**DOCUMENTO DE ARQUITECTURA DE SOFTWARE PARA EL SISTEMA ZENTRO**

**PROYECTO: ZENTRO**

**FECHA DE EMISIÓN: 31 DE AGOSTO DE 2025**

**AUTORES Y RESPONSABILIDADES:**

**JEFE DE PROYECTO: JULIAN ANDRES PERDOMO REINOSO**

**VISIÓN GENERAL, OBJETIVOS, ALCANCE.**

**ARQUITECTO DE SOFTWARE: GEMINI - DISEÑO DE LA ARQUITECTURA, DIAGRAMAS.**

**VERSIÓN DEL DOCUMENTO: 1.0**

**VERSIÓN 1.0 (31/08/2025): CREACIÓN INICIAL DEL DOCUMENTO.**

Tabla de Contenido

[RESUMEN EJECUTIVO 6](#_Toc207566669)

[Propósito del documento 6](#_Toc207566670)

[Alcance y límites: 6](#_Toc207566671)

[Público objetivo: 6](#_Toc207566672)

[Resumen de la solución (alto nivel): 6](#_Toc207566673)

[INTRODUCCIÓN 7](#_Toc207566674)

[OBJETIVOS 8](#_Toc207566675)

[Objetivo General 8](#_Toc207566676)

[Objetivos Específicos 8](#_Toc207566677)

[Contexto organizacional 8](#_Toc207566678)

[Referencias normativas y técnicas 8](#_Toc207566679)

[Definiciones y acrónimos 8](#_Toc207566680)

[DESCRIPCIÓN GENERAL DEL SISTEMA 9](#_Toc207566681)

[Visión del producto 9](#_Toc207566682)

[Problema que resuelve 9](#_Toc207566683)

[Stakeholders y roles 9](#_Toc207566684)

[Ecosistema / Integraciones externas 9](#_Toc207566685)

[Supuestos, restricciones y dependencias 9](#_Toc207566686)

[REQUERIMIENTOS ARQUITECTÓNICOS 10](#_Toc207566687)

[Requerimientos funcionales clave 10](#_Toc207566688)

[Requerimientos no funcionales 10](#_Toc207566689)

[Criterios de aceptación y métricas 10](#_Toc207566690)

[VISTAS DE ARQUITECTURA (4+1 / KRUCHTEN) 11](#_Toc207566691)

[Vista lógica 11](#_Toc207566692)

[Vista de desarrollo 11](#_Toc207566693)

[Vista de procesos (runtime) 11](#_Toc207566694)

[Vista física / de despliegue 11](#_Toc207566695)

[Vista de escenarios / casos de uso 11](#_Toc207566696)

[MODELOS Y DIAGRAMAS 13](#_Toc207566697)

[Diagrama de contexto (C4 nivel 1) 13](#_Toc207566698)

[Diagrama de contenedores / componentes (C4 nivel 2) 13](#_Toc207566699)

[Diagrama de clases 14](#_Toc207566700)

[Diagramas de secuencia / interacción 15](#_Toc207566701)

[Diagrama de despliegue: 15](#_Toc207566702)

[Diagramas ER / modelos de datos: 15](#_Toc207566703)

[Ubicación y formatos: 15](#_Toc207566704)

[Plantillas y templates: 16](#_Toc207566705)

[INTERFACES, API Y CONTRATOS 16](#_Toc207566706)

[API REST / GraphQL: 16](#_Toc207566707)

[Esquemas de eventos y brokers: 16](#_Toc207566708)

[Contratos de versión: 16](#_Toc207566709)

[Especificaciones de integración: 16](#_Toc207566710)

[DECISIONES DE DISEÑO ARQUITECTÓNICO (ADRS) 17](#_Toc207566711)

[Lista de ADRs con resumen: 17](#_Toc207566712)

[Decisiones clave: 17](#_Toc207566713)

[TECNOLOGÍAS Y JUSTIFICACIÓN 18](#_Toc207566714)

[Lenguajes y frameworks: 18](#_Toc207566715)

[Bases de datos y almacenamiento: 18](#_Toc207566716)

[Infraestructura: 18](#_Toc207566717)

[Librerías y dependencias: 18](#_Toc207566718)

[SEGURIDAD Y CUMPLIMIENTO 19](#_Toc207566719)

[Modelo de seguridad: 19](#_Toc207566720)

