UNIVERSIDAD INTERNACIONAL SAN ISIDRO LABRADOR

SEDE SAN CARLOS

PROYECTO

FitTime

CARRERA

Ingeniería en Sistemas (ISB-32)

CURSO

Programación Avanzada

PROFESOR

Estefanía Boza

Villalobos

ELABORADO POR

José David Ramírez Salas

2025

Capítulo 1: Introducción

Objetivo del proyecto: El proyecto FitTime surge a partir de la observación en un gimnasio local, donde se identificó la dificultad para llevar un control adecuado de los cupos y horarios de las clases. Ante esta situación, se propuso desarrollar una herramienta digital que permita a los usuarios reservar sus entrenamientos de forma ordenada y segura, adaptándose a sus necesidades horarias y evitando problemas de sobrecupo o confusión en la programación.

Contexto y justificación: En muchos gimnasios locales, la gestión de horarios y cupos se realiza de forma manual, lo que puede generar errores, confusión o sobrecupo en las clases. El sistema FitTime busca resolver esta problemática mediante una plataforma web que facilite la organización y control de las reservas. De esta forma, los usuarios pueden conocer la disponibilidad en tiempo real y elegir los horarios que mejor se adapten a su rutina, mientras que los administradores logran un manejo más eficiente del flujo de clientes y la planificación diaria.
Metodología:

El tipo de metodología usada es híbrida, ya que se tenía un plan estructurado, pero con la oportunidad realizar cambios y así permitir más flexibilidad en el proyecto.

Estructura del trabajo:

Capítulo 2: Análisis y diseño

1. Conceptos Fundamentales

Un sistema de información es un conjunto organizado de elementos que permiten

recolectar, procesar, almacenar y distribuir información para apoyar la toma de decisiones, coordinar acciones y controlar proceso

Se desarrolló una aplicación web que permite a los usuarios reservar clases de forma autónoma y visualizar horarios disponibles. Esta aplicación integra funcionalidades como autenticación de usuarios, control de cupos y gestión de clases.

1. Entrada de datos (Input):

- Los usuarios ingresan datos como su nombre, correo, y seleccionan horarios específicos para reservar.
- Los administradores cargan clases, definen horarios y cupos máximos.

Forma parte del componente de recolección de información.

2. Procesamiento de información:

- El sistema valida si hay cupos disponibles.
- Actualiza la base de datos cuando se realiza o cancela una reserva.
- Envía notificaciones automáticas por correo.

Forma parte del sistema que ejecuta la lógica del negocio.

3. Almacenamiento de datos:

- Las reservas, los usuarios, horarios y clases están almacenados en una base de datos estructurada (por ejemplo, en Supabase o PostgreSQL).

Este es el componente de almacenamiento de información.

4. Salida de información (Output):

- El usuario puede consultar sus clases reservadas.

- El administrador puede ver reportes de asistencia o reservas realizadas.

Esta es la distribución y presentación de la información, esencial para la toma de decisiones.

5. Retroalimentación y mejora continua:

- El sistema permite modificar horarios, añadir instructores, o adaptar los cupos según la demanda.
- También recopila información útil para analizar qué clases tienen mayor demanda.

Esta es la capacidad del sistema para evolucionar y adaptarse: una característica clave de los sistemas de información modernos.

Importancia en el desarrollo web

- Automatización de procesos: Permite la reducción del trabajo manual, tal como las confirmaciones por correo.
- Toma de decisiones basada en datos: Puedes saber qué clases son más populares o en qué horarios hay más demanda.
- Escalabilidad: Si el sistema crece, solo sería necesario agregar más funciones o modificar la lógica sin rehacer todo el proyecto.
- Accesibilidad: Los usuarios pueden acceder al sistema desde cualquier navegador, en cualquier momento.

2. Principios de la ingeniería del software:

La ingeniería del software se basa en principios que buscan garantizar que las aplicaciones sean eficientes, robustas, fáciles de mantener y escalables.

1. Modularidad

El modularidad implica dividir el software en componentes o módulos independientes que puedan desarrollarse, probarse y mantenerse por separado.

Aplicación en el proyecto:

En mi aplicación de FitTime, la arquitectura fue dividida en módulos claros:

- Módulo de autenticación de usuarios
- Módulo de gestión de clases
- Módulo de horarios y reservas
- Módulo de notificaciones por correo

Cada módulo cumple una función específica, lo cual facilitó su desarrollo y pruebas por separado, además de permitir futuras mejoras sin afectar todo el sistema.

2. Escalabilidad

La escalabilidad es la capacidad de una aplicación para manejar un aumento en la carga de trabajo sin comprometer su rendimiento.

Aplicación en el proyecto:

Desde el diseño inicial, utilicé una base de datos relacional la cual es Supabase y estructuras de datos eficientes que permiten agregar más clases, usuarios e instructores sin que el rendimiento se degrade.

Además, el frontend está construido para consumir datos dinámicamente, lo que

permite escalar la plataforma a más centros o tipos de clases sin necesidad de rediseñar la base del sistema.

3. Mantenibilidad

La mantenibilidad se refiere a la facilidad con la que el software puede ser corregido, adaptado o mejorado con el tiempo.

Aplicación en el proyecto:

El uso de buenas prácticas de programación, como nombres de variables descriptivos, separación de funciones y comentarios en el código, facilita la comprensión del sistema para futuras modificaciones.

Además, al tener una estructura modular, se pueden hacer mejoras o correcciones en un módulo sin afectar el resto de la aplicación.

4. Reutilización

La reutilización promueve el uso de componentes o funciones ya desarrolladas en diferentes partes del sistema o en otros proyectos.

Aplicación en el proyecto:

Varias funciones, como la validación de formularios o el envío de correos electrónicos, fueron diseñadas como funciones reutilizables. Esto no solo reduce el tiempo de desarrollo, sino que mejora la consistencia del sistema.

5. Fiabilidad

La fiabilidad es la capacidad del software para desempeñarse correctamente bajo condiciones específicas durante un período determinado.

