Tabla de Contenido

1 Generalidades del documento

1.1 Objetivo del documento

El objetivo de este documento es establecer tanto la arquitectura como el diseño detallado del sistema en elaboración durante el proyecto para satisfacer los requisitos solicitados, de esta manera, se proporciona una base asentada verificable para realizar tanto la implementación como las pruebas del producto en cuestión.

Este documento define:

- La estructura modular
- El modelo de datos lógico
- Interfaces UI/UX
- Flujos y estados claves
- Decisiones de diseño
- Diagramas lógicos y estructurales

Adicionalmente, busca que se aseguren los principales atributos de calidad discutidos en documentos previos (seguridad, mantenimiento, mantenibilidad, etc) para mantener la trazabilidad con los requerimientos elaborados y de la misma forma, habilitar inspecciones en el marco TSP para prevenir defectos de manera temprana

1.2 Audiencia del documento

este documento está dirigido a:

- Patrocinador o product owner: Toma las decisiones del alcance y priorización
- Equipo de arquitectura: define los lineamientos técnicos y vela por los atributos de calidad
- Equipo de desarrollo (frontend/backend): implementa componentes conformes al diseño
- Equipo de pruebas:diseña y ejecuta pruebas alineadas con la arquitectura y los casos de uso.
- soporte y atención al usuario: comprende los flujos y restricciones de los usuarios para resolver incidencias

1.3 Acrónimos

TIC: Tecnologías de la Información y la Comunicación
API: Interfaz de programación de Aplicaciones

UML: Lenguaje Unificado de Modelado
HTML: Lenguaje de Marcado de Hipertexto
SQL: Lenguaje de COnsulta Estructurada
DNS: Sistema de Nombres de Dominio
SSL/TLS: Capa de Sockets Seguros / Transport Layer Security
IP: Protocolo de Internet
JSON: JavaScript Object Notation
JWT: JSON Web Token
HTTP: Hypertext Transfer Protocol
CRUD: Create, Read, Update and Delete.
ID: Identificador

2 Introducción

2.1 Descripción del problema

Actualmente muchas personas no cuentan con una herramienta que centralice los servicios básicos necesarios de documentación básica de tránsito en Colombia como el SOAT y la tecnomecanica , por lo cual los usuarios están expuestos a plataformas fraudulentas que promocionan estos servicios, pero en realidad, son personas que crean estas plataformas para estafar a los usuarios. Con el fin de ayudar a los usuarios a tramitar sus documentos y a su vez obtener un beneficio como empresa, se crea la aplicación Smart Traffic, una plataforma que brinda estos servicios a los usuarios actuando como un intermediario asegurando veracidad y calidad en la prestación de sus servicios.

Smart Traffic hace uso de HTTPS/443 y PostgreSQL vía JDBC/5432 y dispondrá de las siguientes funcionalidades. RF-01 gestiona registro/login con verificación por correo, BCrypt y JWT; RF-02 inscribe a cursos con control de cupos;RF-03 recomienda oficina de tecnomecánica por distancia/ETA; RF-04 procesa pagos con pasarela y webhooks; RF-05 expone historial de pagos; RF-06 permite CRUD de cursos; RF-07 entrega estadísticas para rol administrador ; RF-08 envía recordatorios de documentos vencidos vía SMTP/API; RF-09 permite recuperar contraseña con token.

2.2 Objetivos de la aplicación

- Desarrollar una plataforma web intermediaria que asegure un servicio de calidad entre empresas prestadoras de servicios automotrices y clientes, permitiendo la gestión ágil y segura de trámites como SOAT, tecnomecánica y cursos de conducción.
- Destacar el sector automotriz mediante la promoción de empresas aliadas, reconocidas por su gran servicio de calidad, respaldadas por una evaluación exhaustiva que lo certifican.
- Unificar en un portal de manejo intuitivo la gestión y agendamiento de los distintos servicios de documentación de tránsito que un usuario pueda requerir, priorizando la seguridad y la rapidez para su uso
- Realizar la plataforma web en un periodo no mayor a 3 meses y medio.
- Aplicar correctamente la Ley 1581 de Habeas Data junto con las normas ISO 27000 y 27001.
- Soportar más de 1000 usuarios simultáneamente sin afectar su rendimiento.
- Redirigir aproximadamente 1000 clientes mensuales a negocios aliados.