[Autenticación y autorización: 19](#_Toc207566721)

[Gestión de secretos y cifrado: 20](#_Toc207566722)

[Logging, auditoría y evidencia: 20](#_Toc207566723)

[Requisitos legales y normativos: 20](#_Toc207566724)

[GESTIÓN DE DATOS 20](#_Toc207566725)

[Modelo de datos y esquemas principales: 21](#_Toc207566726)

[Estrategia de persistencia: 21](#_Toc207566727)

[Políticas de retención, purga y anonimización: 21](#_Toc207566728)

[Backup, restore y verificación: 21](#_Toc207566729)

[Estrategia de Migración de Datos: 21](#_Toc207566730)

[ESCALABILIDAD Y DISPONIBILIDAD 22](#_Toc207566731)

[Estrategia de escalado: 22](#_Toc207566732)

[Balanceo de carga: 22](#_Toc207566733)

[Alta disponibilidad: 22](#_Toc207566734)

[RESILIENCIA Y RECUPERACIÓN ANTE DESASTRES 23](#_Toc207566735)

[Patrones de tolerancia a fallos 23](#_Toc207566736)

[Estrategia de DR: 23](#_Toc207566737)

[OBSERVABILIDAD Y MONITOREO 24](#_Toc207566738)

[Logs: 24](#_Toc207566739)

[Métricas: 24](#_Toc207566740)

[Tracing distribuido: 25](#_Toc207566741)

[Dashboards y alertas: 25](#_Toc207566742)

[OPERACIONES Y DESPLIEGUE (CI/CD) 26](#_Toc207566743)

[Entornos: 26](#_Toc207566744)

[Pipelines CI/CD: 26](#_Toc207566745)

[Herramientas de provisioning: 26](#_Toc207566746)

[MANTENIBILIDAD Y EVOLUCIÓN 27](#_Toc207566747)

[Estrategia de versionado: 27](#_Toc207566748)

[Prácticas de calidad: 27](#_Toc207566749)

[ESTRATEGIA DE PRUEBAS Y CALIDAD 28](#_Toc207566750)

[Tipos de pruebas: 28](#_Toc207566751)

[Cobertura mínima: 28](#_Toc207566752)

[COSTOS Y PROYECCIONES 29](#_Toc207566753)

[Estimación de costos: 29](#_Toc207566754)

[CUMPLIMIENTO LEGAL Y PROTECCIÓN DE DATOS 30](#_Toc207566755)

[Políticas: 30](#_Toc207566756)

[Requisitos legales: 30](#_Toc207566757)

[ANEXOS 31](#_Toc207566758)

[Glosario de términos: 31](#_Toc207566759)

[Lista de abreviaturas: 31](#_Toc207566760)

[Enlaces a repositorios, diagramas fuente y scripts: 31](#_Toc207566761)

[Plantillas útiles: 31](#_Toc207566762)

[APÉNDICES TÉCNICOS 32](#_Toc207566763)

[PlantUML / Diagrams.net sources 32](#_Toc207566764)

[Ejemplos de configuración 32](#_Toc207566765)

[Scripts de provisión 32](#_Toc207566766)

# 

# RESUMEN EJECUTIVO

Propósito del documento**:** Describir la arquitectura del software Zentro, un sistema integral de gestión para gimnasios, abarcando decisiones técnicas, componentes y lineamientos para su desarrollo y operación.

Alcance y límites: El sistema soportará la gestión de clientes, membresías, entrenadores, rutinas, nutrición, pagos, reservas y tienda. Las funciones avanzadas como IA, multisede y analítica predictiva están fuera del alcance de la versión 1.0.

Público objetivo: Desarrolladores, administradores de sistemas, *stakeholders* y cualquier persona que necesite comprender el diseño y la construcción del sistema.

Resumen de la solución (alto nivel): Zentro es una aplicación modular basada en una arquitectura cliente-servidor con un *backend* en FastAPI (Python), un *frontend* en Vue.js/React y una base de datos MySQL.

# INTRODUCCIÓN

Este **Documento de Arquitectura de Software** describe la estructura, decisiones y lineamientos técnicos del sistema **Zentro** —una plataforma integral de gestión para gimnasios— con el propósito de garantizar que cualquier miembro del equipo (o un ingeniero nuevo) pueda comprender, desplegar, mantener y evolucionar la solución de forma fiable y reproducible.