Aplicación en el proyecto:

Se implementaron validaciones para asegurar que los datos ingresados por los usuarios sean correctos, que no se exceda el cupo de clases y que los horarios estén disponibles. Esto reduce errores y mejora la experiencia del usuario.

3. Paradigmas de desarrollo y lenguajes de programación:

En el desarrollo de software, los paradigmas de programación son enfoques o modelos que guían la forma en que se estructura y escribe el código. La elección del paradigma y del lenguaje adecuado es crucial, ya que influye directamente en la eficiencia, claridad, escalabilidad y mantenimiento del proyecto.

➢ Paradigma Orientado a Objetos (OOP)

Este paradigma organiza el código en clases y objetos que representan entidades del mundo real, encapsulando datos y comportamientos.

- Aplicación en el proyecto:

En este caso, aunque no hay clases explícitas en .ts (TypeScript), la lógica del backend ubicada en la carpeta server/api/ sigue principios de OOP al trabajar con objetos estructurados, modularidad y separación de responsabilidades. Por ejemplo:

- El archivo supabase.ts gestiona conexiones y operaciones con Supabase de forma reutilizable y encapsulada.
- Los endpoints como register.post.ts o obtener_usuarios_auth.ts actúan como

funciones organizadas que tratan con entidades como usuarios o autenticación.

➢ Paradigma Funcional

Se basa en funciones puras, evita efectos secundarios y promueve operaciones sobre datos inmutables.

- Aplicación en el proyecto:

En el frontend, los archivos .vue usan JavaScript y TypeScript moderno, que permite un enfoque funcional, especialmente cuando se manejan listas, filtros o transformaciones de datos en tiempo real con métodos como .map(), .filter() y .reduce().

➢ Paradigma Declarativo

Se enfoca en el qué debe hacer el software, en lugar del cómo.

- Aplicación en el proyecto:
  - El uso de HTML dentro de los archivos .vue y el uso de Tailwind CSS representa un enfoque declarativo.
  - Por ejemplo, clases como bg-green-500 o text-center declaran estilos sin preocuparse por la lógica de implementación, lo cual hace que el diseño sea más limpio y claro.

Lenguajes de Programación en el Proyecto

Con base en la estructura del proyecto FitTime, se emplearon varios lenguajes clave:

➢ HTML integrado en .vue

- Uso:

  Define la estructura de las páginas pages/registro.vue, pages/lobby.vue y demás.

- Rol:

  Esencial para mostrar formularios de reserva, botones, tablas de horarios y demás.

➢ CSS a través de Tailwind CSS

- Uso:

  Tailwind permite aplicar estilos de forma rápida y eficiente usando clases de utilidad.

- Ejemplo:
- Clases como bg-blue-500, rounded-lg, p-4 aplican colores, bordes y márgenes sin escribir CSS personalizado.

➢ JavaScript / TypeScript (.ts y .vue)

- Uso:

- La lógica de interacción con el usuario tales como los clicks, validaciones y la navegación se manejan con JavaScript.
- En este proyecto, TypeScript se utiliza para añadir tipado fuerte y mayor robustez, especialmente en los archivos de api y configuración nuxt.config.ts, supabase.ts.
- Rol:

  Permite manejar los datos de usuarios, procesar reservas, consultar la base de datos y comunicarse con Supabase.

➢ No se utilizó PHP ni Python

- Motivo:

  El backend se desarrolló directamente en TypeScript, usando Nuxt 3 y Supabase como backend-as-a-service, lo que elimina la necesidad de servidores PHP tradicionales o scripts en Pytho

Justificación de la elección

- Vue + Nuxt 3: Son opciones ideales para proyectos modernos con múltiples vistas, navegación SPA y SSR, y buena organización del código.
- Tailwind CSS: Acelera el diseño responsivo sin necesidad de escribir CSS desde cero.
- TypeScript: Mejora la calidad del código JavaScript al detectar errores antes de ejecutar, lo que lo hace más mantenible.
- Supabase: Permite conectar y gestionar una base de datos de forma simple y eficiente, reemplazando la lógica que tradicionalmente se haría con PHP o Node puro.

2. **Análisis de Procedimientos y Requerimientos**

La etapa de análisis permite comprender el funcionamiento esperado del sistema, identificar los procesos clave y definir si el desarrollo es viable técnica y económicamente. A continuación, se describen los métodos utilizados para el análisis y la evaluación de la factibilidad del sistema.

1 - Métodos y herramientas para el análisis de procedimientos

Para analizar los procedimientos que involucra la aplicación web de reservas, se utilizarán métodos visuales que permitan representar de forma clara y estructurada el flujo de procesos y las interacciones entre el usuario y el sistema.

Método aplicado:

Se optará por el uso de diagramas de flujo y diagramas de casos de uso, los cuales permiten visualizar las acciones clave dentro del sistema, como el registro, la autenticación, la selección de clases y la confirmación de una reserva. Estos elementos ayudan a comprender cómo fluye la información y qué decisiones se toman en cada etapa.

Herramienta utilizada:

- Lucidchart:

  Esta herramienta en línea permite construir diagramas profesionales con notación estándar para procesos, casos de uso y modelos UML. Se utilizará para modelar:

  - El flujo de interacción del usuario, desde el inicio de sesión hasta la confirmación de la reserva.
  - Los casos de uso que describen las funcionalidades principales del sistema, como gestionar horarios o administrar cupos.

Esta representación visual es clave para validar los procedimientos esperados antes de implementar la lógica del sistema, ya que permite detectar posibles fallos o redundancias en el flujo.

2 - Evaluación de la factibilidad y diseño del sistema ➢ Viabilidad técnica:

El sistema está desarrollado utilizando tecnologías modernas como Vue/Nuxt 3, Tailwind CSS, TypeScript y Supabase, las cuales ofrecen:

- Modularidad y facilidad de mantenimiento.
- Escalabilidad y buen rendimiento.
- Infraestructura en la nube sin necesidad de servidores propios.

Estas herramientas permiten que una sola persona pueda desarrollar el sistema completo de forma organizada y eficiente.

➢ Viabilidad económica:

- No existen costos de licencias.
- Supabase ofrece autenticación y base de datos gratuita en su plan inicial.

