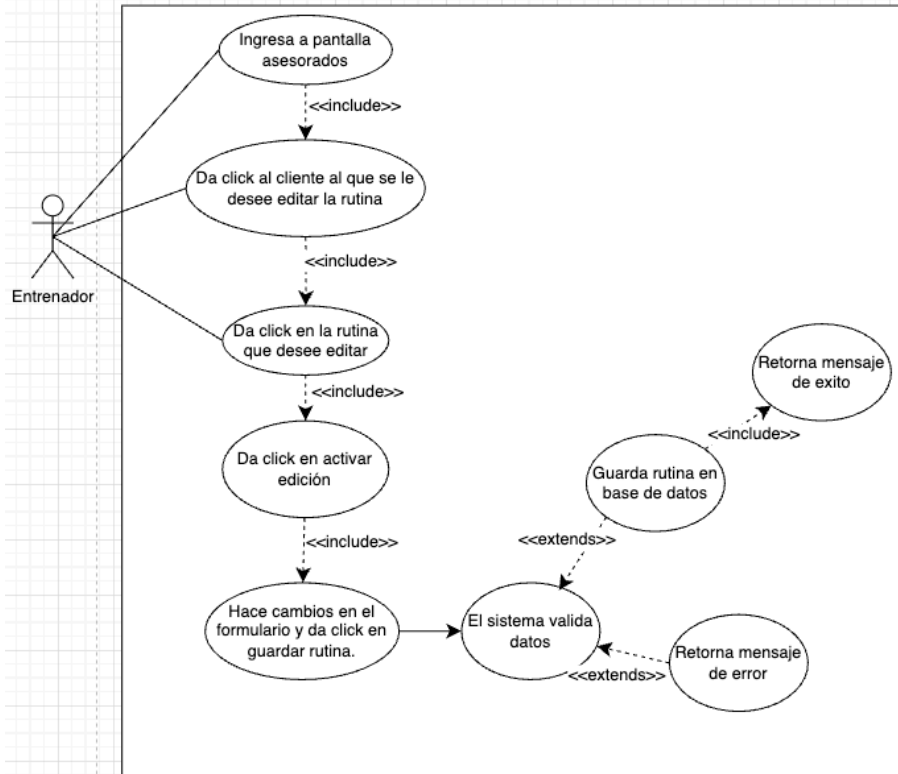
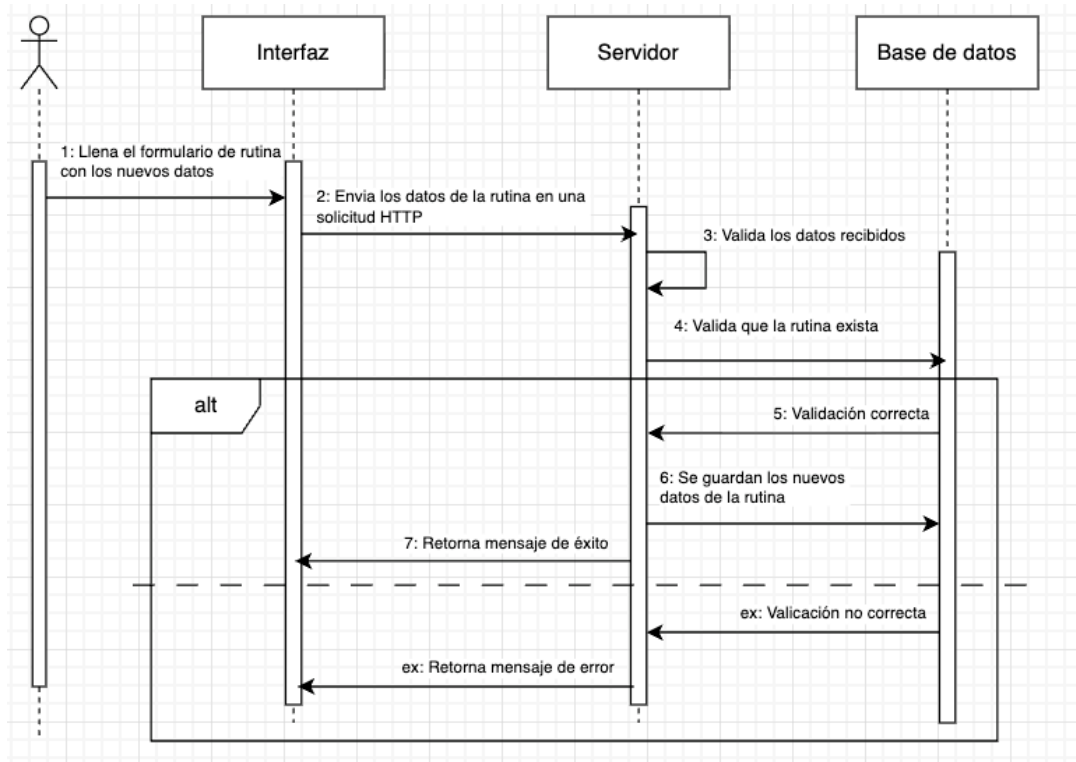


Diagrama de secuencia

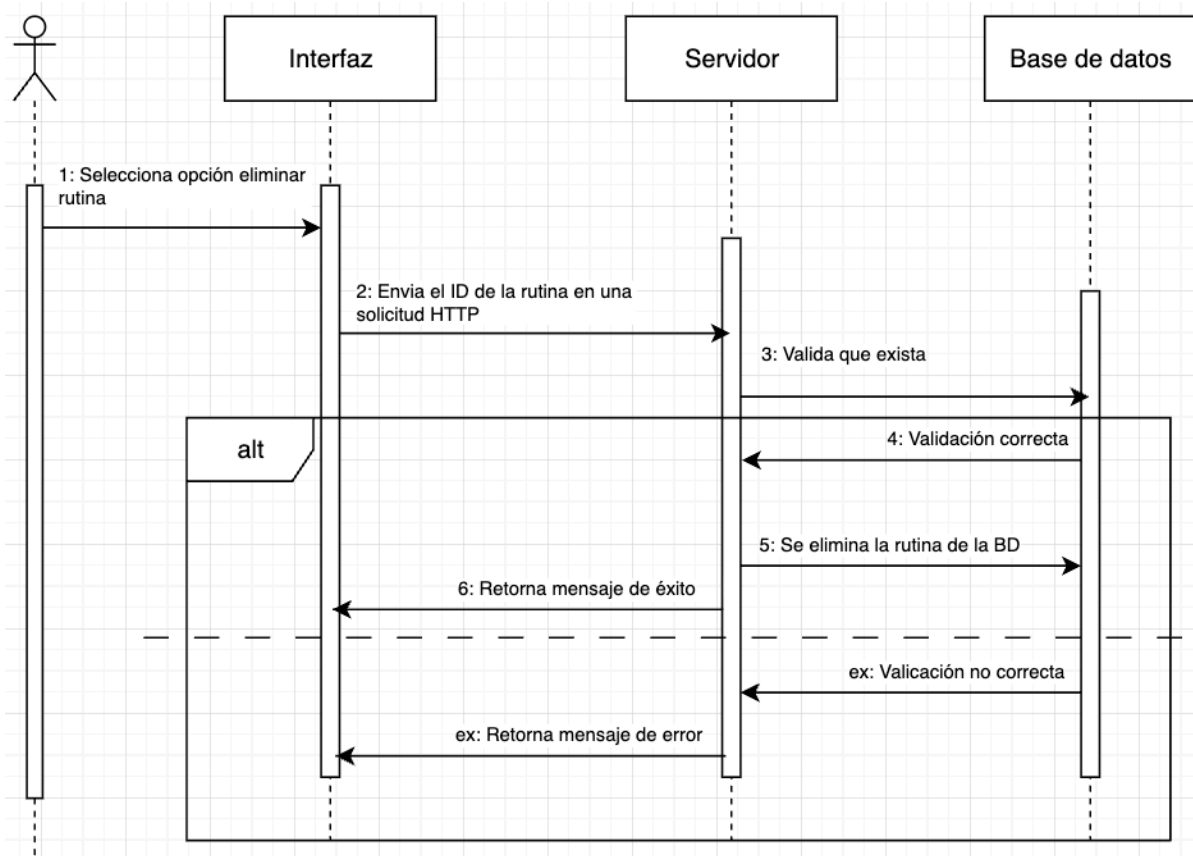


Eliminar rutina

Caso de uso

Caso de uso	Eliminar rutina	
Actores	Entrenador	
Descripción	El entrenador podrá eliminar las rutinas para los clientes con quienes comparten una relación en el perfil de entrenamiento.	
Precondición	Haber iniciado sesión al sistema con una cuenta de entrenador, tener un perfil de entrenamiento con algún cliente y rutina existente.	
Flujo principal	Paso	Acción
	1	Ingresa a pantalla asesorados.
	2	Da clic en el cliente y el sistema muestra las rutinas del cliente.
	3	Da clic en la opción eliminar a la rutina que desee eliminar.
	4	El sistema elimina la rutina.
	5	Retorna mensaje de éxito.
Excepciones	Paso	Acción
	1	En caso de error al eliminar y procesar información retorna mensaje de error y se repite paso 3.
Post condición	—	
Diagrama	<pre> sequenceDiagram actor Entrenador Entrenador->>U1: Ingresa a pantalla asesorados U1->>U2: <<include>> U2->>U3: Da click al cliente y el sistema muestra todas las rutinas del cliente U3->>U4: <<include>> U4->>U5: Da click en la opción de eliminar de un rutina especifica U5->>U6: <<include>> U6->>U7: El sistema elimina la rutina de la base de datos U7-->>U8: <<extends>> U8->>U9: El sistema retorna mensaje de éxito U7-->>U10: <<extends>> U10->>U11: El sistema retorna mensaje de error U11-->>U5: <<extends>> </pre> El diagrama de flujo de secuencia ilustra el proceso de eliminación de una rutina. Comienza con el actor 'Entrenador' que interactúa con el sistema. El flujo principal sigue los pasos: 'Ingresa a pantalla asesorados', 'Da click al cliente y el sistema muestra todas las rutinas del cliente', 'Da click en la opción de eliminar de un rutina especifica', y 'El sistema elimina la rutina de la base de datos'. Desde el paso de eliminación, hay dos caminos de extensión: uno que devuelve un 'mensaje de éxito' y otro que devuelve un 'mensaje de error', este último lo cual hace que el flujo retorne al paso de selección de la rutina para su eliminación.	

Diagrama de secuencia



3.2 Definición de la arquitectura

La arquitectura de software de un sistema representa las decisiones de diseño relacionadas con su estructura y comportamiento. En otras palabras, hace referencia a la forma en que se organizan los elementos de un sistema en el proceso de su creación.

Dichos elementos pueden hacer referencia a los detalles o el diseño de implementación, tecnológica o la infraestructura. Por lo tanto, una arquitectura de software sólida facilita la adaptación y los ajustes, simplificando la implementación de nuevas funciones y satisfaciendo las demandas cambiantes. (Communications, 2025)

Una buena arquitectura de software debe ser escalable, robusta y permitir trabajar en equipo para que el desarrollo sea incremental. Asimismo, debe testearse para validar que funcione correctamente, comprobar la cobertura del código y soportar reestructuraciones en el código fuente para mejorar su funcionamiento sin alterar su funcionalidad cada que se requiera hacerlas. (Communications, 2025)

3.2.1 Cliente - Servidor

Es un modelo de diseño de software en el que las aplicaciones se dividen en 2 componentes, esta separación facilita la gestión de recursos y permite que interactúen de manera eficiente.