La arquitectura documentada aquí corresponde a la **Versión 1.0 (31/08/2025)** y cubre el diseño lógico, los principales componentes y sus interacciones, los diagramas (contexto, contenedores, clases, secuencias y ERD), los requisitos funcionales y no funcionales relevantes, las decisiones arquitectónicas (ADRs), y las pautas de despliegue y operación. El alcance técnico de esta versión incluye gestión de clientes, membresías, entrenadores, rutinas, pagos, reservas y tienda; funciones avanzadas como IA, analítica predictiva y multisede quedan fuera del alcance de la 1.0 y se tratarán en futuras versiones según roadmap.

Este documento adopta y referencia buenas prácticas y estándares internacionales aplicables a la arquitectura de software (por ejemplo, ISO/IEC/IEEE 42010, el modelo 4+1 de Kruchten y vistas C4) y mapea, además, los controles y requisitos legales relevantes para Colombia (p. ej. Ley 1581/2012 y normativa de facturación electrónica). Está pensado para varios públicos: desarrolladores backend y frontend, ingenieros de DevOps, QA, seguridad, gestores de producto y responsables legales/operativos; cada sección indica claramente su audiencia objetivo y cómo usar los artefactos incluidos (diagramas fuente, PlantUML, OpenAPI, ADRs y runbooks).

La estructura del documento está organizada para facilitar la navegación y la trazabilidad: desde el resumen ejecutivo y los requisitos, pasando por las vistas arquitectónicas y los modelos de datos, hasta las secciones operativas (CI/CD, observabilidad, backups) y los anexos con plantillas, scripts y fuentes de los diagramas (docs/architecture/). Se recomienda mantener este documento **vivo**: cualquier cambio significativo en la arquitectura debe registrarse mediante un ADR y actualizar las secciones y diagramas correspondientes.

Para dudas, ampliaciones o pruebas de concepto relacionadas con cualquiera de las decisiones aquí consignadas, contactar a los autores y responsables indicados en la portada.

# OBJETIVOS

Objetivo General

Desarrollar una aplicación integral de gestión para gimnasios que optimice la administración de clientes, entrenadores, clases, pagos y servicios complementarios.

### Objetivos Específicos

* + - Implementar un módulo de clientes con historial de pagos y rutinas.
    - Crear un módulo de control de membresías y pagos con notificaciones automáticas.
    - Integrar sistemas de rutinas y nutrición personalizados.
    - Garantizar la seguridad y escalabilidad mediante una arquitectura modular y roles de usuario.

Contexto organizacional

El proyecto está siendo desarrollado por un equipo multidisciplinario con el objetivo de ofrecer una solución de software que sea asequible, adaptable a las necesidades locales y escalable para el mercado colombiano.

Referencias normativas y técnicas

Se aplican las directrices de la arquitectura 4+1 de Kruchten y las vistas de arquitectura C4 para la documentación.

Definiciones y acrónimos

Se adjunta un glosario en el Anexo 21.1.

# 

**Estado del Arte y Análisis de la Competencia**

**1. Panorama del Mercado Actual**

El mercado de software de gestión para gimnasios está dominado por soluciones globales y, en menor medida, por actores locales. Las soluciones globales como **Mindbody** o **Zenplanner** ofrecen una amplia gama de funcionalidades (desde programación de clases y pagos hasta marketing), pero a menudo son costosas y tienen una curva de aprendizaje pronunciada, lo que las hace menos atractivas para los gimnasios más pequeños o independientes.

A nivel local en Latinoamérica, la oferta es variada. Existen soluciones con funcionalidades básicas de administración y facturación, pero generalmente carecen de:

* Una **experiencia de usuario (UX)** moderna y fluida.
* Funcionalidades avanzadas como **integraciones con pagos en línea**, facturación electrónica o analítica de datos.
* Una arquitectura modular que permita una fácil escalabilidad y personalización.

Este panorama representa una oportunidad para **Zentro**, que puede posicionarse como una solución **integral, accesible y adaptada** a las necesidades específicas de la región, combinando un precio justo con una arquitectura robusta y la incorporación de tecnologías de vanguardia.