- El despliegue podría realizarse con plataformas como Vercel o Netlify, también gratuitas.
- No se requieren inversiones en infraestructura, lo que reduce considerablemente los costos del proyecto.

➢ Viabilidad temporal:

El sistema se puede desarrollar en un plazo estimado de 14 a 15 semanas, divididas en etapas: análisis, diseño, desarrollo, pruebas y ajustes.

3 - Roles y responsabilidades en el equipo, en este caso un único desarrollador.

- Aunque el proyecto fue desarrollado por una sola persona, se asumieron múltiples roles para abarcar todas las áreas del ciclo de vida del software:
- Rol: Responsabilidades principales
- Analista: Identificar los requerimientos funcionales y no funcionales, diseñar los diagramas de flujo y casos de uso en LucidChart.
- Diseñador: Estructurar la interfaz de usuario usando Vue y Tailwind CSS, garantizando una experiencia intuitiva y adaptable.
- Programador: Codificar la lógica del sistema en Nuxt 3 y TypeScript, conectar la aplicación con Supabase, implementar los endpoints de API, y realizar pruebas.
- Administrador del proyecto: Planificar las etapas del desarrollo, gestionar los avances, documentar el proceso y asegurar la calidad del sistema final.

Capítulo 3: Ingeniería de Requerimientos

En este capítulo se detalla el proceso de definición, recolección y modelado de los requerimientos del sistema. La correcta identificación de lo que el sistema debe hacer es esencial para garantizar que el producto final cumpla con las necesidades del usuario y sea técnicamente viable.

Definición y Recolección de Requerimientos

1. Importancia de los requerimientos

Los requerimientos son la base sobre la cual se construye todo el sistema.

Representan las necesidades y expectativas del usuario final respecto al comportamiento del software. Una buena definición de requerimientos garantiza que el

desarrollo esté alineado con los objetivos del proyecto, evitando retrabajos, malentendidos o funciones innecesarias.

Posibles dificultades en este proceso:

- Ambigüedad: El cliente puede expresar sus necesidades de forma poco clara o general.
- Cambios constantes: A medida que el cliente visualiza avances, pueden surgir nuevos requerimientos o cambios en los ya definidos.
- Falta de participación del usuario: Si el usuario no colabora activamente, se dificulta comprender sus necesidades reales.
- Exceso de requerimientos no prioritarios: Puede hacer que el alcance del sistema se descontrole.

2. Métodos para la recolección de requerimientos

Este proyecto fue desarrollado de manera individual y siendo un cliente del lugar, además se cuenta con testimonios de otros clientes, de esta manera se llegó a las necesidades principales del sistema, las cuales son:

➢ Rol de usuario-cliente:

Se elaboró una lista de preguntas clave como:

- ¿Qué funcionalidades desea que tenga la aplicación?
- ¿Qué datos deben capturarse al registrar una clase o una reserva?
- ¿Qué tipo de usuario utilizará el sistema?

➢ Prototipos de baja fidelidad:

Se crearon bocetos visuales (wireframes) de las principales vistas de la aplicación registro.vue, tabla.vue y demás, lo que permitió visualizar la interfaz y validar si cumplía con los objetivos funcionales.

➢ Análisis comparativo:

Se revisaron otras plataformas de reserva similares para identificar requerimientos mínimos estándar del sector.

3.1 Introducción al Pre-diseño Visual

El presente apartado presenta el pre diseño visual del sistema FitTime**,** una plataforma web orientada a la gestión de reservas de clases en un gimnasio. El objetivo de este pre diseño es

establecer una línea gráfica coherente que represente el estilo deportivo del sistema, así como la estructura visual de las pantallas principales que utilizarán los usuarios (miembros/clientes) y los administradores. Este diseño servirá como base para la implementación final del sistema durante las siguientes etapas del proyecto.

3.2 Estructura General de Pantallas

A continuación, se describen las principales pantallas incluidas en el sistema:

Pantalla	Descripción	Usuario que la utiliza
Pantalla de Inicio de Sesión (Login)	Permite al usuario ingresar sus credenciales para acceder al sistema.	Miembro / Administrador
Recuperación de Contraseña	Solicita el correo para enviar un enlace de restablecimiento.	Miembro / Administrador
Dashboard / Panel de Inicio	Da la bienvenida al usuario y muestra el botón principal "Reservar Horario", así como accesos rápidos a próximas clases, progreso e historial.	Miembro
Pantalla de Reserva de Horarios Vista de Próximas	Permite al usuario seleccionar una clase disponible según fecha, instructor y cupo.	Miembro
Clases	Muestra las clases ya agendadas por el usuario.	Miembro
Historial de Reservas	Permite visualizar sesiones pasadas.	Miembro
Panel Administrativo	Incluye gestión de clases, horarios, instructores y cupos.	Administrador
Formulario de de Clase	Formulario para crear nuevas clases con horario y cupos.	Administrador Registro
Reportes	Genera reportes de asistencia y ocupación de clases.	Administrador

3.3 Línea Visual del Sistema

Color	Uso
Verde Lima (#00C853	Color principal para botones y elementos activos, transmite energía y aprox.) acción.
Negro/Gris Oscuro	Fondo principal para dar enfoque al contenido.
Blanco	Texto en elementos oscuros y fondos secundarios.

Gris Claro	Para bordes, cajas de texto y áreas neutrales.

Tipografía propuesta

- Poppins o Montserrat, por su estilo moderno, legible y asociado a tendencias deportivas/ fitness.

Íconos

- Uso de íconos minimalistas de librerías como FontAwesome o Lucide**,** por ejemplo: calendario, reloj, progreso, historial.

Imágenes

- Fotografías de gimnasios modernos con máquinas, que transmitan disciplina, fuerza y profesionalismo.

Logo y Marca

- Marca principal: FitTime**.**
- Enfocada en transmitir una idea de optimización del tiempo y disciplina física.

3.4 Justificación Visual

La paleta de colores se escogió basada en el concepto de energía (verde lima), contraste fuerte (fondo oscuro) y claridad (texto blanco). Estos elementos aportan dinamismo, modernidad y enfoque en la acción, lo cual es coherente con una plataforma de entrenamiento.
La disposición central de botones como Reservar Horario busca priorizar la acción más importante del usuario. Además, el estilo tipo dashboard permite una navegación rápida e intuitiva.