2.3 Restricciones

- Tiempo: entrega funcional en 3,5 meses con alcance priorizado
- tecnología:
    - backend en java más Spring Boot; frontend en React
    - PostgreSQL, comunicaciones HTTP/REST, autenticacion JWT
- Cumplimiento normativo: Ley 1581 de 2012 (habeas data) y lineamiento ISO/IEC 27000/27001
- Integraciones: dependencias de terceros (Proveedor de mapas)
- seguridad: todo tráfico externo HTTPS/TLS 1.2, protección de datos personales, no almacenar datos sensibles de tarjeta
- Rendimiento y capacidad:
    - ≥1000 usuarios concurrentes sin degradación perceptible.
    - Tiempo objetivo: 500ms para operaciones de lectura, <1200 ms en operaciones de terceros
- Disponibilidad: objetivo operativo ≥99,5% mensual para capa de aplicación

- Alcance fuera de proyecto: no se implementa agenda académica para cursos, no se contemplan microservicios en esta fase
- Datos y tarifas: Precios de servicios persistidos por registro para trazabilidad frente a cambios

3 Arquitectura de software

3.1 Estilo de arquitectura

Frente al diseño seleccionado para el proyecto, se ha optado por una arquitectura monolítica estructurada por capas, basada en el modelo MVC (Modelo–Vista–Controlador) y complementada con las capas de servicios y repositorios. Esta estructura organiza el sistema en componentes con responsabilidades claramente definidas: los controladores gestionan las peticiones HTTP y exponen las funcionalidades mediante API REST, los servicios encapsulan la lógica de negocio y los repositorios se encargan del acceso y la persistencia de los datos.

El uso de API REST permite una comunicación estandarizada, desacoplando la capa de presentación del cliente respecto a la lógica del servidor, lo cual facilita la integración con otros sistemas y posibles clientes externos. Este diseño prioriza la mantenibilidad, la claridad del código y la extensibilidad, brindando además una base sólida para una futura evolución hacia un modelo modular o de microservicios, según las necesidades de escalabilidad o distribución del sistema.

La lógica de negocio es cohesionada y beneficiada por medio de las transacciones locales y la estructura hexagonal permite aislar las integraciones volátiles, como la pasarela de pagos, la cual fue integrada con MercadoPago, integrando NGROK para la creación de tuberías seguras

Dentro de la estructura lógica, se tienen en cuenta las siguientes consideraciones:

- Un núcleo de dominio con entidades y reglas de pago para la lógica del negocio
- Los casos de uso son quienes orquestan las transacciones y las políticas internas de la aplicación
- Los puertos de entrada corresponden a los REST Controllers que exponen los endpoints para su correspondiente consumo
- Los puertos de salida corresponden a las interfaces con el repositorio de PostgreSQL, la pasarela de pagos, y la API de mapas integrada con el FrontEnd
- Los adaptadores de la arquitectura corresponden a las implementaciones del desarrollo concretas de los puertos previamente mencionados

Para las políticas técnicas de los flujos, se van a priorizar los siguientes aspectos:

- Idempotencia para los pagos para tener una transaccionalidad económica segura
- Una estrategia de circuit breaker/retry frente PSP (pasarela de pagos)
- Transacciones locales para efectuar los cambios en la información de la base de datos

Los atributos de calidad que serán abordados y sus respectivas tácticas serán los siguientes:

- Mantenibilidad: Mediante este modelo de arquitectura, el dominio se mantiene aislado de los adaptadores
- Seguridad: Por medio de JWT, va a darse una verificación a la sesión de cada usuario para asegurar la integridad de la información
- Disponibilidad: Por medio de reintentos, timeouts y los previamente mencionados circuits breaker/retry
- Escalabilidad: La estructura de monolito basado en MVCpermite una escala horizontal del sistema
- Observabilidad: Se pueden evidenciar los logs y registros relacionados a partir de la información de la base de datos