**2. Tecnologías y Tendencias Emergentes**

Para superar a la competencia y asegurar su relevancia a largo plazo, Zentro debe integrar o tener un plan para incorporar las siguientes tendencias tecnológicas:

* **Inteligencia Artificial (IA) y Aprendizaje Automático (ML):**
  + **Planes de Entrenamiento y Nutrición Personalizados:** Usar algoritmos de ML para sugerir rutinas y dietas basadas en los datos de progreso, objetivos y condiciones físicas del cliente.
  + **Análisis Predictivo de Abandono (Churn):** Analizar el comportamiento del cliente (frecuencia de asistencia, uso de servicios) para predecir quiénes están en riesgo de abandonar el gimnasio. El sistema podría enviar alertas a los administradores para que tomen medidas de retención proactivas.
  + **Automatización de Soporte:** Implementar *chatbots* para responder preguntas frecuentes, asistir con la reserva de clases y resolver problemas básicos de manera inmediata.
* **Integración con *Wearables* y Salud Digital:**
  + Sincronizar datos de actividad física de dispositivos como **Apple Health, Google Fit, Garmin y Fitbit**. Esto permitiría a los entrenadores y clientes tener un seguimiento más completo de su progreso dentro y fuera del gimnasio, enriqueciendo los datos del perfil del cliente en Zentro.
  + **API Abiertas:** Zentro podría ofrecer su propia API para que desarrolladores de terceros creen integraciones con otras aplicaciones o dispositivos de salud.
* **Gamificación y Comunidad:**
  + Implementar elementos de juego como **tablas de clasificación**, logros y recompensas virtuales para motivar a los clientes a asistir con regularidad y alcanzar sus metas. Esto crea un sentido de competencia amistosa y fomenta la retención.
  + **Muro de la Comunidad:** Un espacio dentro de la aplicación donde los clientes y entrenadores puedan interactuar, compartir fotos de su progreso y celebrar logros.
* **Pagos Inteligentes y Financieros (*Fintech*):**
  + **Pagos Recurrentes Automáticos:** Permitir a los clientes configurar pagos automáticos para renovar sus membresías, reduciendo el trabajo manual y mejorando la puntualidad de los ingresos.
  + **Integración con Pagos Móviles:** Permitir pagos a través de **Apple Pay, Google Pay** y otras billeteras digitales.
  + **Facturación Electrónica:** La integración con la **DIAN** es un requisito legal y una funcionalidad clave. Zentro debe ofrecer una solución que automatice y simplifique este proceso para los gimnasios.
* **Arquitectura Modular y Escalable:**
  + El uso de **microservicios** o una arquitectura modular permite a Zentro agregar nuevas funcionalidades como la tienda, la gestión de nutriólogos o el análisis predictivo sin afectar el núcleo del sistema, facilitando la evolución del producto.

Al considerar estas tendencias, Zentro no solo resuelve los problemas actuales, sino que también se posiciona como una solución innovadora y lista para el futuro del sector del *fitness*.

**Requisitos del Sistema Zentro**

En esta fase, se documenta todo lo que el sistema debe hacer, sin entrar en detalles técnicos de implementación. Se divide en dos categorías:

**Requisitos Funcionales**

Estos son los "qué". Describen las características del producto desde la perspectiva del usuario. Para Zentro, los requisitos funcionales clave son:

* **Gestión de Clientes:** Crear, editar, eliminar y consultar perfiles de clientes. Incluye el historial de pagos, rutinas y datos de salud.
* **Gestión de Membresías y Pagos:** Registrar pagos, renovar membresías automáticamente, emitir facturas y enviar notificaciones de vencimiento.
* **Sistema de Clases y Reservas:** Un calendario para programar clases, permitir a los clientes reservar su cupo y gestionar listas de espera.
* **Módulo de Entrenadores:** Perfiles de entrenadores, asignación a clientes y la creación de planes de entrenamiento y nutrición.
* **Inventario y Ventas:** Gestionar productos en la tienda del gimnasio (bebidas, suplementos) y registrar ventas.
* **Acceso y Seguridad:** Un sistema de *login* con diferentes roles de usuario (administrador, recepcionista, entrenador, cliente) y un método de *check-in* rápido (ej. con código QR).

**Requisitos No Funcionales**

Estos son los "cómo". Describen las cualidades del sistema y cómo se comportará en diferentes situaciones. Son cruciales para la arquitectura.