El uso de tipografías modernas refuerza la percepción de una plataforma tecnológica actual y confiable.

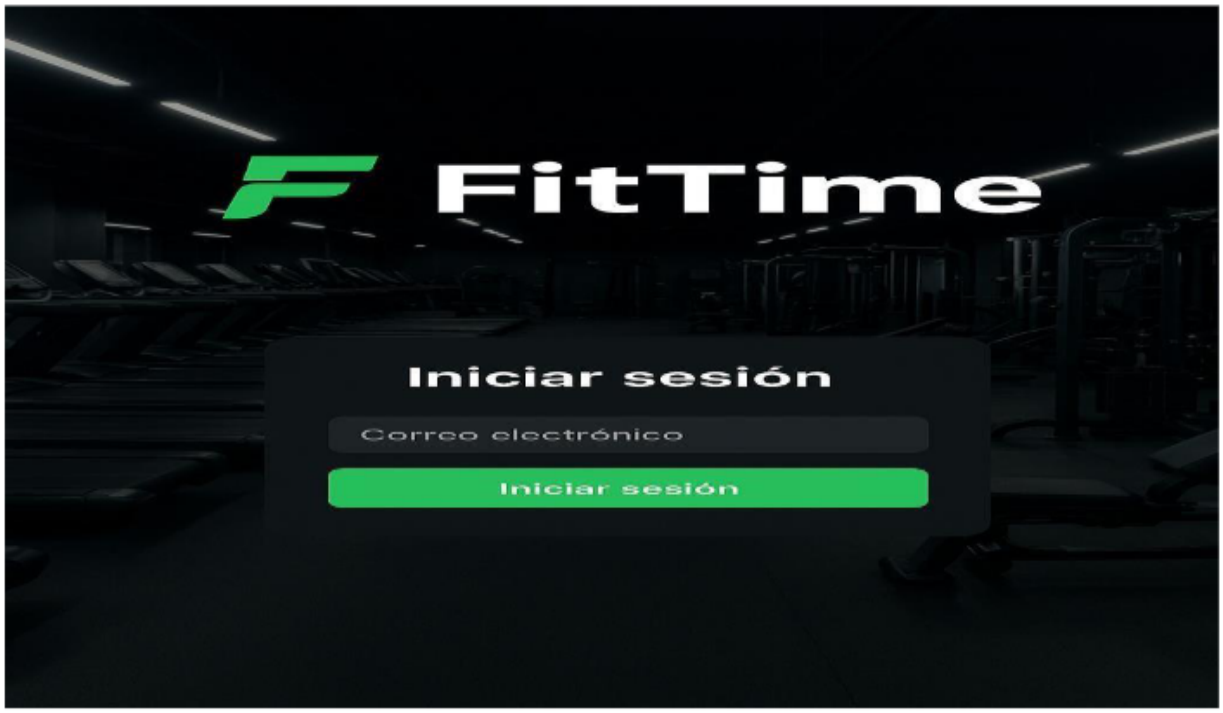
3.5 Ejemplo de Pantalla: Dashboard del Usuario (HomePage)

La siguiente pantalla corresponde al panel inicial del usuario una vez ha iniciado sesión. Se muestra una bienvenida personalizada y se coloca la opción de Reservar Horario como acción principal, seguida de accesos directos a secciones relevantes como próximas clases, progreso e historial de reservas.

Pagina principal


Bienvenido,
Usuario
Reservar Horario
Próximas Clases
Mi Progreso
Reservar Horario
Historial de Reservas

login


FitTime
Iniciar sesión
Correo electrónico
Iniciar sesión

Recuperación de la contraseña

Registro

Perfil

3.6 Conclusión del Pre diseño

Este pre diseño permite visualizar de forma inicial la estructura y apariencia del sistema FitTime, garantizando una dirección clara para el desarrollo. La coherencia gráfica propuesta permitirá mantener una experiencia intuitiva, moderna y alineada con el enfoque deportivo del proyecto.

Capítulo 4: Análisis y diseño

Conceptos Fundamentales

Un sistema de información es un conjunto organizado de elementos que permiten recolectar, procesar, almacenar y distribuir información para apoyar la toma de decisiones, coordinar acciones y controlar procesos

Se desarrolló una aplicación web que permite a los usuarios reservar clases de forma autónoma y visualizar horarios disponibles. Esta aplicación integra funcionalidades como autenticación de usuarios, control de cupos y gestión de clases.

1. Entrada de datos (Input):

- Los usuarios ingresan datos como su nombre, correo, y seleccionan horarios específicos para reservar.
- Los administradores cargan clases, definen horarios y cupos máximos.

2. Procesamiento de información:

- El sistema valida si hay cupos disponibles.
- Actualiza la base de datos cuando se realiza o cancela una reserva.
- Envía notificaciones automáticas por correo.

3. Almacenamiento de datos:

- Las reservas, los usuarios, horarios y clases están almacenados en una base de datos estructurada (por ejemplo, en Supabase o PostgreSQL).

4. Salida de información (Output):

- El usuario puede consultar sus clases reservadas.
- El administrador puede ver reportes de asistencia o reservas realizadas.

5. Retroalimentación y mejora continua:

- El sistema permite modificar horarios, añadir instructores, o adaptar los cupos según la demanda.
- También recopila información útil para analizar qué clases tienen mayor demanda.

Aplicación en el proyecto:

Desde el diseño inicial, utilicé una base de datos relacional la cual es Supabase y estructuras de datos eficientes que permiten agregar más clases, usuarios e instructores sin que el rendimiento se degrade.

Modelado con UML y Diccionarios de Datos

1. Modelado con UML

UML (Lenguaje Unificado de Modelado) se empleó para representar gráficamente los distintos aspectos del sistema, facilitando la comprensión de su estructura, funcionamiento y comportamiento esperado.