El cliente es el componente que inicia las solicitudes de servicio, generalmente se ejecuta en la máquina del usuario y se presenta en varias formas como; navegador web, aplicación de escritorio o móvil. Su función es enviar peticiones al servidor a través de la red y recibir respuesta para mostrarlas en una interfaz.

El servidor es el componente que responde a las solicitudes que recibe del cliente, almacena y gestiona recursos, que pueden incluir bases de datos, archivos y aplicaciones. Su procesamiento es crucial ya que maneja varias peticiones simultáneamente y devuelve respuestas de manera eficiente. Suelen tener especificaciones y configuraciones dependiendo su propósito (servir páginas web o gestión de bases de datos y archivos). (Staff, 2024)

La comunicación entre los dos componentes se realiza a través de protocolos de red que dictan cómo se deben enviar y recibir los datos. El protocolo más utilizado es el TCP/IP, donde el *Transmission Control Protocol* (TCP) es el encargado de garantizar la entrega segura y ordenada de paquetes de datos, el *Internet Protocol* (IP) se encarga de la dirección y el envío de esos paquetes a través de la red. Otros protocolos son el HTTP/HTTPS para la comunicación en la web y FTP para archivos. (Staff, 2024)

Esta arquitectura es considerada la mejor opción para el núcleo del sistema al darnos diferentes ventajas como; la centralización de gestión de datos desde el servidor para tener la información actualizada todo el tiempo, integridad, consistencia de datos, seguridad mejorada al proteger los servidores con firewalls, sistemas de detección de intrusos, autenticación robusta, escalabilidad al dividir interfaces y lógica para un desarrollo independiente.

A pesar de que el núcleo del software se realizará con cliente - servidor es necesario definir una estructura modular para el backend el cual necesita de una estructura clara para adaptarse a mejoras continuas para ello se considera clean architecture como la mejor opción.

3.2.2 Clean architecture (backend)

La arquitectura limpia o clean architecture en inglés es un enfoque de diseño de software propuesto por Robert C. Martin en su libro "*Clean Architecture: A Craftsman's Guide to Software Structure and Design*". La idea fundamental es que la estructura debe enfocarse en separar las preocupaciones del negocio de las preocupaciones técnicas con el objetivo de crear

un software de alta calidad, sencillo de modificar y que se adapte fácilmente a las necesidades cambiantes del negocio. (Martin, Robert C, 2018)

Esta arquitectura se caracteriza por los siguientes puntos:

- **Separación de preocupaciones:** separa código relacionado con la lógica del negocio del código que se encarga de la infraestructura técnica como la base de datos o la interfaz del usuario.
- **Capas de abstracción:** cada capa tiene una responsabilidad clara y definida. Las capas internas contienen la lógica de negocio mientras las externas manejan la infraestructura técnica permitiendo que cada capa pueda cambiar o reemplazar sin afectar a otra.
- **Principios SOLID:** estos principios nos indican cómo manejar nuestras funciones y estructura de datos dentro de clases y cómo esas clases deben ir conectadas. Una clase es un simple grupo de funciones y datos. El objetivo de estos principios es crear una estructura de software mid-level que tolere cambios, sea fácil de entender y son las bases de los componentes para que puedan ser usados en distintos sistemas. Esta es una descripción de cada principio que se cumple:
 - SRP (Single Responsibility Principle): el principio de responsabilidad única indica que cada módulo de software tiene una y solo una razón de cambiar. En otros términos, cada parte del código debe tener una sola responsabilidad o trabajo definido.
 - OCP (Open Closed Principle): especifica que cada entidad (clases, módulos, etc.) debe ser abiertos a extensiones, pero cerrados a modificaciones. Lo que nos da posibilidad de añadir funcionalidades sin alterar el código existente.
 - LSP (Liskov Substitution Principle): dice que, para construir un sistema a partir de partes intercambiables, esas partes deben adherirse a un contrato que permita que esas partes sean sustituidas por otras. Esto asegura que la herencia se use de manera adecuada y que el código sea más robusto y fácil de mantener.
 - ISP (Interface Segregation Principle): el principio de segregación de interfaces establece que el cliente no debe verse obligado a depender de métodos que no se utilizan. Significa dividir las interfaces grandes y generales en más pequeñas y específicas de modo que las clases solo necesiten implementar los métodos relevantes. Ayuda a reducir la complejidad del código mejorando la mantenibilidad y evitar dependencias innecesarias.
 - DIP (Dependency Inversion Principle): establece que módulos de alto nivel no deben depender de módulos de bajo nivel. Ambos deben depender de

abstracciones y esas abstracciones no deben depender de detalles, en vez de eso los detalles deben depender de abstracciones. Promueve el acoplamiento flexible al confiar en interfaces o clases abstractas en lugar de implementaciones concretas.

En la imagen 11 se puede apreciar la arquitectura completa y como está formada.

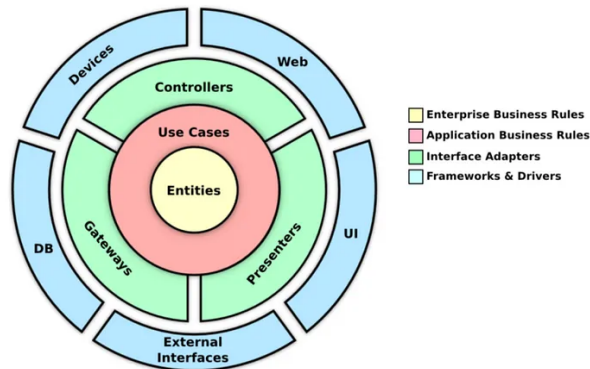


Imagen 11: Clean Architecture Estructura

3.3 Diseño de interfaces de usuario:

3.3.1 Autenticación

Login diseño (imagen 12)

Diseño de Login. La interfaz tiene un fondo oscuro. En el centro hay un formulario con el título 'Inicio de sesión'. Hay dos campos de entrada: 'Email' con el valor 'correo@correo.com' y 'Password' con caracteres ocultos por asteriscos. Debajo de los campos hay un botón rojo con el texto 'Iniciar sesión'. En la parte inferior del formulario hay tres enlaces: '¿No Tienes cuenta? Registerate' y 'Olvide contraseña'.

Imagen 12: Login diseño

Registro para miembros diseño (imagen 13)

Crear cuenta

Nombre
Ej. Julian

Apellidos
Ej. Pérez Gómez

Correo electrónico
Ej. usuario@ejemplo.com

Contraseña
Mínimo 8 caracteres

Teléfono
Ej. 1234567890

Género
Seleccione una t

Fecha de nacimiento
dd/mm/yyyy

Registrar cuenta

[¿Ya tienes cuenta? Inicia sesión](#)

Imagen 13: Registro diseño

Confirmar cuenta template correo (imagen 14)

KingsLayer Gym

Hola Felipe Gúzman Andrés,

¡Gracias por registrarte en KingsLayer Gym! Tu cuenta está casi lista.

Haz clic en el botón:

Confirmar Cuenta

Si no solicitaste este correo, puedes ignorarlo.

Imagen 14: Template correo diseño

Confirmar cuenta ÉXITO diseño (imagen 15)

Cuenta Confirmada correctamente :)

¡Cuenta confirmada!

Tu cuenta ha sido verificada correctamente.

Serás redirigido automáticamente...

Imagen 15: Cuenta confirmada diseño

Confirmar cuenta ERROR diseño (imagen 16)

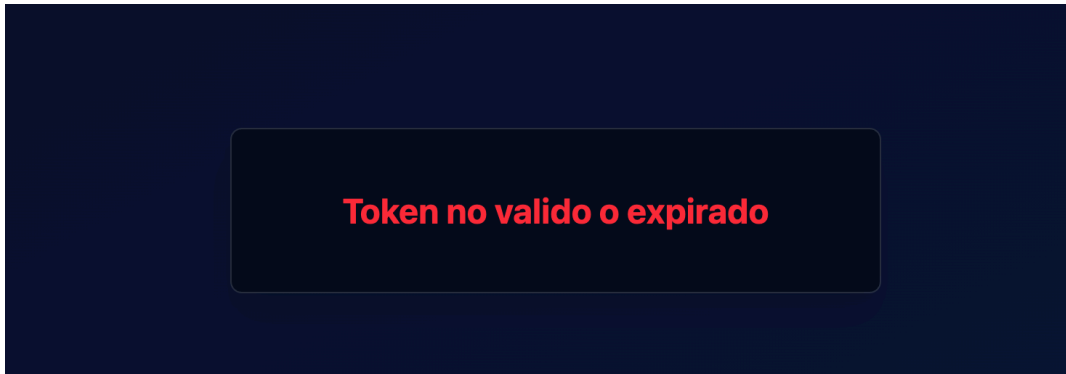


Imagen 16: Token no válido diseño

Olvide contraseña formulario diseño (imagen 17)

A dark blue background with a central rounded rectangle containing a password reset form. The form has the title "Olvidé contraseña", a subtitle "Ingresa tu correo para enviar instrucciones", an "Email" label, a text input field with the placeholder "correo@correo.com", and a purple button labeled "Solicitar reestablecer".

Imagen 17: Olvide contraseña diseño

3.3.2 Dashboard - Administrador

Agregar clientes diseño (imagen 18), se usa el mismo formulario para editar datos

Kings Layer

Usuarios

- Usuarios
- Agregar usuario**
- Clientes
- Entrenadores
- Administradores
- Planes

Tipo de usuario

- Cliente**
- Administrador
- Entrenador

Cliente

Ingresa la información del cliente, indique al cliente la matrícula que te da como respuesta el envío del formulario

Nombre(s): Oliver

Apellidos: Andrés Gómez

Correo: correo@correo.com

Télefono: 2221234789

Genero: Seleccione una opción

Nacimiento: dd/mm/yyyy

Enviar formulario

Cerrar sesión

Imagen 18: Agregar cliente diseño

Agregar administrador diseño (imagen 19), se usa el mismo formulario para editar datos

Kings Layer

Usuarios

- Usuarios
- Agregar usuario**
- Clientes
- Entrenadores
- Administradores
- Planes

Tipo de usuario

- Cliente
- Administrador**
- Entrenador

Admin

Ingresa la información del administrador y asigna niveles de acceso

Nombre(s): Oliver

Apellidos: Andrés Gómez

Correo: correo@correo.com

Télefono: 2221234789

Permisos completos: ☒ **** El acceso completo da acceso a funcionalidades como eliminado en bloque**

Enviar formulario

Imagen 19: Agregar administrador diseño

Agregar entrenador diseño (imagen 20), se usa el mismo formulario para editar datos

Kings Layer

Agregar usuario

Tipo de usuario

- Cliente
- Administrador
- Entrenador**

Entrenador

Ingresar la información del entrenador

Nombre(s): Carlos

Apellidos: Andrés Gómez

Correo: correo@correo.com

Teléfono: 2221234789

Descripción breve: Presenta al entrenador en un párrafo corto

Habilidades: + Agregar Beneficio #1

Enviar formulario

Imagen 20: Agregar entrenador diseño

Todos los administradores diseño (imagen 21)

Kings Layer

Administradores

Centro de control de administradores

Todos los administradores

	NOMBRE	TELEFONO	CUENTA	ACCESO	OPCIONES
1	JC Joaquin Cervantes Prueba joaq@admin.com	121-313-4123	Pendiente	LIMITED	
2	AS admin super admin admin@gymapp.com	111-111-1111	Confirmada	FULL	

Imagen 21: Todos los administradores diseño

Todos los entrenadores diseño (imagen 22)