* **Rendimiento:** El sistema debe responder en menos de 500 milisegundos a las peticiones críticas (ej. *login*, registro de pago).
* **Seguridad:** La información del cliente debe estar cifrada en tránsito y en reposo. El sistema debe ser resistente a inyecciones de código y otros ataques comunes.
* **Disponibilidad:** El sistema debe tener una disponibilidad del 99.9% para evitar interrupciones en las operaciones del gimnasio.
* **Escalabilidad:** Debe ser capaz de manejar un aumento de usuarios y datos sin comprometer el rendimiento.
* **Usabilidad:** La interfaz de usuario debe ser intuitiva, fácil de usar y compatible con dispositivos móviles.
* **Mantenibilidad:** El código debe ser modular y estar bien documentado para facilitar futuras actualizaciones.

**Diseño de Alto Nivel**

En esta etapa, se crea un esquema visual de la arquitectura del sistema, sin decidir aún la tecnología específica. Los diagramas de alto nivel que creaste en tu documento arquitectónico (como el de contexto y el de contenedores) son ejemplos perfectos de esta fase.

* **Diagrama de Contexto:** Muestra a Zentro como una "caja" y cómo interactúa con el mundo exterior: el cliente, el administrador, las pasarelas de pago y la DIAN (entidad fiscal colombiana).
* **Diagrama de Contenedores:** Abre esa "caja" y muestra los componentes principales de Zentro: el frontend, el backend y la base de datos. Explica cómo se comunican entre sí.
* **Flujos de Usuario:** Se documentan los caminos que un usuario toma dentro del sistema. Por ejemplo, el flujo de "registro de un cliente nuevo" o el de "reserva de una clase".

# DESCRIPCIÓN GENERAL DEL SISTEMA

Visión del producto

Un software modular, escalable y adaptable para gimnasios de todos los tamaños, que integra servicios adicionales (nutrición, tienda, accidentes) con potencial de expansión futura.

Problema que resuelve

La falta de soluciones integrales, asequibles y adaptadas al contexto latinoamericano en el mercado de la gestión de gimnasios.

Stakeholders y roles

Administrador, recepcionista, entrenador, cliente y equipo de desarrollo.

Ecosistema / Integraciones externas

Pasarelas de pago (Stripe/PayU/MercadoPago), servicios de notificación (SMS/Email), servicios de facturación electrónica (DIAN).

Supuestos, restricciones y dependencias

El sistema operará bajo la legislación colombiana (Ley 1581 de 2012), la infraestructura de nube debe ser escalable (AWS/GCP/Azure) y se requiere un certificado digital para la facturación

# REQUERIMIENTOS ARQUITECTÓNICOS

## Requerimientos funcionales clave

**Clientes:** CRUD completo, historial médico, rutinas y pagos.

**Recepción:** Check-in con QR y pases de invitados.

**Membresías y Pagos:** Gestión de planes, cobros y facturación.

**Clases y Reservas:** Calendario, reservas en línea, listas de espera.

**Entrenadores:** Perfiles, horarios, asignación a clases y clientes.

**Rutinas y Nutrición:** Creación y asignación de planes personalizados.

**Tienda:** Inventario, registro de ventas, alertas de stock.

**Administración:** Gestión de roles y permisos.

## Requerimientos no funcionales

**Rendimiento:** Tiempo de respuesta <500 ms.

**Seguridad:** Uso de JWT y cifrado de datos sensibles.

**Disponibilidad:** 99.9% de tiempo de actividad.

**Mantenibilidad:** Código modular y bien documentado.

**Usabilidad:** Diseño intuitivo y *responsive* para múltiples dispositivos.

## Criterios de aceptación y métricas

**SLI (Indicadores de Nivel de Servicio):** Latencia de la API, tasa de errores, disponibilidad.

**SLO (Objetivos de Nivel de Servicio):** 99.9% de disponibilidad mensual, 99.5% de peticiones con latencia <500 ms.

# VISTAS DE ARQUITECTURA (4+1 / KRUCHTEN)

## Vista lógica

**Módulos y componentes principales:** Autenticación, Clientes, Membresías, Pagos, Clases, Rutinas, Nutrición, Tienda, Administración.

**Diagramas de componentes:** Se detalla en la sección 7.2.

## Vista de desarrollo

**Estructura del código y paquetes:** El código está organizado en carpetas backend/ y frontend/ con subcarpetas para api/, db/, core/, components/, etc.