Diagramas utilizados:

- Casos de uso:

  Representan las funcionalidades principales del sistema y cómo interactúan los usuarios. Ejemplo: "Reservar clase", "Ver horarios", "Modificar reserva".

- Diagrama de clases:

  Permite identificar las entidades del sistema (Usuario, Clase, Reserva), sus atributos y relaciones. Cada clase representa una tabla o componente en el backend o frontend.

- Diagrama de actividades:

  Modela el flujo lógico de actividades, como el proceso de reserva, paso a paso.

- Diagrama de secuencia:

Describe cómo se comunican los componentes del sistema entre sí en un escenario particular, por ejemplo: el proceso desde que el usuario hace clic en "Reservar" hasta que se guarda en Supabase y se actualiza la interfaz.

Herramienta utilizada:

- Lucidchart para la creación de los diagramas UML.

2. Diccionarios de datos

Un diccionario de datos es una herramienta esencial que permite documentar de forma estructurada los elementos clave del sistema, sus características y relaciones. En este proyecto, se definieron las siguientes entidades:

Ejemplo de estructura del diccionario de datos:

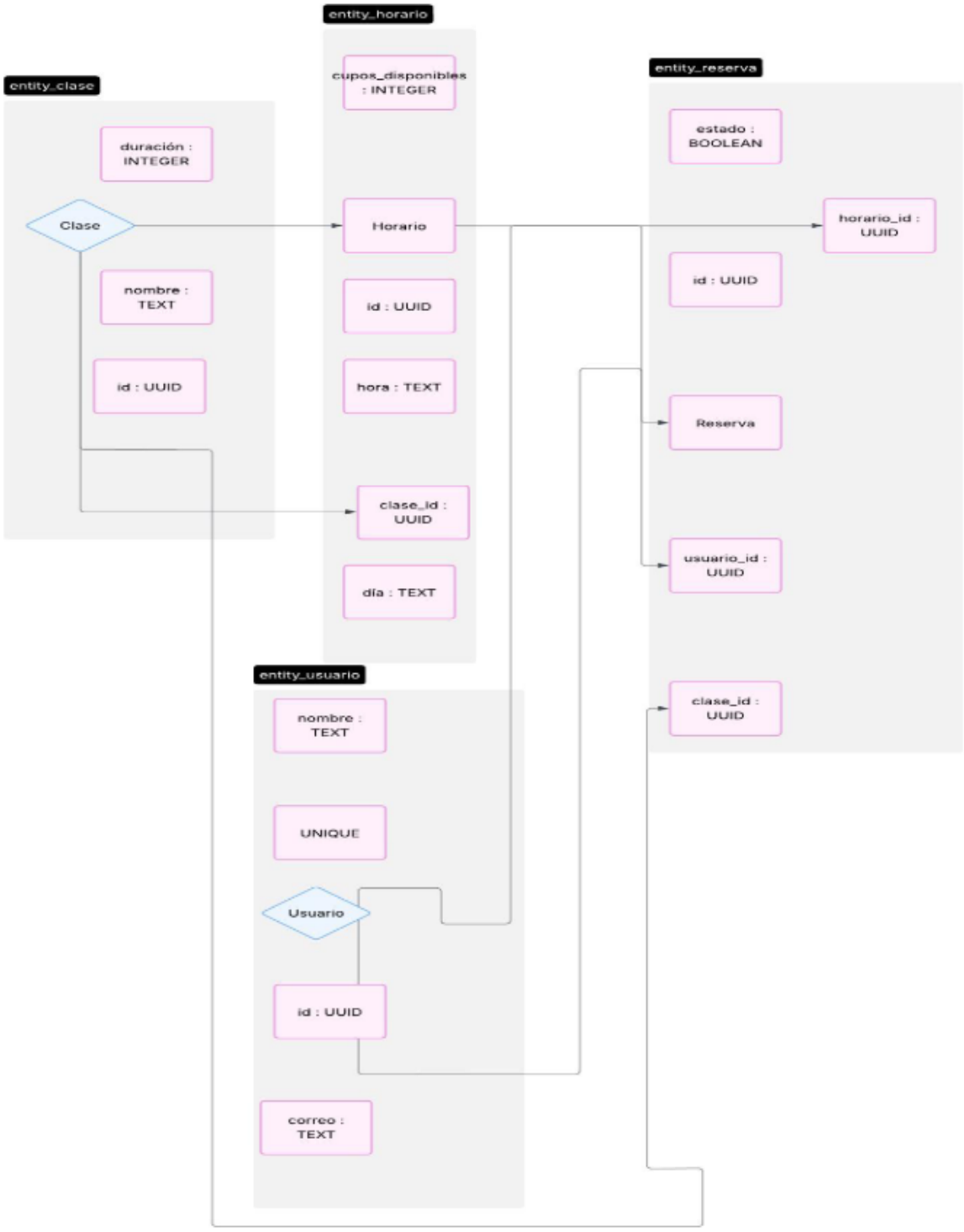
Entidad	**Atributo**	**Tipo de dato**	**Descripción**
Usuario	id	UUID	Identificador único del usuario
	nombre	Texto	Nombre completo del usuario
	correo	Texto (único)	Correo electrónico del usuario
Clase	id	UUID	Identificador de la clase
	nombre	Texto	Nombre o título de la clase
	duracion	Entero	Duración en minutos
Horario	dia	Texto	Día de la semana en que se imparte la clase
	hora	Texto	Hora de inicio de la clase
Reserva	usuario_id	UUID (FK)	Relación con la tabla de usuarios

	clase_id	UUID (FK)	Relación con la tabla de clases
	estado	Texto	Estado de la reserva (activo, cancelado, etc.)

Este diccionario permite mantener una visión clara de la estructura del sistema y sirve de guía para la implementación de la base de datos en Supabase, así como para la validación de formularios y flujos en la interfaz.

Diagramas de Base de Datos

- Diagrama Entidad-Relación (ER) y esquema de base de datos (modelo fisico)



Herramientas de Modelado

Para el desarrollo de los diagramas del sistema, se utilizó Lucidchart, una herramienta en línea que permite modelar de forma visual y profesional diferentes tipos de diagramas. Su interfaz intuitiva y plantillas prediseñadas facilitaron la creación de:

- Diagramas de flujo para representar procesos internos.