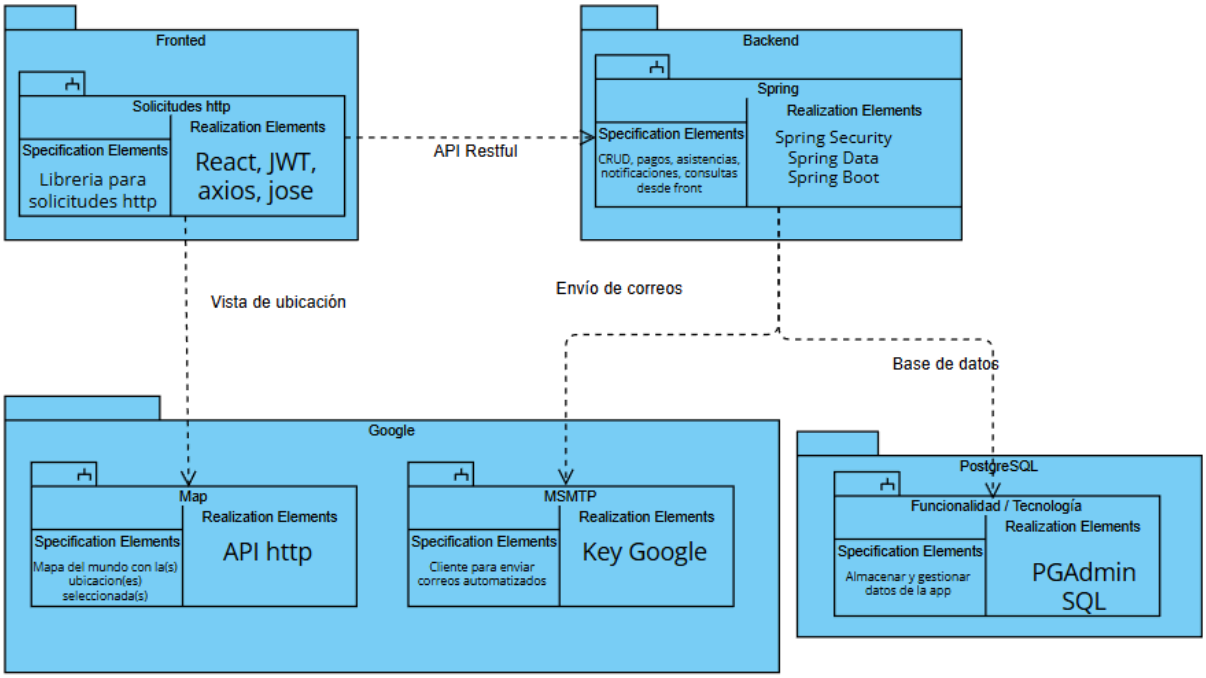
Sin embargo, hay que entrar en las consideraciones de los riesgos y dependencias del sistema, que consisten en la intermitencia de la pasarela de pagos, el crecimiento del tráfico y los cambios de tarifa. Por ende, las soluciones, correspondientes, manejan la reconciliación de pagos con los reintentos de las solicitudes, un escalado horizontal y el modelado de la base de datos donde se persiste el precio a pagar con respecto a un precio establecido.

Para finalizar, las decisiones y restricciones tomadas para abordar el desarrollo y diseño del proyecto son:

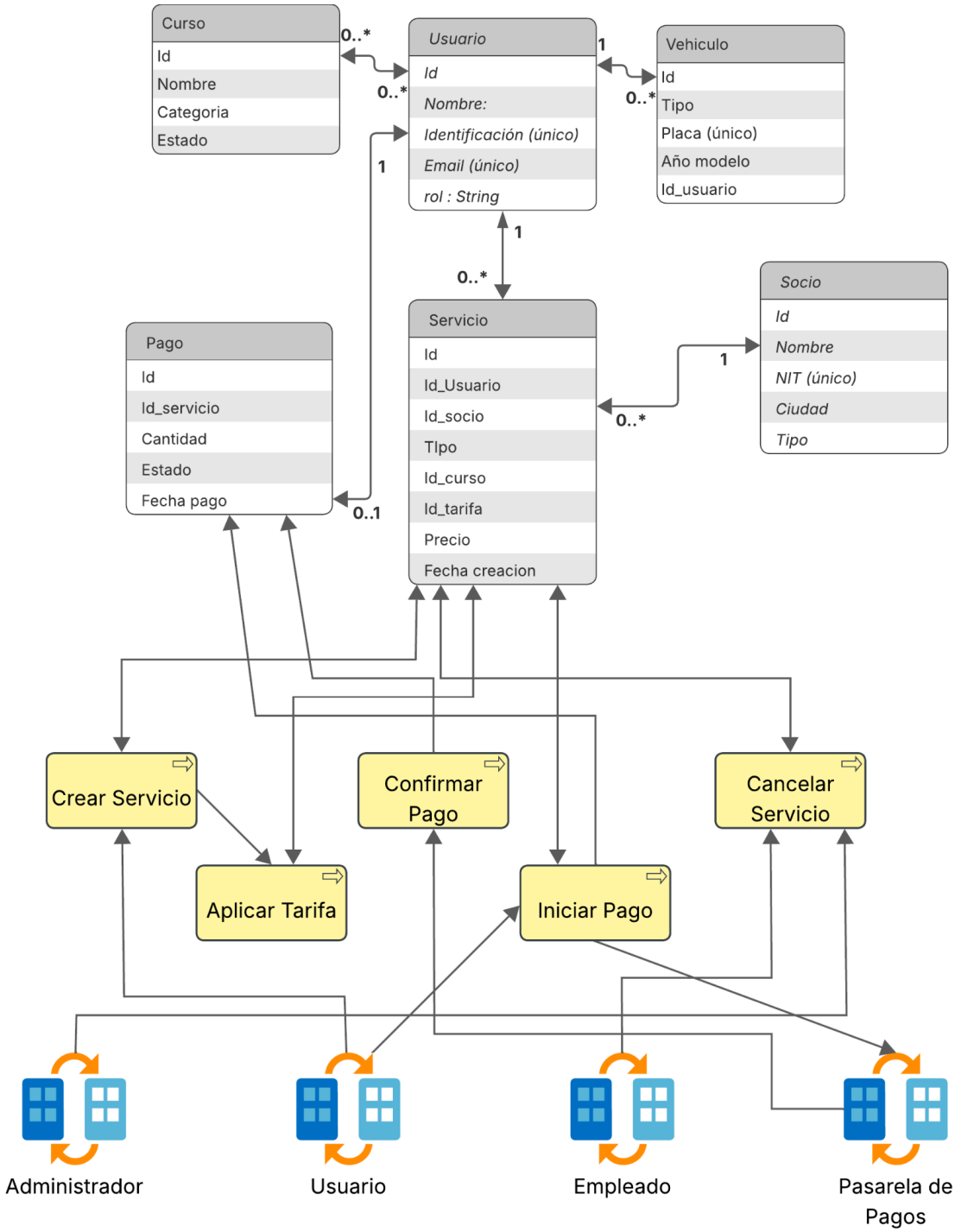
a) Un monolito modular basado en MVC
b) Base de datos con PostgreSQL
c) Las integraciones y la transaccionalidad a través de vía HTTP/REST
d) Para este segundo ciclo, se implementó una lógica de negocio para encolar cursos de conducción y su correspondiente publicación y activación cuando se cumpla el número mínimo de interesados