**Repositorios y branching model:** Se usará un único repositorio Git con un modelo de *branching* tipo **Gitflow**.

## Vista de procesos (runtime)

**Flujos de ejecución y concurrencia:** Las peticiones son procesadas de manera asíncrona por el *backend* de FastAPI.

**Colas, mensajería y coordinación:** Se utilizarán colas de mensajes (ej. con Celery) para tareas de fondo como el envío de notificaciones automáticas y alertas.

## Vista física / de despliegue

**Topología de despliegue:** Se usarán entornos de desarrollo, pruebas (*staging*) y producción.

**Infraestructura y redes:** Se utilizarán contenedores **Docker** y orquestación con Docker Compose para el despliegue.

## Vista de escenarios / casos de uso

**Escenarios críticos:**

**Login de Usuario:** Un usuario introduce sus credenciales y recibe un JWT.

**Registro de Pago:** Un recepcionista registra un pago, lo que actualiza la membresía del cliente.

**Backup de la BD:** Un script automatizado realiza un *dump* de la base de datos.

**Matriz de trazabilidad:** Se mantendrá una matriz de trazabilidad para asegurar que cada requisito no funcional tenga un escenario de prueba asociado.

MODELOS Y DIAGRAMAS

## Diagrama de contexto (C4 nivel 1)

* @startuml contexto !define ICONURL <https://raw.githubusercontent.com/tupadr3/plantuml-icon-font-sprites/master> title Diagrama de Contexto - Sistema Zentro
* actor "Cliente de Gimnasio" as cliente actor "Administrador del Gimnasio" as admin actor "Recepcionista" as recepcion actor "Entrenador" as entrenador
* boundary "Sistema Zentro" as zentro { card "API Backend" as backend card "Frontend Web" as frontend } database "Base de Datos MySQL" as db
* entrenador -- backend : Gestiona rutinas admin -- backend : Gestiona el sistema recepcion -- backend : Check-in y pagos cliente -- frontend : Accede a su perfil frontend ..> backend : Peticiones API REST backend --> db : CRUD de datos
* boundary "Pasarela de Pagos" as pagos { card "Stripe / PayU" as stripe } boundary "Servicios de Notificaciones" as notif { card "Sendgrid / Twilio" as twilio } boundary "Facturador Electrónico" as dian { card "DIAN" as dian\_api }
* backend --> stripe : Pagos y webhooks backend --> twilio : Notificaciones Email/SMS backend --> dian\_api : Facturación electrónica
* @enduml

## Diagrama de contenedores / componentes (C4 nivel 2)

* @startuml contenedores
* !include https://raw.githubusercontent.com/plantuml-stdlib/Cicon-PlantUML/master/Cicon-Common.puml
* title Contenedores del Sistema Zentro
* node "Usuario (web browser)" {
* component "Frontend Zentro" as frontend\_web
* }
* node "Servidor Cloud (AWS/GCP)" as cloud {
* component "Nginx (Proxy)" as nginx
* component "API Backend FastAPI" as api
* component "Celery Worker" as worker
* }
* node "Base de Datos" as db {
* component "MySQL" as mysql
* }
* node "Servicios Externos" {
* component "Pasarela de Pagos" as pagos
* component "Servicios de Email/SMS" as notif
* component "Facturador Electrónico" as facturador
* }
* frontend\_web --> nginx : HTTP/S
* nginx --> api : Reverse Proxy
* api --> mysql : Lectura/Escritura SQL
* api --> worker : Cola de Tareas Asíncronas
* api --> pagos : REST API
* api --> notif : REST API
* api --> facturador : REST API
* @enduml

## Diagrama de clases

* @startuml clases
* title Diagrama de Clases - Módulo Clientes y Membresías

class Cliente { +id: UUID +nombre: string +email: string +telefono: string +fecha\_nacimiento: date +peso: decimal +altura: decimal +estado\_activo: boolean

* +registrar()
* +actualizar\_datos()
* +asignar\_membresia(membresia\_id)
* }

class Membresia { +id: UUID +cliente\_id: UUID +tipo: string +fecha\_inicio: date +fecha\_vencimiento: date +estado: string (activa, vencida, suspendida)

* +renovar()
* +suspender()
* }
* class Plan {
* +id: UUID
* +nombre: string (Básico, Pro, Preferencial)
* +precio: decimal
* +duracion\_dias: int
* }
* class Pago {
* +id: UUID
* +membresia\_id: UUID
* +monto: decimal
* +fecha\_pago: datetime
* +metodo: string (tarjeta, PSE, efectivo)
* }
* Cliente "1" -- "" Membresia : posee
* Membresia "1" -- "1" Plan : se basa en
* Membresia "1" -- "" Pago : tiene
* @enduml