- Diagramas UML tales como casos de uso, clases, actividades y secuencia para modelar el comportamiento y la estructura del sistema.
- Diagramas ER y esquemas de base de datos para planificar la estructura lógica y física de la información.

  Lucidchart permite colaborar en tiempo real, personalizar formas y exportar los modelos en diferentes formatos, lo cual fue clave para documentar correctamente el proyecto.

Conexión con la Aplicación

La conexión entre la aplicación Fit Time y la base de datos se realiza mediante Supabase, un servicio que utiliza PostgreSQL como motor principal.
En el proyecto, esta conexión se gestiona a través del archivo supabase.ts, el cual es responsable de inicializar el cliente de Supabase y permitir que el sistema realice operaciones como:

Registro e inicio de sesión de usuarios

Consultas de clases y horarios

Creación y cancelación de reservas

Validaciones de autenticación

Actualizaciones en tablas como usuario, clase, horario y reserva

Supabase funciona como un intermediario entre la aplicación y la base de datos, de forma segura y escalable, utilizando una API que permite ejecutar operaciones CRUD (Create, Read, Update, Delete) mediante funciones asíncronas.

Fragmento de código (supabase.ts)

```ts
import { createClient } from '@supabase/supabase-js'
const supabase_Url = 'https://kgwbhadfbdelnigprnju.supabase.co'
const supabase_Key ='eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJzdXBhYmFzZSIsInJlZiI6Imtnd2JoYWRmYmRlbG5pZ3Bybmp1Iiwicm9sZSI6InNlcn7

const supabase = createClient(supabase_Url, supabase_Key, {
    auth: {
        persistSession: true
    }
})

export default supabase
```

Con esta configuración, cualquier componente o endpoint del proyecto puede importar el cliente de Supabase y realizar consultas directamente a la base de datos, cumpliendo así con la integración requerida entre la capa lógica y la capa de datos del sistema.

Capítulo 5: Desarrollo del backend

5.1 Arquitectura del backend

El backend de la aplicación **FitTime** se implementa sobre una arquitectura **cliente–servidor por capas**, apoyada en un modelo *Backend as a Service* (BaaS) mediante Supabase. Esta arquitectura se compone de tres niveles principales:

1. **Capa de presentación (frontend)**
   Construida con Nuxt 3 y Vue 3 (Composition API), ubicada en la carpeta pages/ (index.vue, registro.vue, login, horarios.vue, perfil.vue, tabla.vue, etc.). Aquí se definen los componentes de interfaz que permiten el registro de usuarios, el inicio de sesión, la visualización de horarios y la gestión de reservas.

2. **Capa de lógica de negocio / API**
   Implementada en Nuxt a través de los handlers en server/api/, donde se concentran las operaciones que requieren lógica adicional o privilegios elevados, especialmente las relacionadas con autenticación y administración de usuarios. Destacan:

   - server/api/auth/register.post.ts
   - server/api/auth/eliminar_usuarios.ts
   - server/api/auth/obtener_usuarios_auth.ts

3. **Capa de datos y servicios externos**
   Proporcionada por **Supabase**, que combina:

   - Servicio de autenticación (Supabase Auth).
   - Base de datos PostgreSQL en la nube, con tablas como usuarios, public_clases, public_horarios y public_reservas.
     La configuración del cliente se centraliza en supabase.ts, desde donde se exporta una instancia compartida a lo largo del proyecto.

De forma resumida, el flujo típico es:
Usuario → Componente Vue (pages/*.vue) → Llamada a API Nuxt (server/api/*) **o** llamada directa al cliente Supabase → Operación en la base de datos / servicio de autenticación → Respuesta al frontend.
Esta arquitectura es adecuada para el alcance del proyecto porque:

- Aprovecha servicios gestionados (Supabase) que reducen la complejidad de infraestructura.
- Separa de forma clara la interfaz, la lógica de negocio y el acceso a datos.
- Facilita la escalabilidad horizontal: tanto Nuxt como Supabase pueden crecer en capacidad sin reescribir el código base.
- Mantiene una estructura modular donde es sencillo añadir nuevos endpoints o ampliar la lógica de negocio.

5.2 Componentes y estructura del código

El proyecto presenta una organización modular que facilita el mantenimiento y la legibilidad del backend. A nivel de carpetas, la estructura relevante para el backend es la siguiente:

- **supabase.ts**
  Archivo de configuración del cliente Supabase:
- import { createClient } from '@supabase/supabase-js'
- const supabase_Url = 'key1'
- const supabase_Key ='key2'
-
- const supabase = createClient(supabase_Url, supabase_Key, {
- auth: {
- persistSession: true
- }
- })
-
- export default supabase

Esta instancia se reutiliza en el resto del proyecto para realizar autenticación y operaciones sobre las tablas.

- **server/api/auth/register.post.ts**
  Endpoint responsable del proceso de registro de usuarios; combina altas en Supabase Auth y creación del registro correspondiente en la tabla usuarios. Incluye lógica para asignar el rol admin cuando el usuario ingresa una clave secreta válida (CLAVE_SECRETA_ADMIN via useRuntimeConfig()).

- **server/api/auth/eliminar_usuarios.ts**
  Endpoint de administración que utiliza supabaseAdmin (cliente con privilegios de servicio) para:

  1. Eliminar el usuario en el módulo de autenticación (auth.admin.deleteUser).
  2. Eliminar el registro asociado en la tabla pública usuarios.

- **server/api/auth/obtener_usuarios_auth.ts**
  Endpoint que, a partir del token sb-access-token almacenado en cookies, valida la sesión y comprueba el rol admin antes de devolver el listado de usuarios (id, correo, nombre) desde la tabla usuarios.

- **composables/auth.ts**
  Composable que encapsula la lógica de autenticación del lado del cliente: inicio y cierre de sesión, recuperación de sesión activa y carga del nombre y rol del usuario desde la tabla usuarios. Maneja estados reactivos como user, userName, userRole y roleLoaded.