Kings Layer

Entrenadores

Centro de control de entrenadores

Todos los entrenadores

	NOMBRE	TELEFONO	CUENTA	HABILIDADES	OPCIONES
1	LJ Luis Jimenez Vazquez correo@luis.com	123-123-1231	Pendiente	Nutrición	
2	XZ Ximena Zurita ximena@correo.com	342-341-3413	Confirmada	Musculación	
3	PR Pedro Romero Velez pedrop@correo.com	234-123-4234	Confirmada	Gimnasia	
4	AM Alex Martinez Cervantes alex@correo.com	232-341-3412	Pendiente	Nutrición Calistenia	
5	CB Christian Bumsted Martinez cris@correo.com	231-231-2312	Pendiente	Musculación Cross - Fit	
6	CB Carlos Belcast Guzman andoni@correo.com	332-421-4231	Pendiente	Nutricion Musculacion Cross Fit	

Imagen 22: Todos los entrenadores diseño

Todos los clientes diseño (imagen 23)

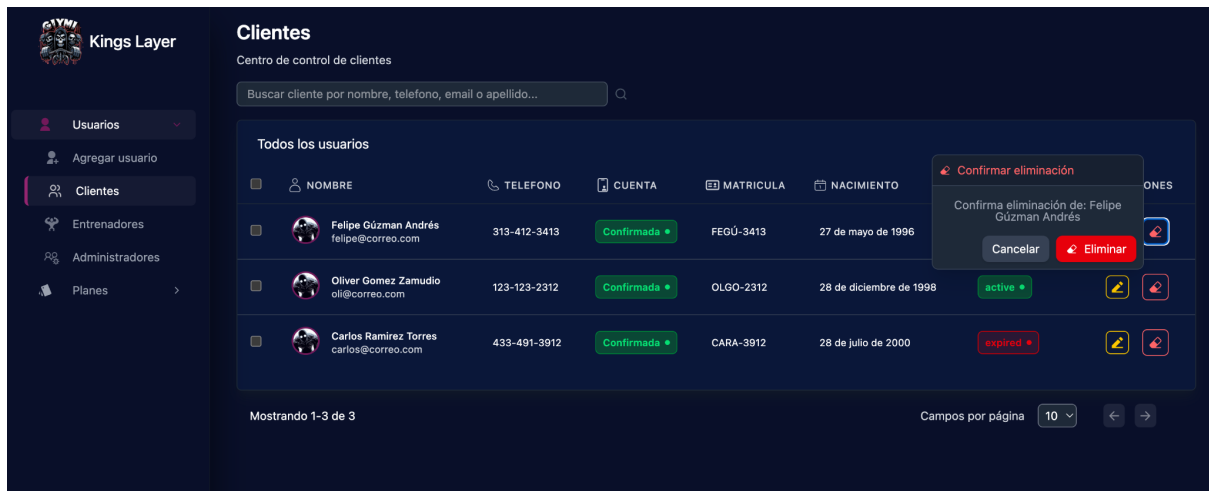


Imagen 23: Todos los clientes diseño

Suscripciones de cliente diseño (imagen 24)

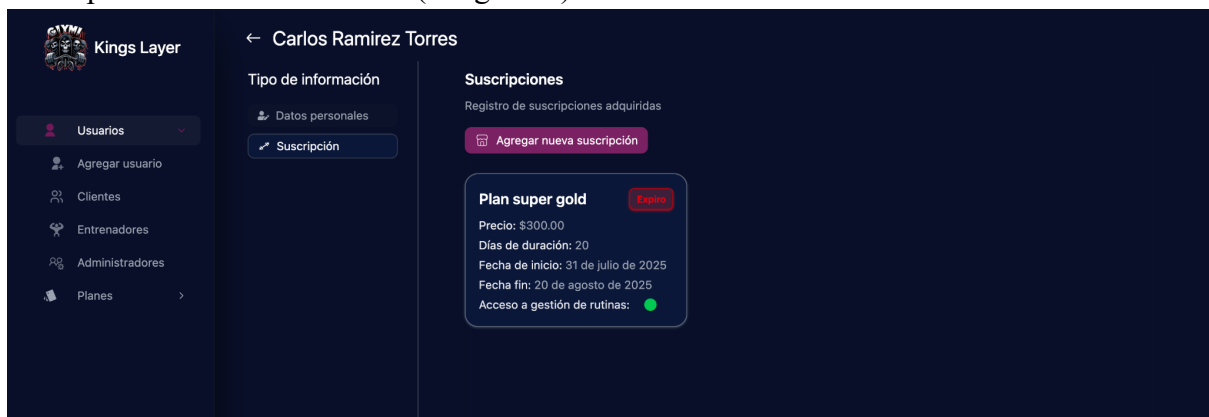


Imagen 24: Suscripciones clientes diseño

Agregar suscripción al cliente / comprar plan diseño (imagen 25)

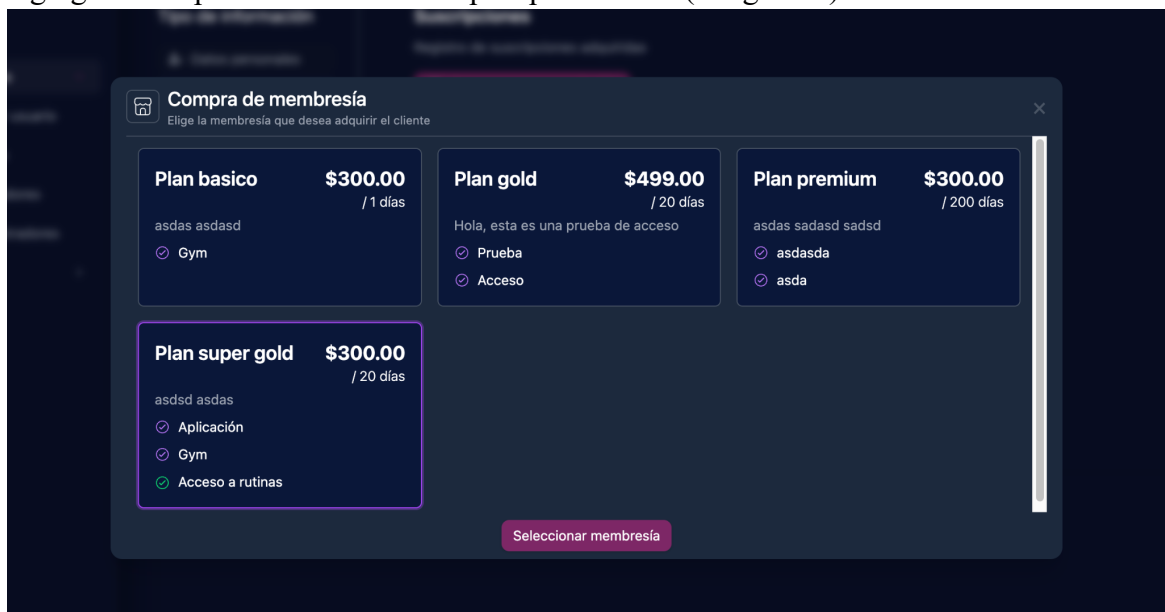


Imagen 25: Comprar plan diseño

Agregar plan diseño (imagen 26), se usa el mismo formulario para editar.


Kings Layer

Agregar un plan

Llena el formulario y envíalo para crear el plan

Usuarios

Planes

Agregar

Ver todas

Nombre

Beneficios

+ Agregar

Beneficio #1

-

Descripción

Plan de 15 días básico para acceso al gimnasio....

Precio

Duración días

Gestión de rutinas

☒

** Este campo es para seleccionar si el plan otorga acceso a la gestión de rutinas

Enviar formulario

Imagen 26: Agregar plan diseño

Todos los planes diseño (imagen 27)


Kings Layer

Planes

Centro de control de planes

Usuarios

Planes

Agregar

Ver todas

Todos los planes existentes

	NOMBRE	BENEFICIOS	DESCRIPCIÓN	APLICACIÓN	PRECIO	DURACIÓN	ACCIONES
1	Plan basico	Gym	asdas asdasd	NO	\$300.00	1 días	 
2	Plan gold	Prueba Acceso	Hola, esta es una prueba de acceso	NO	\$499.00	20 días	 
3	Plan premium	asdasda asda	asdas sadasd sadsd	NO	\$300.00	200 días	 
4	Plan super gold	Aplicación Gym	asdsd asdas	SI	\$300.00	20 días	 

Imagen 27: Todos los planes diseño

3.3.3 App Cliente

Diseño pantallas principales donde puede ver datos de su perfil con el historial de suscripciones (image 28) y el perfil de entrenamiento (imagen 29).

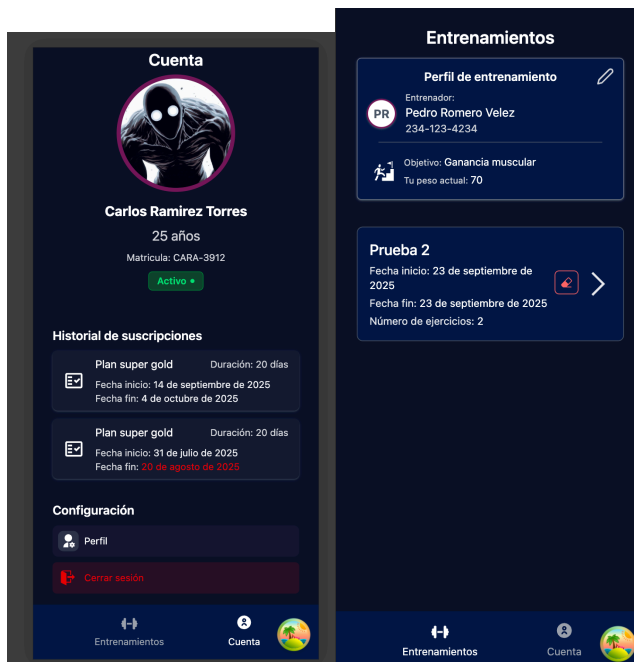


Imagen 28, 29: Cuenta cliente diseño, entrenamientos diseño

Diseño para el perfil de entrenamiento (imagen 30, 31, 32)

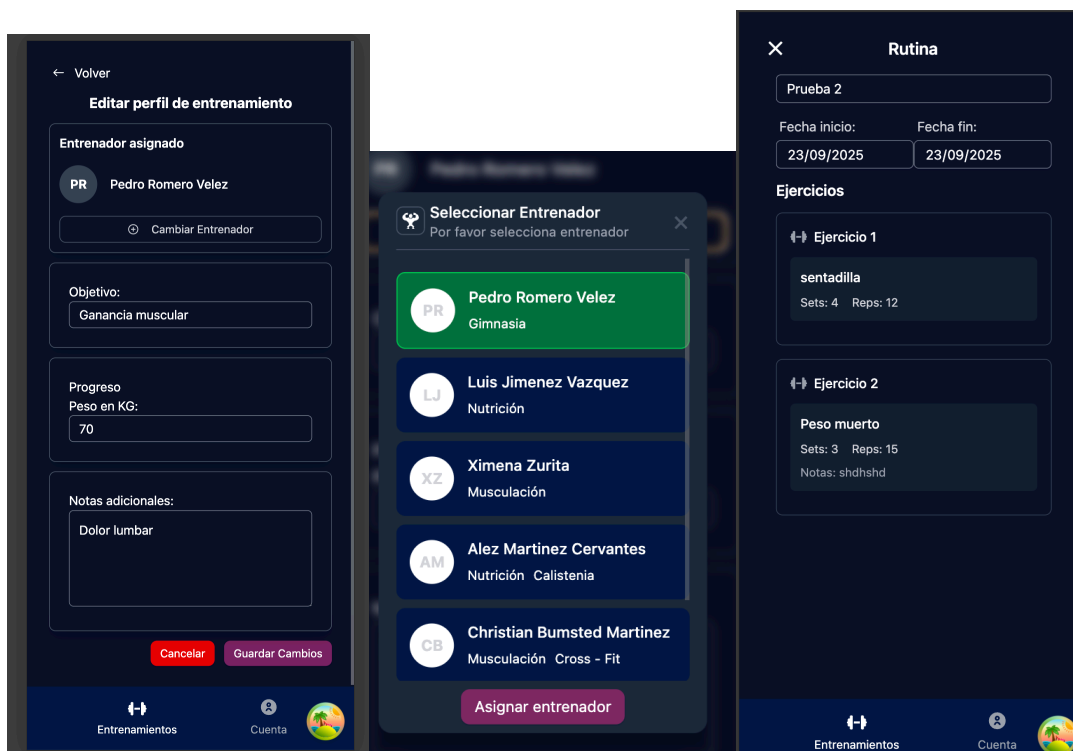


Imagen 30, 31, 32: Perfil entrenamiento, seleccionar entrenador, rutina diseño.