## Diagramas de secuencia / interacción

* @startuml secuencia
* title Flujo: Registro de Pago con Tarjeta
* actor Recepcionista
* participant "Frontend (Web)" as Frontend
* participant "API Backend (Zentro)" as API
* participant "Pasarela de Pagos (Stripe)" as Stripe
* participant "Base de Datos (MySQL)" as DB
* Recepcionista -> Frontend : Inicia proceso de pago
* Frontend -> API : POST /api/v1/pagos/checkout {cliente\_id, plan\_id}
* API -> DB : Consulta datos de cliente y plan
* DB --> API : Retorna datos
* API -> Stripe : Crea sesión de checkout
* Stripe --> API : Retorna URL de checkout
* API --> Frontend : Retorna URL de checkout
* Frontend --> Stripe : Redirecciona al usuario a la página de pago
* ... el cliente completa el pago ...
* Stripe -> API : Webhook: pago\_exitoso
* API -> DB : Actualiza estado de membresía a 'activa'
* DB --> API : Confirmación
* API -> DB : Registra el pago en tabla 'pagos'
* DB --> API : Confirmación
* API -> Frontend : Notifica pago exitoso
* @enduml

Diagrama de despliegue: Se usará Docker Compose para el entorno de desarrollo y un despliegue en contenedores orquestados con Kubernetes para producción.

Diagramas ER / modelos de datos: Se adjuntará un diagrama ER completo de la base de datos.

Ubicación y formatos: Los diagramas se guardarán en docs/architecture/diagrams/ en formato .puml y se renderizarán a PNG/SVG.

Plantillas y templates: (Se omite la sección para no repetir contenido)

INTERFACES, API Y CONTRATOS

API REST / GraphQL: Se implementará una API RESTful. La especificación completa se encuentra en el archivo openapi.yaml en la ruta docs/api/openapi.yaml.

Esquemas de eventos y brokers: Se usarán eventos para las notificaciones a clientes, como la expiración de una membresía o la reserva de una clase.

Contratos de versión: Se utilizará un versionado semántico de la API (/v1/).

Especificaciones de integración: Se documentarán los *webhooks* y las credenciales necesarias para cada integración.

# DECISIONES DE DISEÑO ARQUITECTÓNICO (ADRS)

Lista de ADRs con resumen: Se mantendrá un registro de las decisiones clave en una carpeta docs/architecture/adr/.

## Decisiones clave:

**Backend: Se eligió FastAPI por su alto rendimiento, su documentación automática (Swagger) y la robustez del ecosistema de Python.**

**Frontend: Vue.js o React por su popularidad, gran comunidad y facilidad para construir SPAs (Single Page Applications) modulares.**

**Base de Datos: MySQL por su fiabilidad, madurez y compatibilidad con la mayoría de los servicios de alojamiento en la nube.**

# TECNOLOGÍAS Y JUSTIFICACIÓN

Lenguajes y frameworks: Python (FastAPI), JavaScript (Vue.js/React).

Bases de datos y almacenamiento: MySQL (Base de datos principal).

Infraestructura: Se utilizarán servicios de nube (AWS, GCP) para el despliegue en producción.

Librerías y dependencias: Se gestionarán con requirements.txt (Python) y package.json (Node.js).

# SEGURIDAD Y CUMPLIMIENTO

Modelo de seguridad: Se aplicarán principios de seguridad como el "mínimo privilegio" (RBAC) y el cifrado de datos en tránsito (HTTPS) y en reposo.

Autenticación y autorización: Se utilizará JWT para la autenticación y un modelo de RBAC (Control de Acceso Basado en Roles) para la autorización.

Gestión de secretos y cifrado: Las claves de API y secretos se gestionarán a través de variables de entorno o un gestor de secretos.

Logging, auditoría y evidencia: Se implementarán *logs* de auditoría para registrar acciones críticas, como la creación o eliminación de usuarios.