- **middleware/auth.ts**
  Middleware global que protege rutas específicas, invocando checkSession() desde el composable useAuth() y redirigiendo al usuario a la página inicial ('/') si no existe sesión activa.

- **Páginas clave (pages/):**

  1. registro.vue: contiene el formulario de registro, validado con Yup, que envía la información al endpoint /api/auth/register.
  2. horarios.vue: se conecta a Supabase para consultar horarios del día y gestionar reservas en public_reservas, aplicando la regla de negocio "un usuario solo puede tener una reserva por día".

Esta estructura permite localizar fácilmente la responsabilidad de cada módulo:

- **Autenticación y roles** → supabase.ts, composables/auth.ts, middleware/auth.ts, /api/auth/*.
- **Gestión de horarios y reservas** → horarios.vue + consultas a tablas public_horarios, public_clases y public_reservas.
- **Administración de usuarios** → /api/auth/obtener_usuarios_auth.ts y /api/auth/eliminar_usuarios.ts.

5.3 Seguridad y validaciones

La seguridad del sistema se basa en cuatro componentes principales:

1. **Autenticación con Supabase Auth**
   El cliente Supabase configurado en supabase.ts se utiliza tanto en el frontend como en los endpoints del backend para gestionar:

   - **Inicio de sesión** mediante supabase.auth.signInWithPassword en useAuth().signIn(email, password).

- **Mantenimiento de sesión** con supabase.auth.getSession() y el listener supabase.auth.onAuthStateChange, que actualizan el estado user y recargan el nombre y rol del usuario.
- **Cierre de sesión** usando supabase.auth.signOut() en el método signOut, que además limpia los estados reactivos y redirige al usuario a la página principal.

2. **Gestión de roles (autorización)**
   El rol del usuario se almacena en la tabla usuarios en la columna tipo_usuario, con dos valores principales:
   - admin: usuarios que ingresan correctamente la CLAVE_SECRETA_ADMIN durante el registro.
   - socio: usuarios regulares que no proporcionan o no aciertan la clave.

El endpoint register.post.ts determina el rol:
const tipoUsuario = body.claveRol === CLAVE_SECRETA_ADMIN ? 'admin' : 'socio'
Posteriormente, obtener_usuarios_auth.ts verifica, a partir del token de acceso y la consulta a usuarios, que el usuario autenticado tenga tipo_usuario = 'admin'. Solo en ese caso devuelve el listado de usuarios; de lo contrario, responde con 403 Acceso denegado.

3. **Validaciones en el frontend (registro de usuarios)**
   En registro.vue se implementan validaciones de entrada mediante Yup:
   - nombre: cadena de al menos 2 caracteres.
   - correo: formato de correo válido.
   - contraseña: mínimo 8 caracteres.
   - telefono: solo números y mínimo 8 dígitos.
   - tipo_usuario: campo opcional utilizado como clave para admin.

Solo si la validación se supera se envía la solicitud POST a /api/auth/register. De esta manera se evita cargar el backend con entradas claramente inválidas.

4. **Reglas de negocio en reservas**
   En horarios.vue, antes de insertar una nueva reserva, se verifica que el usuario no tenga ya una reserva para la fecha actual:

5. const { data: reservasExistentes } = await supabase
6. .from('public_reservas')
7. .select('*')
8. .eq('usuario_id', usuarioActual.value.id)
9. .eq('dia', fechaHoy)

Si reservasExistentes.length > 0, se muestra un mensaje al usuario indicando que ya tiene una reserva para el día de hoy y se aborta la operación. Esta restricción garantiza coherencia en las reservas y evita sobreuso de los cupos.

Adicionalmente, las contraseñas nunca se almacenan en texto plano en la base de datos: Supabase Auth maneja el cifrado y almacenamiento seguro de credenciales, mientras que la tabla usuarios solo guarda referencias (ID de usuario, nombre, correo y tipo de usuario).

5.4 Manejo de errores y pruebas

El backend incorpora gestión de errores principalmente a través de createError en los endpoints de Nuxt, lo que facilita devolver códigos HTTP adecuados y mensajes claros al cliente.

Ejemplos de manejo de errores:

- En register.post.ts:
    - Si la clave de administrador es incorrecta, se lanza un error 403:
    - if (body.claveRol && body.claveRol !== CLAVE_SECRETA_ADMIN) {
    - throw createError({ statusCode: 403, message: "Clave de administrador incorrecta" })
    - }
    - Si signUp en Supabase Auth falla, se devuelve 400 con el mensaje de error.
    - Si no se obtiene el userId o falla la inserción en usuarios, se lanza igualmente un 400.
- En eliminar_usuarios.ts:
    - Si no se recibe el id en el cuerpo de la petición, se retorna un 400 ID de usuario requerido.
    - Si ocurre un error en la eliminación del usuario en Auth o en la tabla usuarios, se generan errores 500 con mensajes específicos para cada etapa (auth vs. tabla pública).
- En obtener_usuarios_auth.ts:
    - Si no hay sb-access-token en cookies, se devuelve 401 No autorizado.
    - Si el token es inválido o la sesión está expirada, también 401.
    - Si el usuario autenticado no es admin, se responde con 403 Acceso denegado: solo administradores.
    - Si ocurre un problema al consultar la tabla usuarios, se responde con 500 Error al obtener usuarios.

En el frontend, los métodos de horarios.vue y composables/auth.ts complementan este manejo con console.error y mensajes al usuario (por ejemplo, alert('Error durante el inicio de sesión: ...') o alert('Error al reservar')).

Pruebas funcionales (propuestas para documentar):

A partir de la lógica implementada, es coherente documentar pruebas del siguiente tipo:

- **Prueba de registro de usuario socio**
    - Endpoint: POST /api/auth/register
    - Datos: nombre válido, correo nuevo, contraseña ≥ 8 caracteres, teléfono válido y sin claveRol.

  - Resultado esperado: respuesta { success: true, tipo_usuario: "socio" } y creación de registro en usuarios con ese rol.