3.3.4 App Entrenador

Módulo de entrenador pantallas principales (imagen 33, 34)

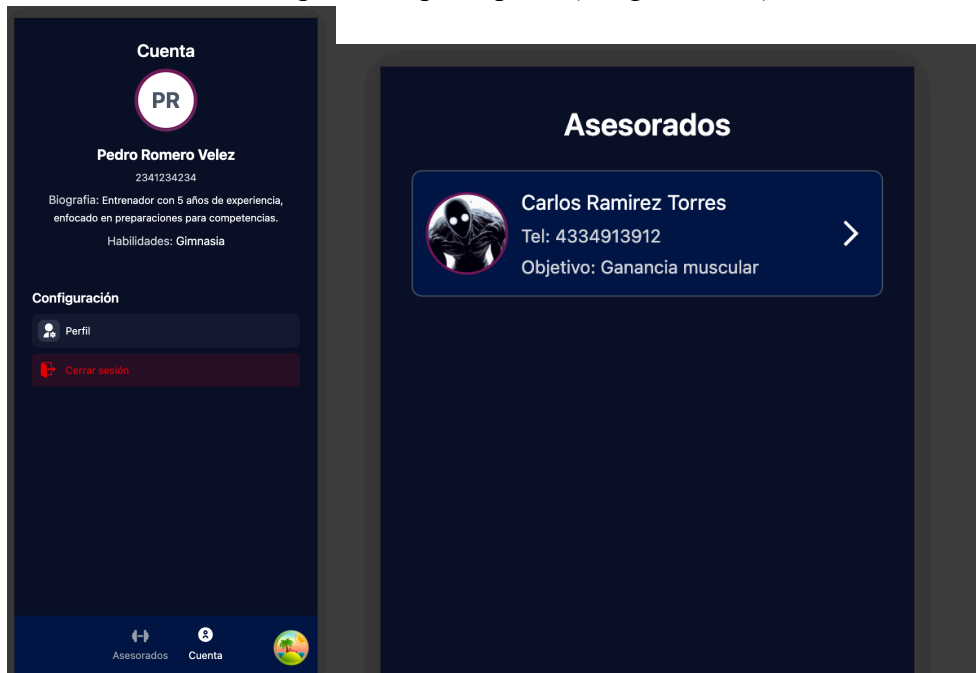


Imagen 33, 34: Perfil entrenador, asesorados entrenador diseño.

Diseño de pantallas para las rutinas (imagen 35, 36)

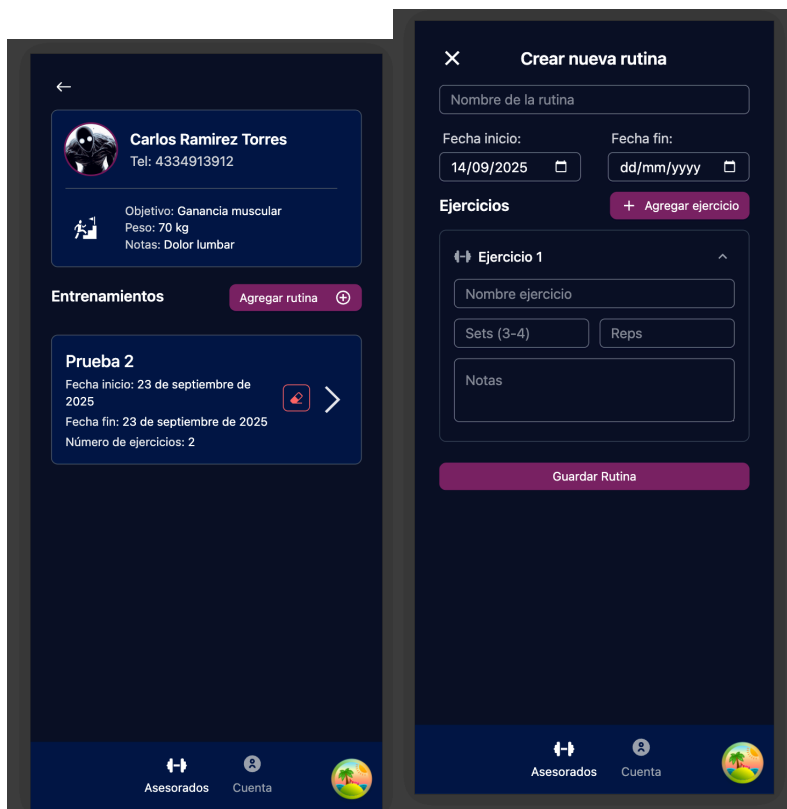


Imagen 35, 36: Rutinas asesorado, crear rutina asesorado diseño.

3.4 Diseño de la base de datos

3.4.1 Tablas

users		
PK	id	uuid
	name	varchar(60)
	last_name	varchar(60)
	email	varchar(30) UNIQUE
	phone	varchar(10) UNIQUE
	password	varchar(255)
	confirmed	boolean DEFAULT false
	token	varchar(6)
	login_attempts	integer DEFAULT 0
	deleted	boolean DEFAULT false
FK	role_id	integer
	created_at	timestamptz(6)
	updated_at	timestamptz(6)

Imagen 37: Usuarios diseño tabla

roles		
PK	id	serial
	name	enum_roles_name UNIQUE
	createdAt	timestamptz(6)
	updatedAt	timestamptz(6)

Imagen 38: Roles diseño tabla

admins		
PK	id	uuid
	access_level	enum_admins_access_level
FK	user_id	uuid UNIQUE

Imagen 39: Administrador diseño tabla

members		
PK	id	uuid
FK	user_id	uuid UNIQUE
	gender	enum_members_gender NOT NULL
	born_date	date NOT NULL
	matricula	varchar(255) UNIQUE

Imagen 40: Miembros / clientes diseño tabla

trainers		
PK	id	uuid
FK	user_id	uuid UNIQUE
	bio	text
	skills	varchar(255) []

Imagen 41: Entrenadores diseño tabla

plans		
PK	id	uuid
	name	varchar(60) UNIQUE
	description	varchar(70)
	benefits	varchar(30)[]
	price	double precision
	duration_days	integer
	application_access	boolean DEFAULT false
	is_active	boolean DEFAULT true
	deleted	boolean DEFAULT false
	createdAt	timestampz
	updatedAt	timestampz

Imagen 42: Planes diseño tabla

suscriptions		
PK	id	uuid
FK	member_id	uuid
FK	plan_id	uuid
	start_date	date
	end_date	date NOT NULL
	created_at	timestampz(6)
	updated_at	timestampz(6)

Imagen 43: Suscripciones diseño tabla

member_trainer		
PK	id	uuid
FK	member_id	uuid UNIQUE
FK	trainer_id	uuid
	notes	text
	goal	varchar(255)
	progress	json
	createdAt	timestampz(6)
	updatedAt	timestampz(6)

Imagen 44: Perfil de entrenamiento diseño tabla

workout		
PK	id	uuid
FK	relation_id	uuid
	name	varchar(20)
	start_date	date
	end_date	date NOT NULL
	exercises	json[]
	createdAt	timestampz(6)
	updatedAt	timestampz(6)

Imagen 45: Entrenamiento diseño tabla

3.4.1 Diagrama Entidad-Relación (ERD) (imagen 46)

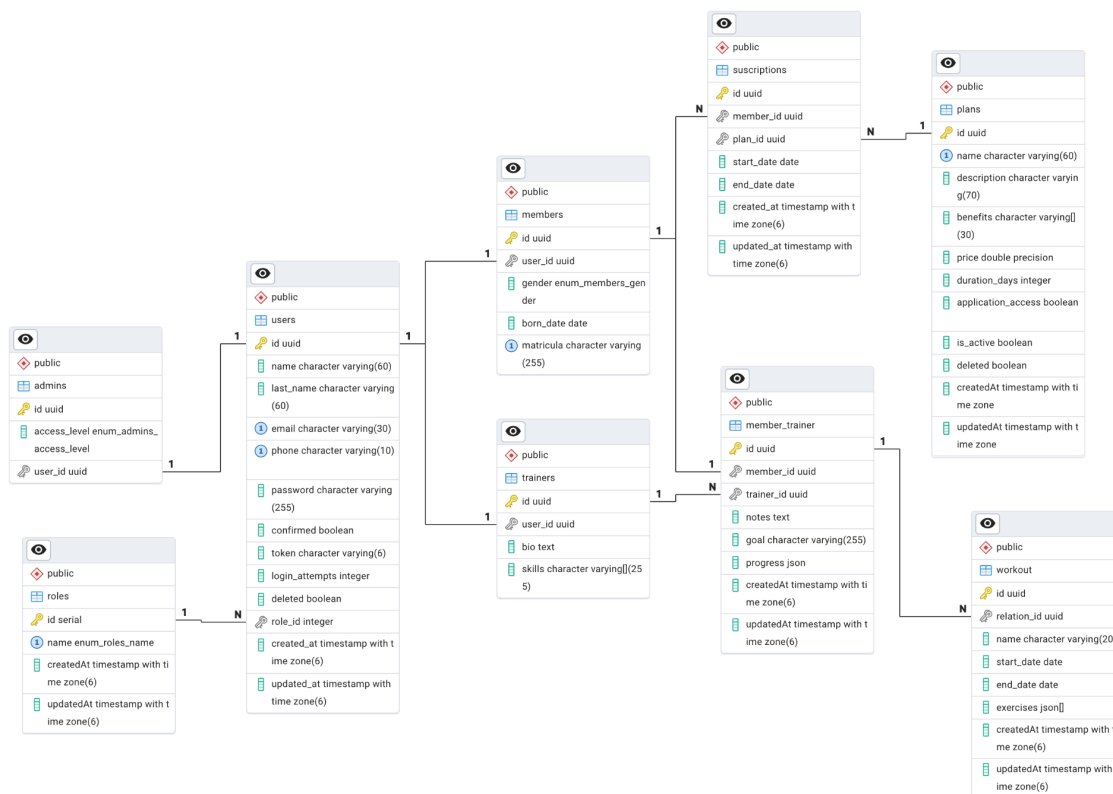


Imagen 46: ERD de la base de datos

3.4.2 Implementación con Sequelize

Descargar sequelize en el proyecto comando

`npm install --save sequelize`

Conexión con la base de datos

Para la conexión segura a la base de datos se divide el código entre las credenciales que se pueden ver en la imagen 47 y la función encargada de realizarla se visualiza en la imagen 48.

```

export const db = new Sequelize({
  database: process.env.DATABASE_NAME,
  user: process.env.DATABASE_USERNAME,
  password: process.env.DATABASE_PASSWORD,
  host: process.env.DATABASE_HOST,
  port: parseInt(process.env.DATABASE_PORT as string, 10),
  dialect: PostgresDialect,
  timezone: "America/Mexico_City",
  logging: false,
  models: [
    UserModel,
    RoleModel,
    AdminModel,
    MemberModel,
    TrainerModel,
    PlansModel,
    SubscriptionModel,
    AttendanceModel,
    MemberTrainerModel,
    WorkoutModel,
  ],
});
  