## Requisitos legales y normativos:

**Legislación colombiana relevante:**

* **Ley 1581 de 2012: Protección de datos personales.**
* **Decreto 1377 de 2013: Reglamentación parcial de la Ley 1581.**
* **Resolución 000042 de 2020 (DIAN): Requisitos de facturación electrónica.**

**Normas y marcos internacionales relevantes:**

* **ISO/IEC 27001: Sistema de gestión de seguridad de la información.**
* **PCI DSS: Si se procesan directamente datos de tarjetas de crédito.**

# GESTIÓN DE DATOS

Modelo de datos y esquemas principales: Se adjunta el diagrama ER completo en los anexos.

Estrategia de persistencia: Se utilizará MySQL para el almacenamiento relacional.

Políticas de retención, purga y anonimización: Los datos se conservarán según las normativas legales y se implementarán procesos de purga para datos antiguos.

Backup, restore y verificación: Se realizarán *backups* automáticos diarios de la base de datos.

Estrategia de Migración de Datos: Se usará Alembic para gestionar los cambios en el esquema de la base de datos de forma incremental y reproducible.

# ESCALABILIDAD Y DISPONIBILIDAD

Estrategia de escalado: El sistema está diseñado para escalado horizontal a nivel de contenedores (*stateless*).

Balanceo de carga: Se usará un balanceador de carga para distribuir el tráfico entre múltiples instancias del *backend*.

Alta disponibilidad: El sistema se desplegará en múltiples zonas de disponibilidad en la nube para garantizar la continuidad del servicio.

# RESILIENCIA Y RECUPERACIÓN ANTE DESASTRES

Patrones de tolerancia a fallos

Se aplicarán patrones como *Circuit Breaker* para manejar fallos de servicios externos.

Estrategia de DR:

Se definirá un runbook con los pasos para restaurar el servicio en caso de un desastre.

# OBSERVABILIDAD Y MONITOREO

Logs:

**Los *logs* se centralizarán en un sistema de gestión de *logs* para facilitar el monitoreo y la depuración.**

Métricas:

**Se monitorearán métricas clave como la latencia, el uso de CPU y RAM, y la tasa de errores de la API.**

Tracing distribuido:

**Se implementará OpenTelemetry para rastrear las peticiones a través de los microservicios.**

Dashboards y alertas:

**Se crearán *dashboards* con Grafana o similar para visualizar métricas y se configurarán alertas para notificar al equipo sobre posibles problemas.**

# OPERACIONES Y DESPLIEGUE (CI/CD)

Entornos:

**Se mantendrán entornos separados para desarrollo, pruebas (*staging*) y producción.**

Pipelines CI/CD:

**Se usará GitHub Actions o GitLab CI para automatizar las pruebas, el *linting* y el despliegue del código.**

Herramientas de provisioning:

**Se utilizarán *templates* de Terraform y docker-compose.yml para gestionar la infraestructura.**

# MANTENIBILIDAD Y EVOLUCIÓN

Estrategia de versionado:

**Se utilizará un versionado semántico (MAJOR.MINOR.PATCH).**

Prácticas de calidad:

**Se aplicarán *linters* y análisis estático en el *pipeline* de CI.**

# ESTRATEGIA DE PRUEBAS Y CALIDAD

Tipos de pruebas: **Pruebas unitarias, de integración y de extremo a extremo.**

Cobertura mínima: **Se buscará una cobertura de código mínima del 80% en las pruebas unitarias.**

# COSTOS Y PROYECCIONES

Estimación de costos: **Se hará un cálculo de costos basado en el uso de los servicios de la nube (instancias EC2, bases de datos RDS, etc.).**

# CUMPLIMIENTO LEGAL Y PROTECCIÓN DE DATOS

Políticas: **Se incluirá una política de privacidad detallada y los formularios de consentimiento requeridos por la Ley 1581.**

Requisitos legales: **Se garantizará el cumplimiento de las leyes colombianas y los estándares internacionales de protección de datos.**

# ANEXOS

Glosario de términos: **(Pendiente de completar)**

Lista de abreviaturas: **(Pendiente de completar)**

Enlaces a repositorios, diagramas fuente y scripts: **(Pendiente de completar)**

Plantillas útiles: **(Pendiente de completar)**

# APÉNDICES TÉCNICOS

PlantUML / Diagrams.net sources: **Se incluirán los archivos .puml para que los diagramas puedan ser editados.**

Ejemplos de configuración: **Ejemplos de archivos nginx.conf, docker-compose.yml y .env.example.**

Scripts de provisión**: Scripts para la configuración inicial del entorno.**