- **Prueba de registro de administrador con clave secreta correcta**
  - Mismos datos que antes pero con claveRol igual a CLAVE_SECRETA_ADMIN.
  - Resultado esperado: { success: true, tipo_usuario: "admin" } y registro en usuarios con tipo_usuario = 'admin'.
- **Prueba de restricciones de reservas diarias**
  - Escenario: usuario ya tiene una reserva en public_reservas para el día actual.
  - Acción: desde horarios.vue se intenta reservar otro horario.
  - Resultado esperado: el sistema muestra el mensaje "Ya tienes una reserva para el día de hoy." y no inserta una nueva fila en public_reservas.

Este tipo de pruebas funcionales, realizadas de forma manual (por ejemplo desde la interfaz o herramientas como Postman), permiten verificar que el manejo de errores y las reglas de negocio están correctamente implementados.

5.5 Servicios externos y escalabilidad

El backend se apoya fuertemente en **Supabase** como servicio externo principal:

- **Autenticación (Supabase Auth)**: gestión de credenciales, sesiones, tokens de acceso y recuperación de usuarios autenticados.
- **Base de datos PostgreSQL administrada**: almacenamiento de la información de usuarios (usuarios), clases (public_clases), horarios (public_horarios) y reservas (public_reservas).
- **API REST y cliente oficial (@supabase/supabase-js)**: permiten realizar operaciones de lectura, inserción, actualización y borrado de forma segura tanto desde el servidor (Nuxt) como desde el cliente.

Desde el punto de vista de escalabilidad:

- La aplicación puede crecer en número de usuarios y en volumen de reservas sin necesidad de reestructurar la arquitectura, ya que Supabase gestiona la infraestructura subyacente.
- Es posible añadir nuevas sedes, tipos de clases o funcionalidades (por ejemplo, listas de espera y reportes de asistencia) simplemente incorporando nuevas tablas o columnas y extendiendo la lógica en server/api y en los componentes Vue.
- La implementación en Nuxt permite un despliegue sencillo en plataformas como Vercel o Netlify, lo que facilita el escalado horizontal de la capa de presentación y la API.

5.6 Conexión frontend–backend–base de datos

La comunicación entre la interfaz y la capa de datos se realiza de dos maneras complementarias:

1. **A través de endpoints del backend (Nuxt server/api)**
   Especialmente para operaciones de autenticación y administración. La tabla siguiente resume los endpoints principales:

Ruta	Método	Descripción	Parámetros principales	Respuesta principal
/api/auth/register	POST	Registra un nuevo usuario en Supabase Auth y en usuarios.	email, password, nombre, telefono, claveRol	{ success: boolean, tipo_usuario: string }
/api/auth/eliminar_usuarios	POST	Elimina un usuario tanto del sistema de auth como de usuarios.	id (ID del usuario en Supabase Auth)	{ message: string }
/api/auth/obtener_usuarios_auth	GET	Devuelve el listado de usuarios, solo si el solicitante es admin.	Token sb-access-token en cookie	Array de usuarios (id, correo, nombre)

Desde el frontend, por ejemplo, registro.vue realiza la llamada:

```
await $fetch('/api/auth/register', {
 method: 'POST',
 body: {
  nombre: nombre.value,
  password: contraseña.value,
  email: correo.value,
  telefono: telefono.value,
  claveRol: tipo_usuario.value
 }
})
```

2. **Mediante el cliente Supabase directamente desde los componentes Vue**

En horarios.vue, el componente utiliza el cliente Supabase para:

- Obtener horarios del día desde public_horarios:
- const { data, error } = await supabase
- .from('public_horarios')
- .select('*, public_clases(nombre)')
- .eq('dia', diaCapitalizado)
- Consultar reservas existentes del usuario en public_reservas:
- const { data, error } = await supabase
- .from('public_reservas')

- .select('horario_id')
- .eq('usuario_id', usuarioActual.value.id)
- .eq('dia', fechaHoy)
- Insertar y eliminar reservas:
- // Insertar reserva
- await supabase.from('public_reservas').insert({
- usuario_id: usuarioActual.value.id,
- clase_id: horario.clase_id,
- horario_id: horario.id,
- estado: 'confirmada',
- dia: fechaHoy
- })
-
- // Cancelar reserva
- await supabase
- .from('public_reservas')
- .delete()
- .eq('usuario_id', usuarioActual.value.id)
- .eq('horario_id', horario.id)
- .eq('dia', fechaHoy)

Este diseño híbrido (endpoints propios + llamadas directas a Supabase) mantiene una capa de backend suficiente para cubrir requisitos de seguridad avanzada y administración, mientras que conserva la simplicidad y rapidez de un BaaS para las operaciones más frecuentes.

5.7 Evidencias y conclusión técnica del backend

Como evidencias de implementación se pueden incluir en el documento final:

- Capturas de pantalla del panel de Supabase mostrando:
  - Tabla usuarios con campos id, nombre, correo, tipo_usuario.
  - Tablas public_clases, public_horarios y public_reservas con registros reales.
- Capturas de la interfaz:
  - Formulario de registro (registro.vue) con validaciones activas.

- Pantalla de horarios (horarios.vue) mostrando la lista de clases del día y el estado de las reservas.
- Panel o tabla donde el administrador visualiza usuarios utilizando /api/auth/obtener_usuarios_auth.

Conclusión técnica

El backend de FitTime integra de forma coherente Nuxt 3 y Supabase para ofrecer un sistema de reservas robusto, con autenticación diferenciada por rol, control de acceso a rutas administrativas y reglas de negocio claras (como la limitación de una reserva diaria por usuario). La modularidad de la estructura de carpetas (server/api, composables, middleware) facilita la localización y ampliación de funcionalidades, mientras que el uso de Supabase reduce la complejidad de gestión de infraestructura y mejora la escalabilidad potencial del sistema.

El manejo de errores mediante createError, las validaciones en frontend con Yup y las verificaciones de rol en los endpoints críticos contribuyen a la fiabilidad y seguridad del flujo de datos. A futuro, el backend puede ampliarse incorporando endpoints dedicados para la gestión de horarios y reservas (en lugar de todas las operaciones desde el cliente), así como mecanismos adicionales de auditoría y generación de reportes para la administración de la operación diaria del gimnasio.