```

Imagen 47: Configuración con base de datos

```

*/
private static async connectDB(): Promise<void> {
  try {
    await db.authenticate();
    db.sync(); // Esto permite que si creamos modelos nuevos se creen en la
               // base de datos si los tenemos especificados en la configuración de la DB
    await runLoaders();
    console.log(colors.blue.bold("Conexión exitosa a la BD"));
  } catch (error) {
    console.log(colors.red.bold("Falló la conexión a la BD"));
  }
}

```

Imagen 48: Conexión de base de datos al sistema

Definición de un modelo

Un modelo representa una tabla de la base de datos, permitiendo así interactuar con la base de datos mediante la misma sintaxis de JS lo que facilita el manejo de operaciones dentro del backend como se visualiza en la imagen 49, al igual que las relaciones se crean como en la imagen 50.

```

backend > src > infrastructure > models > TS user.model.ts > UserModel
28 export class UserModel extends Model<
30   InferCreationAttributes<UserModel>
31 > {
32   @Attribute(DataTypes.UUID)
33   @Default(DataTypes.UUIDV4)
34   @PrimaryKey
35   declare id: CreationOptional<string>;
36
37   @NotNull
38   @Attribute(DataTypes.STRING(60))
39   declare name: string;
40
41   @NotNull
42   @Attribute(DataTypes.STRING(60))
43   declare last_name: string;
44

```

Imagen 49: Modelo de sequelize

```

// Relationships
@BelongsTo(() => RoleModel, "role_id")
declare role?: NonAttribute<RoleModel>;

// 1 ... 1 Foreign key
@HasOne(() => MemberModel, {
  foreignKey: "user_id",
  inverse: { as: "member_profile" }, // nombre explícito de la asociación
})
declare member_profile?: NonAttribute<MemberModel>;

// 1 ... 1 Foreign key
@HasOne(() => AdminModel, "user_id")
declare admin?: NonAttribute<AdminModel>;

```

Imagen 50: Relaciones del modelo sequelize

Sincronización y uso el modelo

Al tener creados los modelos se pueden usar funciones que están implementadas en sequelize para interactuar con la base de datos, solo nos encargamos de enviar los datos correctos como se muestra en la imagen 51.

```
async findByPhone(phone: string): Promise<User | false> {
  const userModel = await UserModel.findOne({
    where: { phone, deleted: false },
  });
  if (!userModel) return false;
  return new User(userModel.get());
}

async findByToken(token: string): Promise<User | false> {
  const userModel = await UserModel.findOne({
    where: { token, deleted: false },
  });
  if (!userModel) return false;
  return new User(userModel.get());
}
```

Imagen 51: Uso de un modelo

3.6 Planificación del desarrollo (planeación de sprints):

Para una mejor planificación se utilizó una plantilla de Notion la cual cumple con una estructura clara de Dashboard de proyectos ágiles con SCRUM.

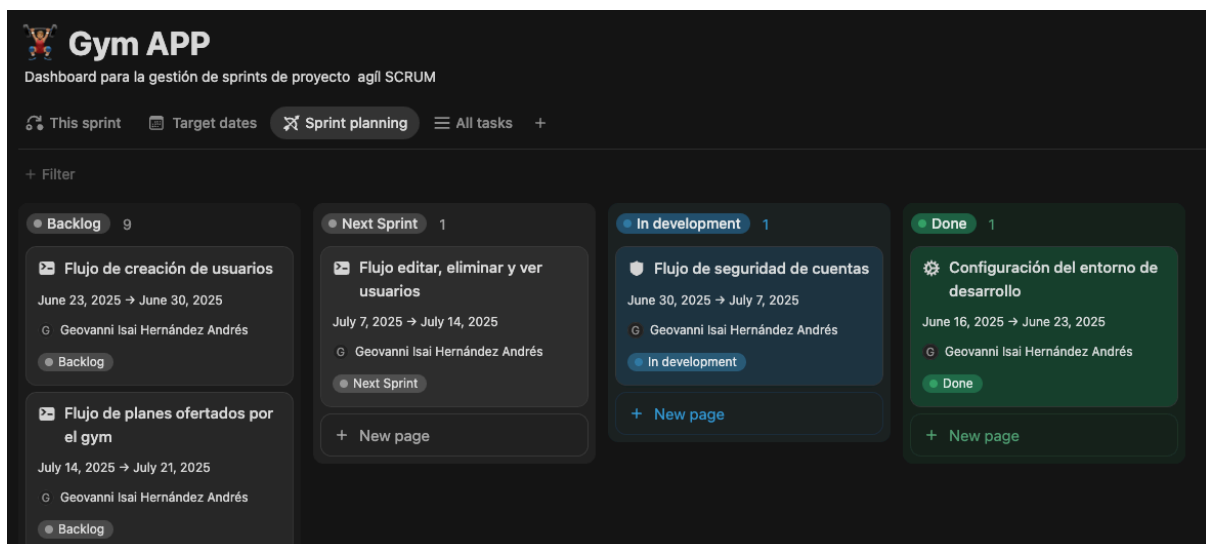


Imagen 52: Dashboard SCRUM

La planificación se dividió en 12 sprints los cuales tienen como mínimo una semana para ser entregados y máximo 2 en casos donde la lógica es extensa



Configuración del entorno de desarrollo

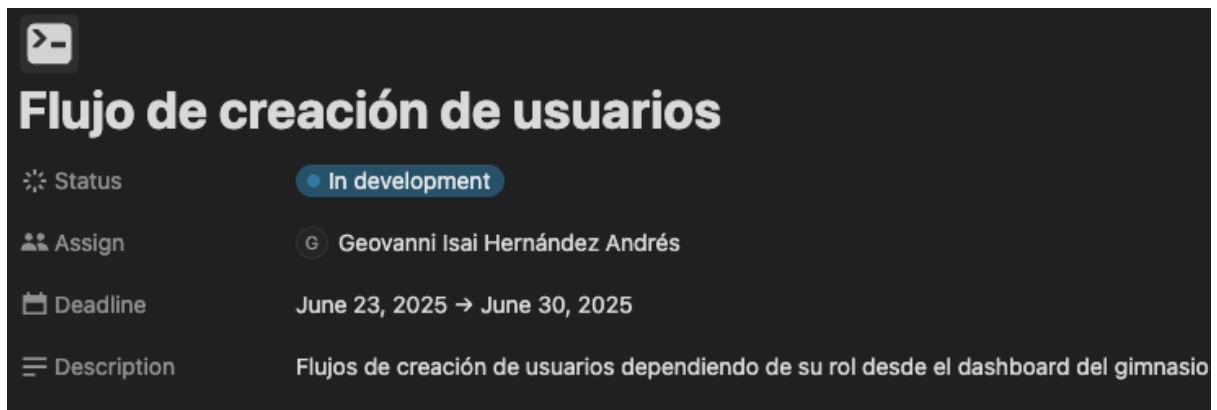
Status: Done

Assign: Geovanni Isai Hernández Andrés

Deadline: June 16, 2025 → June 23, 2025

Description: El objetivo de está tarea es iniciar con el entorno de desarrollo en Visual Studio Code con la estructura basica cliente-servidor con Node.js y React para dar pie al desarrollo del sistema con estructura clara de carpetas. Tambien se realizara la configuración inicial de los modelos de la base de datos con Sequelize.

Imagen 53: Sprint 1



Flujo de creación de usuarios

Status: In development

Assign: Geovanni Isai Hernández Andrés

Deadline: June 23, 2025 → June 30, 2025

Description: Flujos de creación de usuarios dependiendo de su rol desde el dashboard del gimnasio

Imagen 54: Sprint 2



Flujo de seguridad de cuentas

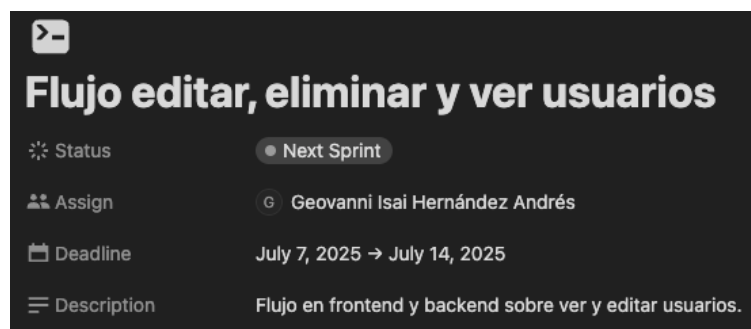
Status: Next Sprint

Assign: Geovanni Isai Hernández Andrés

Deadline: June 30, 2025 → July 7, 2025

Description: Gestión de flujo de seguridad con las cuentas como solicitar cambio de contraseña, cambio de contraseña, JWT para la autenticación.

Imagen 55: Sprint 3



Flujo editar, eliminar y ver usuarios

Status: Next Sprint

Assign: Geovanni Isai Hernández Andrés

Deadline: July 7, 2025 → July 14, 2025

Description: Flujo en frontend y backend sobre ver y editar usuarios.

Imagen 56: Sprint 4



Imagen 57: Sprint 5

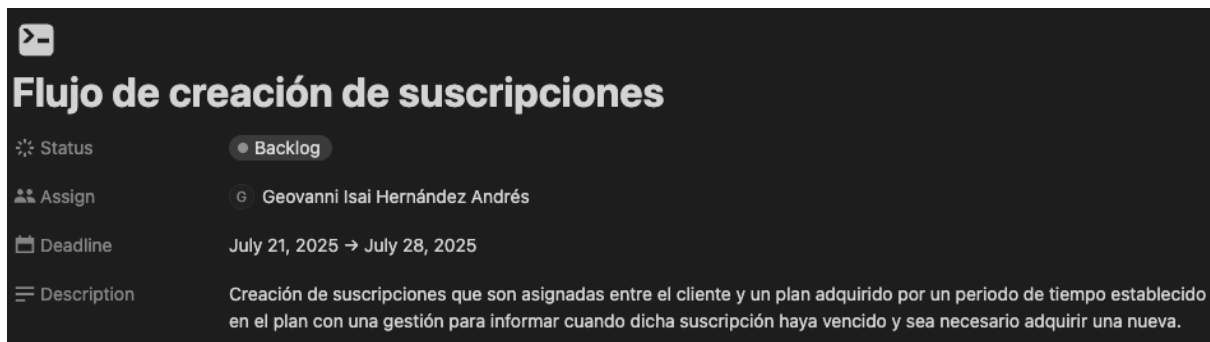


Imagen 58: Sprint 6

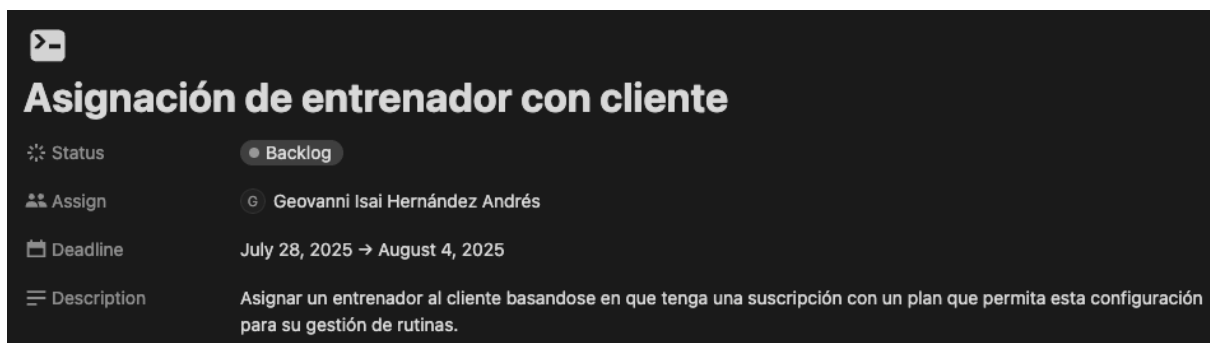


Imagen 59: Sprint 7

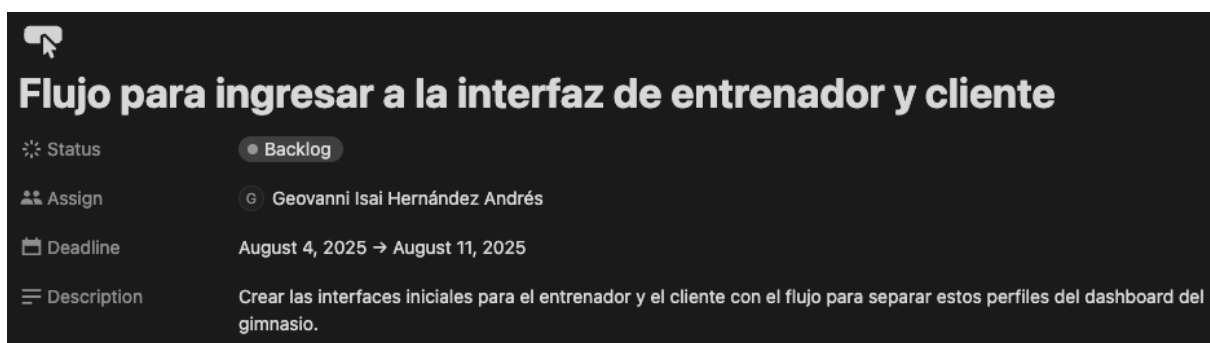


Imagen 60: Sprint 8

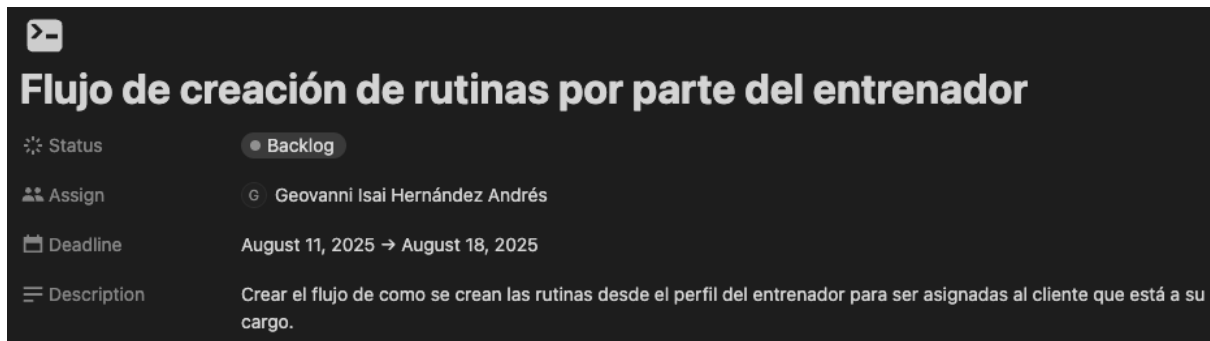


Imagen 61: Sprint 9



Imagen 62: Sprint 10

Capítulo 4: Construcción

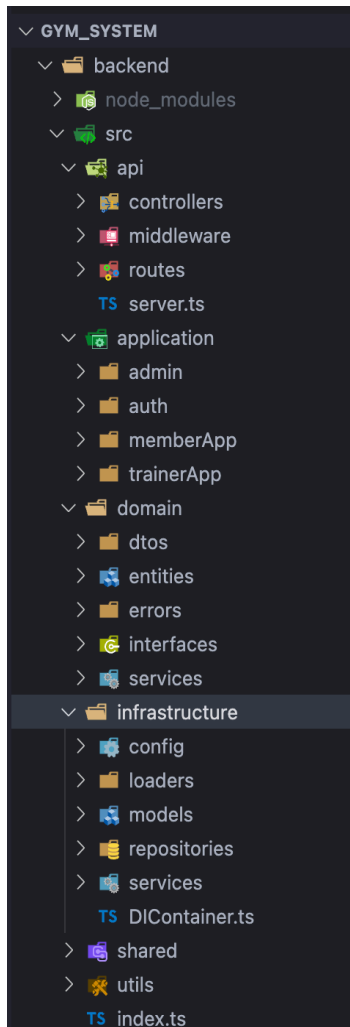
4.1 Estructura de Clean Architecture en el Backend

En esta imagen se muestra la estructura de nuestro servidor, esta manera de organizar el sistema se efectuó así para dividir cada módulo o componente del sistema con su responsabilidad

específica e independiente lo cual sigue el principio de separación de responsabilidades. Cada carpeta dentro de src es una capa del proyecto la cual se encarga de una sola capa de abstracción, dicha estructura se puede visualizar en la imagen 63.

Breve descripción de cada carpeta:

Image 63: clean architecture backend



Capa de presentación / API (src/api):

Puntos de entrada - controladores y rutas

- controllers/: Manejo de request/response HTTP.
- middleware/: Interceptores (auth, validaciones, rate limit).
- routes/: Define los endpoints REST.

Capa de aplicación (src/application):

Casos de uso - orquestación de flujos de negocio

- admin/: casos de uso específicos del administrador.
- auth/: lógica de autenticación y autorización.
- memberApp/: casos de uso para clientes.
- trainerApp/: casos de uso para entrenadores.

Capa de dominio (src/domain):

Corazón del negocio - entidades y reglas puras

- dtos/: objetos de transferencia de datos (intercambio entre capas).
- entinties/: entidades de negocio principales
- errors/: intercepción de errores del sistema
- interfaces/: Contratos o abstracciones de reglas de operaciones (repositorios y servicios)
- services/: Contratos o abstracciones de los servicios.

Capa de infraestructura (src/infrastructure):

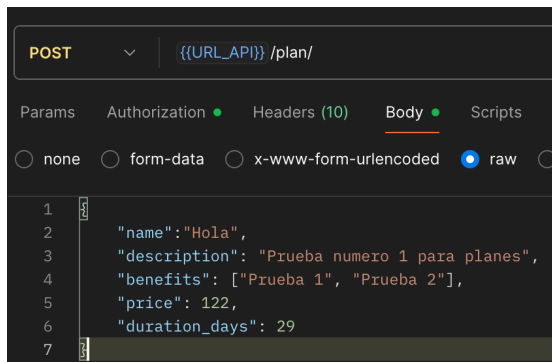
Conexión con framework, bases de datos y servicios externos

- config/: Configuración de la aplicación (base de datos, servicios externos).
- loaders/: Datos que se configuran en BD al iniciar el sistema.
- models/: Modelos de la base de datos con sequelize.
- repositories/: Implementación concreta de las consultas con la base de datos.
- services/: Servicios de infraestructura (email y storage).

4.2 Flujo de backend con clean architecture

1. Entrada - capa de API / presentación

Request llega de la petición en el frontend



Se envía una petición HTTP a una URL definida en routes, cada ruta se divide por la información que se maneja, por ejemplo, si se quiere crear un plan se envía una petición con un body que contenga los datos necesarios para dicho plan como se muestra en la imagen 64.

Imagen 64: HTTP REQUEST POSTMAN

Procesamiento en api/

La información del request es recibida por la ruta definida como en la imagen 65

src/api/routes/plan.routes.ts

```
planRouter.use(limiter, authenticate, authorize(["admin"]));
planRouter.param("planId", validatePlanId);
planRouter.param("planId", validatePlanExists);

// Ruta creación de plan
planRouter.post(
  "/",
  validatePlanInput,
  handleInputErrors,
  planController.create.bind(planController)
);
```

limiter, authenticate, authorize, validatePlanId, validPlanExists son middlewares que funcionan como intermediarios en la petición, estos nos dan control para validaciones antes de que ingrese la solicitud al controlador para manejar datos y errores de manera segura.

Imagen 65: Router Backend

src/api/controllers/plan.controller.ts

```

export class PlanController {
  constructor(
    private getPlansUseCase: GetAllPlansUseCase,
    private createPlanUseCase: CreatePlanUseCase,
    private updatePlanUseCase: UpdatePlanUseCase,
    private deletePlanUseCase: DeletePlanUseCase
  ) {}

  async create(req: Request, res: Response, next: NextFunction) {
    try {
      const {
        name,
        description,
        benefits,
        price,
        duration_days,
        application_access,
      } = req.body;
      const newPlan = await this.createPlanUseCase.execute({
        name,
        description,
        benefits,
        price,
        duration_days,
        application_access,
      });
      res.json({
        success: true,
        message: "Plan creando correctamente",
        newPlan,
      });
    } catch (error) {
      next(error);
    }
  }
}

```

Dentro de cada controlador se manejan los datos que se reciben en la solicitud para posteriormente enviarse al caso de uso específico el cual manejara toda la operación y retorna una respuesta la cual se almacena al controlador para que este envíe la respuesta de manera correcta a donde se envió la solicitud HTTP. La estructura del controlador se visualiza en la imagen 66.

Imagen 66: Controlador backend

2. Capa de aplicación - Orquestación

src/api/application/admin/plan/create.usecase.ts

En los casos de uso se representa y ejecuta la lógica de cada caso de uso definido en los requerimientos que se hicieron en la planeación del sistema. Un ejemplo del código es el de la imagen 67

```

export class CreatePlanUseCase {
  constructor(private planRepository: PlanRepository) {}

  async execute(input: PlanProps): Promise<void> {
    // 1. Verificamos que no exista un plan con el mismo nombre
    const planExists = await this.planRepository.findByName(input.name);
    if (planExists) {
      throw new ConflictError("Ya existe plan con ese nombre");
    }

    // Creamos el objeto de plan

    const plan = new Plan(input);
    await this.planRepository.create(plan);
  }
}

```

Para manejar las operaciones de manera segura y modular se hace uso de las entidades, repositorios y servicios que al trabajar en manera conjunta permiten que toda la interacción entre la base de datos y la lógica del sistema sea más concisa siendo orquestada desde un mismo lugar.

Imagen 67: Caso de uso backend

3. Capa de dominio - Lógica del negocio

src/domain/entities/plan.entity.ts

```
export class Plan {
  constructor(private readonly props: PlanProps) {}

  get id(): string | undefined {
    return this.props.id;
  }
  > get name(): string {--
  }

  softDelete(): void {
    this.props.is_active = false;
    this.props.deleted = true;
  }

  restore(): void {
    this.props.is_active = true;
    this.props.deleted = false;
  }
}
```

Las entidades son las encargadas de almacenar la lógica del negocio, cada entidad del negocio se representa en una clase que contiene los atributos y maneja la lógica mediante métodos que son accionados desde un caso de uso, esto permite que cada entidad tenga una sola responsabilidad sobre las operaciones, ayudando a que el código sea fácil de entender y mantener. Un ejemplo del código es el de la imagen 68

Imagen 68: Entidad backend

src/domain/errors/conflict.error.ts

```
export class ConflictError extends AppError {
  statusCode = 409;

  constructor(public message: string) {
    super(message);
  }

  serializeErrors() {
    return [{ message: this.message }];
  }
}
```

El manejo de cada tipo de error se almacena por separado, con esto pueden ser customizadas las respuestas para ser usadas en diferentes lugares. Un ejemplo del código es la imagen 69.

Imagen 69: Error manejador backend

4. Capa de infraestructura - Implementaciones concretas

src/infrastructure/repositories/plan.repository.ts

```
export class SequelizePlanRepository implements PlanRepository {
  async getAll(): Promise<PlanProps[]> {
    const plans = await PlansModel.findAll({ where: { deleted: false } });
    return plans;
  }

  async create(plan: Plan): Promise<Plan> {
    const planNew = await PlansModel.create({
      name: plan.name,
      description: plan.description,
      benefits: plan.benefits,
      price: plan.price,
      duration_days: plan.duration_days,
      application_access: plan.application_access,
    });
    return new Plan(planNew);
  }
}
```

Los repositorios manejan las operaciones con la base de datos que se creó con sequelize, dentro de cada repositorio interactúan los datos que recibe de un caso de uso y lo dirige a la base de datos, manteniendo en un solo lugar la interacción. Al mismo tiempo separa cada función con la BD en un método específico. Un ejemplo es la imagen 70.

Imagen 70: Repositorios base de datos backend

5. Respuesta al cliente

Cuando el sistema termina el flujo de la petición recibe una respuesta similar a la de la imagen 71 indicando si fue correcta la respuesta.

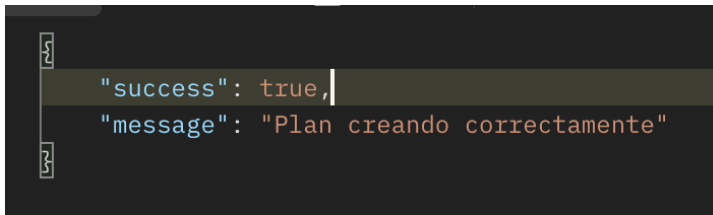
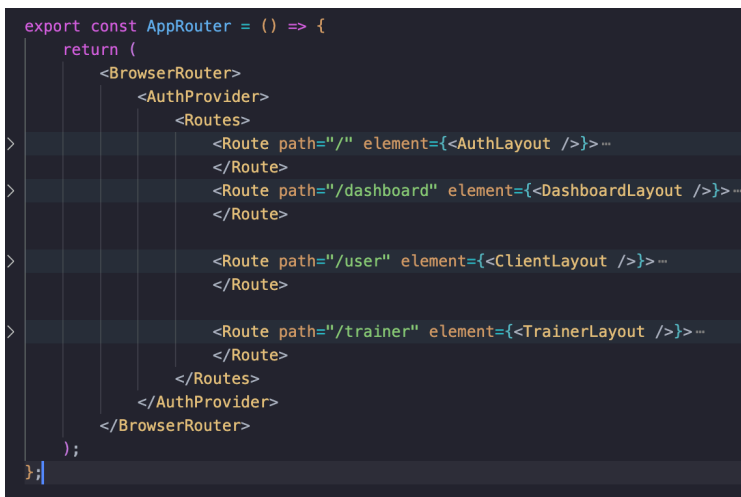


Imagen 71: Respuesta POSTMAN

4.3 Uso de react router



El uso de react router es indispensable en este sistema ya que permite el enrutamiento estándar de una aplicación React creando una single page application con navegaciones entre diferentes vistas sin tener que recargar la aplicación liberando así tiempo de espera al interactuar y mejorando la experiencia de usuario. Un ejemplo de la estructura es el de la imagen 71.

Imagen 71: React router

4.4 Creación de componentes en React.js

React permite que escribir páginas completas sea más sencillo siguiendo su enfoque de construir dichas páginas mediante componentes que se ensamblan como piezas de LEGO. Al crear un componente se tiene que tomar en cuenta distintos aspectos como; ser independiente, reutilizable y se debe especializar en una tarea específica con la finalidad de hacer el código más organizado, mantenible y escalable.

Aquí hay un ejemplo en la imagen 72 de un mensaje de error que se puede utilizar en diferentes componentes para mostrar los mensajes de error sin necesidad de que se repita el mismo código como en la imagen 73.

```

interface ErrorMessageProps {
  message: string;
  error?: Error;
  className?: string;
}

export const ErrorMessage = ({
  message,
  error,
  className,
}: ErrorMessageProps) => {
  const errorMessage = handleApiError(error);
  return (
    <div className={clsx("rounded-md bg-red-50 p-4", className)}>
      <div className="flex">
        <div className="ml-3">
          <h3 className="text-sm font-medium text-red-800">
            {message}
          </h3>
          {error && (
            <div className="mt-2 text-sm text-red-700">
              <p>{errorMessage}</p>
            </div>
          )}
        </div>
      </div>
    </div>
  );
};

```

Imagen 72: Componente react

```

if (isErrorTrainers) {
  return (
    <ErrorMessage
      message="Error al cargar los datos"
      error={errorTrainers!}
      className="my-4"
    />
  );
}

```

Imagen 73: Componente react

4.5 Consumir backend desde frontend

Al necesitar que el sistema almacene misma información en diferentes vistas y que las actualizaciones de los datos sean optimizadas constantemente, se tomó como opción utilizar TanStack Query para el consumo de las API's, dicha librería es muy común hoy en día en proyectos React, ya que otorga mejoras en las peticiones a diferencia de usar solo Fetch o Axios. Algunas de las mejoras más importantes y por las que se eligió dicha herramienta para el desarrollo son; caché inteligente automático, query keys para identificar las peticiones, estados integrados de consulta (isLoading, isSuccess, isFetching, etc.), re-fetch automático, sincronización en tiempo real, evita llamadas duplicadas. Un ejemplo de las funciones necesarias y el flujo de la petición está en la imagen 74.

```

frontend > src > lib > config > TS axios.ts > api.interceptors.request.use() callback
3   const api = axios.create({
4     |   baseURL: import.meta.env.VITE_API_URL,
5   });
6
7   api.interceptors.request.use(async (config) => {
8     |   try {
9       |     const token = localStorage.getItem("AUTH_TOKEN");
10      |     if (token) {
11        |       config.headers.Authorization = `Bearer ${token}`;
12      |     }
13      |     return config;
14    } catch (error) {
15      |     console.error("Error al obtener el token:", error);
16      |     return Promise.reject(error);
17    }
18  });
19
20  export default api;

```

S TrainerApi.ts ×

```

frontend > src > api > TS TrainerApi.ts > getTrainers
4   export async function getTrainers() {
5     |   const { data } = await api.get("/trainer/");
6     |   return data || [];
7   }
8

```

SelectTrainer.tsx ×

```

frontend > src > components > organisms > clientTrainerApp > SelectTrainer.tsx > [⌵] Props
21  export const SelectTrainer = ({ selectedTrainer, onSelectTrainer }: Props) => {
22    |   // Use Query para obtener los entrenadores
23    |   const {
24      |     data: dataTrainers,
25      |     isLoading: isLoadingTrainers,
26      |     isError: isErrorTrainers,
27      |     error: errorTrainers,
28    } = useQuery({
29      |     queryFn: (): Promise<TrainerProps[]> => getTrainers(),
30      |     queryKey: ["trainers"],
31    });

```

Imagen 74: Consulta a API

Capítulo 5: Pruebas y liberación

Para garantizar la calidad y estabilidad de la aplicación web, se implementaron distintos entornos de despliegue y etapas de prueba. Cada entorno cumple una función específica en el ciclo de vida de la aplicación.

Entorno de desarrollo (DEV)

Al ser el entorno donde como desarrollador constantemente se están implementando nuevas funcionalidades y corrigen errores se necesitan pruebas que verifiquen el funcionamiento de componentes o funciones individuales, para ello se hizo uso de pruebas unitarias y de integración. Estas pruebas fueron realizadas con Jest qué es un framework de pruebas para JavaScript.

Pruebas unitarias

Comprueban que el código funciona según lo previsto mediante la creación y ejecución de pruebas individuales para cada función como se muestra en la imagen 75. Se catalogan como pruebas unitarias porque desglosa la funcionalidad del programa en comportamientos probables discretos que se pueden probar como unidades individuales, además permiten que la aplicación sea más robusta, pues si se añaden cambios al código de la aplicación, los test aseguran que no se rompa alguna funcionalidad o dé resultados no esperados como en la imagen 76.

```
backend > test > TS test.ts > describe('AuthServiceImpl') callback
6 describe('AuthServiceImpl', () => {
7   let authService: AuthServiceImpl;
8
9   beforeEach(() => {
10    authService = new AuthServiceImpl();
11    jest.clearAllMocks();
12  });
13
14  describe('generateJWT', () => {
15    it('should generate a valid JWT token when JWT_SECRET is defined', () => {
16      process.env.JWT_SECRET = 'test-secret-key';
17      const userId = '12345';
18      const mockToken = 'mock-jwt-token';
19
20      jest.spyOn(jwt, 'sign').mockReturnValue(mockToken as any);
21
22      const result = authService.generateJWT(userId);
23
24      expect(jwt.sign).toHaveBeenCalledWith(
25        { id: userId },
26        'test-secret-key',
27        { expiresIn: '1d' }
28      );
29      expect(result).toBe(mockToken);
30    });
31  });
32 });
```

Imagen 75: Test unitario

```
PASS src/__tests__/auth.service.impl.test.ts
AuthServiceImpl
generateJWT
  ✓ should generate a valid JWT token when JWT_SECRET is defined (15 ms)
  ✓ should throw an error when JWT_SECRET is not defined (8 ms)
  ✓ should throw an error when JWT_SECRET is empty string (4 ms)
  ✓ should use correct expiration time (3 ms)

Test Suites:
1 passed, 1 total
Snapshots:
0 total

Tests:
4 passed, 4 total
Time:
1.234 s

Ran all test suites matching /auth.service.impl.test.ts/i.
```

Imagen 76: Test resultado

Pruebas de integración

Las pruebas de integración garantizan que diferentes módulos y componentes de una aplicación funcionen correctamente, simula dependencias y servicios externos, abarca pruebas que tienen interacción con la base de datos.

Un ejemplo es un controller, que al entrar a una función del controller se disparan consultas a la base de datos para que se ejecuten correctamente los flujos dentro de la aplicación. El test del controller se muestra en la imagen 77 y el resultado correcto en la imagen 78.

```
6
7 describe('MemberController Integration Tests', () => {
8   let app: Express;
9   let database: Database;
10
11   beforeAll(async () => {
12     app = await createApp();
13     database = new Database();
14     await database.connect();
15   });
16
17   afterAll(async () => {
18     await database.disconnect();
19   });
20
21   beforeEach(async () => {
22     // Limpiar datos de prueba antes de cada test
23     await database.clearTestData();
24   });
25
26   describe('GET /api/members', () => {
27     it('should return paginated members with real database', async () => {
28       // Arrange - Insertar datos reales en BD
29       await database.query(
30         'INSERT INTO members (name, email) VALUES (?, ?), (?, ?)',
31         ['John Doe', 'john@test.com', 'Jane Smith', 'jane@test.com']
32       );
33
34       // Act - Llamada HTTP real
35       const response = await request(app)
36         .get('/api/members?page=1&limit=10')
37         .expect(200);
38
39       // Assert - Verificar respuesta real
40       expect(response.body.success).toBe(true);
41       expect(response.body.members).toHaveLength(2);
42       expect(response.body.pagination.total).toBe(2);
43       expect(response.body.members[0].name).toBe('John Doe');
44     });
45   });
46 }
```

Imagen 77: Test integración

```
PASS  __tests__/integration/member.controller.integration.test.ts
MemberController Integration Tests
GET /api/members
✓ should return paginated members with real database (245 ms)
✓ should search members by name (156 ms)

Test Suites:
1 passed, 1 total
Snapshots:
0 total

Tests:
2 passed, 2 total
Time:
2.456 s

Ran all test suites matching /member.controller.integration.test.ts/i.
```

Imagen 78: Test resultado 2

Entorno de calidad (QA)

Las pruebas en este entorno tienen como objetivo verificar que la aplicación web cumpla con los requisitos funcionales, de seguridad y de rendimiento antes de su despliegue a producción. Para estas pruebas se desplegó la aplicación en un entorno limitado de pruebas en donde se realizaron consultas automatizadas con postman y postman cloud a las API's para verificar la fiabilidad de los datos como en la imagen 79, rendimiento del sistema y comprobar que cumple con las limitaciones de consultas que se configuraron al inicio del sistema en el limit rater como en la imagen 80, al igual que solo sea accesible el uso de las funciones por usuarios registrados y autorizados.

Para el lado del frontend se hicieron pruebas en distintos navegadores para comprobar que se permita el uso de la aplicación web en diferentes dispositivos accediendo desde navegadores como Chrome, Safari, Opera, Firefox y Edge. Estas medidas de prueba nos dan la certeza de que el sistema funciona como lo planeado y que esté optimizado para soportar el acceso desde distintos lugares.

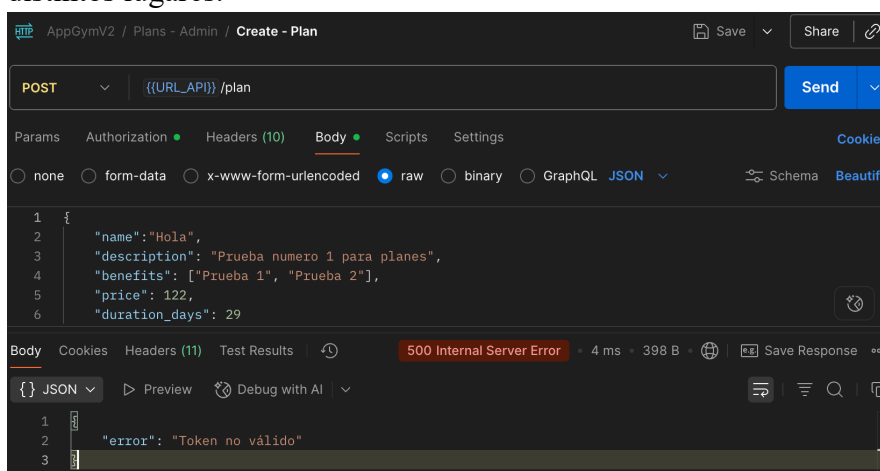


Imagen 79: Postman respuesta no autorizado

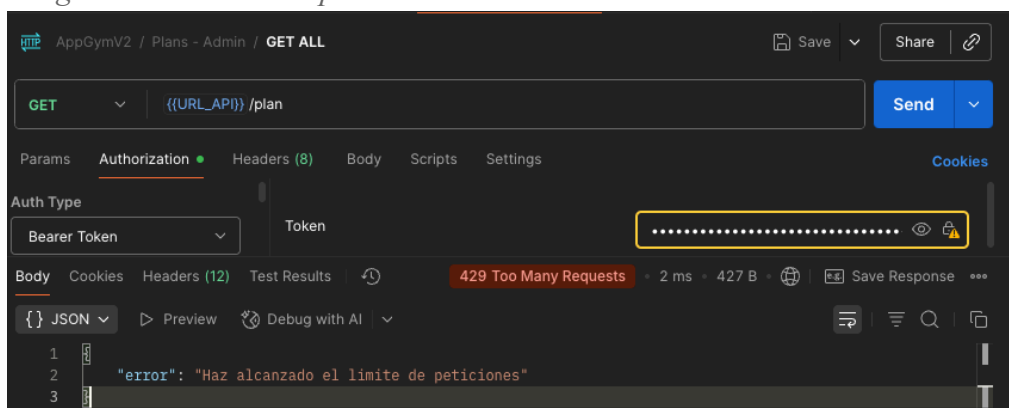


Imagen 80: Postman too many requests

Entorno de producción (PRD)

Para el despliegue de la aplicación y que nos permita hacer pruebas en un entorno de producción al que acceden los usuarios reales se hizo uso de Hostinger, un proveedor de hosting web que ofrece servicios para alojar webs completas, permite tener un dominio especial para

la aplicación el cual si se desea volver un dominio profesional se puede elegir entre distintos paquetes que ofrece dependiendo del tamaño del sistema, por lo cual se paga por lo que se requiere, evitando costes altos.

Conclusiones y trabajos futuros

Conclusiones

El desarrollo de esta aplicación web permite encontrar una alternativa a cómo se gestiona la relación entre los usuarios de un gimnasio y su personal, materializando una solución tecnológica escalable que responde correctamente ante la problemática central que es la falta de registros que otorguen al personal la posibilidad de crear planes de entrenamiento acorde a las necesidades específicas de cada persona. A través de la implementación de una arquitectura cliente-servidor y el uso de un stack tecnológico moderno conformado por React y Typescript se permite que la escalabilidad al sistema sea posible sin enfrentarse a cambios abrumadores.

La aplicación permite a los clientes tener un registro y seguimiento de sus avances dentro de su tiempo dentro del gimnasio, con los registros de su fisiología el entrenador puede ofrecer mejores servicios al conocer más lo que necesita cada cliente sin perder información con el pasar del tiempo. Esto ayuda a que las personas se centren en mejorar mes con mes y no abandonar el estilo de vida que están buscando alcanzar.

En un punto de vista técnico, la implementación de clean architecture en backend fue un pilar fundamental para el éxito del prototipo. La arquitectura permitió que el código sea mantenible, testeable y escalable al tener una clara separación de responsabilidades dentro de cada función.

Trabajos a futuro

En definitiva, este proyecto puede escalar a mucho más, por ello como trabajos futuros se planea integrar más servicios, tomando en cuenta todo lo que pueden encontrar dentro de un gimnasio. Por ejemplo, la gestión de planes alimenticios de igual manera forma un papel fundamental para esta disciplina de realizar actividad física constantemente, por ello se propone integrar un perfil de nutricionista el cual pueda otorgar estos servicios e ir de la mano con las rutinas para que se complementen. En el ámbito técnico se considera la implementación de triggers dentro de la base de datos para mejorar la fluidez del servidor y de actualizaciones de estados como las suscripciones se logren automatizar mediante estas herramientas que nos ofrece PostgreSQL.

Referencias

1. World Health Organization: WHO. (2024, 26 junio). Actividad física. <https://www.who.int/es/news-room/fact-sheets/detail/physical-activity>
2. Scarpetta, R. Á. G., Arismendy, L. D., Sosa, L. J. C., Vargas, C., & Becerra, N. R. R. (2012). Revista Hacia la Promoción de la Salud. 17, 13–24. <https://www.redalyc.org/pdf/3091/309126695013.pdf>
3. Techfit, & Techfit. (2023, 2 mayo). La evolución del software de gimnasio: tendencias y avances tecnológicos. Techfit Blog. <https://techfit.app/blog/la-evolucion-del-software-de-gimnasio-tendencias-y-avances-tecnologicos/>
4. Baena-Arroyo, M. J., García-Fernández, J., Gálvez-Ruiz, P., & Grimaldi-Puyana, M. (2020). Analyzing Consumer Loyalty through Service Experience and Service Convenience: Differences between Instructor Fitness Classes and Virtual Fitness Classes. Sustainability, 12(3), 828. <https://doi.org/10.3390/su12030828>
5. Urbina, P. (2024, 30 diciembre). Los mejores softwares para gimnasios en 2025 en México. TrainingGym. <https://blog.traininggym.com/es-mx/blog/mejores-softwares-gimnasios>
6. Stonerock, G. L., & Blumenthal, J. A. (2016). Role of Counseling to Promote Adherence in Healthy Lifestyle Medicine: Strategies to Improve Exercise Adherence and Enhance Physical Activity. Progress In Cardiovascular Diseases, 59(5), 455-462. <https://doi.org/10.1016/j.pcad.2016.09.003>
7. ¿Qué es la metodología Scrum? (2025, April 18). ENAE Business School. https://www.enaes.es/blog/metodologia-scrum?gad_source=1&gad_campaignid=19523271597&gbraid=0AAAAACffcHKL8UxBqDibGSL1sYWZSexCV&gclid=CjwKCAjw3MXBBhAzEiwA0vLXQbjzqF6Y-keiBdmtc3MsbMnBXzZnT6kcRCBLYUu74_WgMPyPZfhWDRoCrVkJQAvD_BwE&_adin=11551547647
8. What is Scrum? (s. f.). Scrum.org. <https://www.scrum.org/resources/what-scrum-module>
9. *Welcome to Glofox: The #1 fitness management software.* (2025, 13 mayo). Boutique Fitness And Gym Management Software - Glofox. <https://www.glofox.com/>
10. Best Fitness & Wellness Management Software. (2025, 19 mayo). Mindbody. <https://www.mindbodyonline.com/>

11. ClubOS. (s. f.). ClubOS: Gym Management Software. ClubOS. <https://www.club-os.com/>
12. Virtuagym. (s. f.). Virtuagym | El software de fitness No 1 para empresas y consumidores. <https://virtuagym.com/es/>
13. Oracle. (2021, 18 junio). *What is a Relational Database (RDBMS)?* <https://www.oracle.com/mx/database/what-is-a-relational-database/>
14. https://rapidapi.com/justin-WFnsXH_t6/api/exercisedb
15. ¿Qué es JSON Web Token? | Alura Cursos Online. (s. f.). Alura. <https://www.aluracursos.com/blog/que-es-json-web-token>
16. Expósito, J. (2023, 18 julio). *Qué es el rate limit y por qué debes limitar el tráfico a tu site*. Transparent Edge. <https://www.transparentedge.eu/blog/que-es-el-rate-limit-y-por-que-debes-limitar-el-trafico-a-tu-site/>
17. ¿Qué es JavaScript? - Explicación de JavaScript (JS) - AWS. (s. f.). Amazon Web Services, Inc. <https://aws.amazon.com/es/what-is/javascript/>
18. *Node.js — Introduction to TypeScript*. (s. f.). <https://nodejs.org/es/learn/typescript/introduction>
19. *Node.js — Introduction to Node.js*. (s. f.). <https://nodejs.org/es/learn/getting-started/introduction-to-nodejs>
20. *PostgreSQL 17.5 documentation*. (2025, 8 mayo). PostgreSQL Documentation. <https://www.postgresql.org/docs/current/index.html>
21. C, D. (2025, 20 febrero). ORM en acción: Conectando Node.js con PostgreSQL mediante Sequelize. *Medium*. <https://medium.com/@diego.coder/conexi%C3%B3n-a-una-base-de-datos-postgresql-con-node-js-y-sequelize-d93b0546e4cc>
22. *Hooks integrados en React – React*. (s. f.). <https://es.react.dev/reference/react/hooks>
23. pmoinformatica.com. (2018, 30 mayo). ¿Qué es un requerimiento funcional? <https://www.pmoinformatica.com/2018/05/que-es-requerimiento-funcional.html>
24. Staff, C. (2024, 3 octubre). What is Client-Server Architecture? Coursera. <https://www.coursera.org/articles/client-server-architecture>
25. Martin, Robert C et al. *Clean Architecture : A Craftsman’s Guide to Software Structure and Design*. 1st ed. Boston: Prentice Hall, 2018. Print.
26. Communications. (2025, 15 septiembre). Arquitectura de «software» o sistemas: qué es y ejemplos. BBVA NOTICIAS. <https://www.bbva.com/es/innovacion/arquitectura-de-software-o-sistemas-que-es-y-ejemplos/>