3.2 Vistas de arquitectura de arquitectura

-Desarrollo

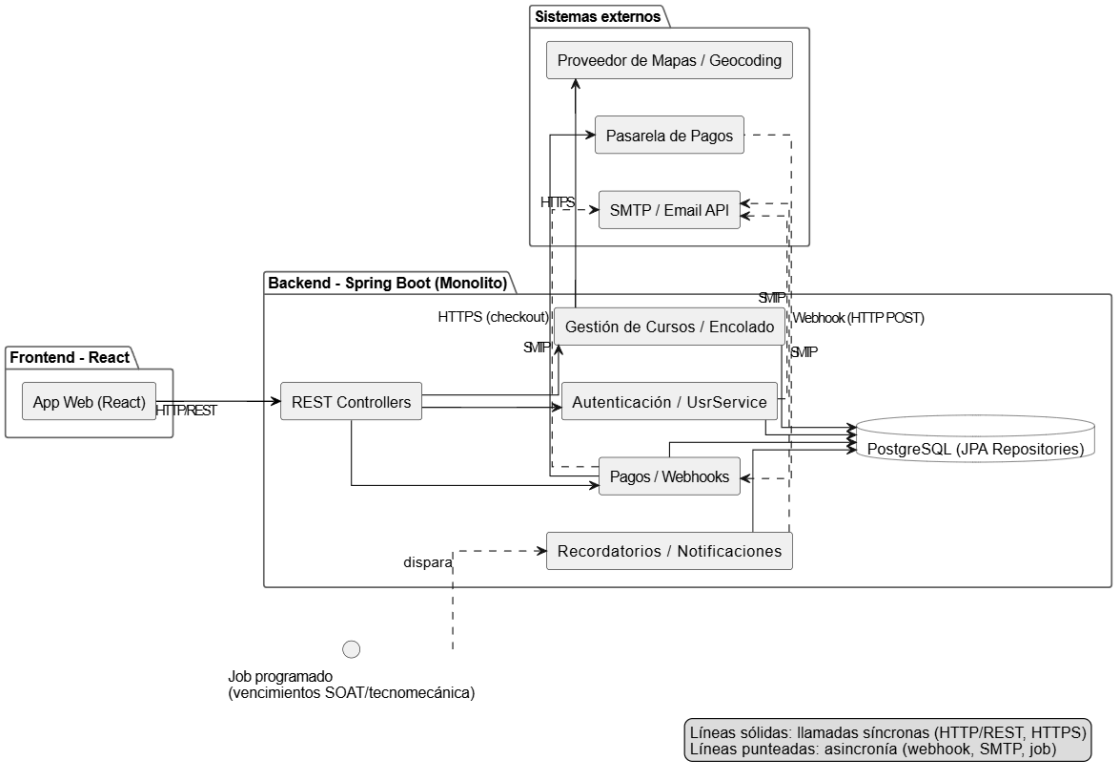


-Información

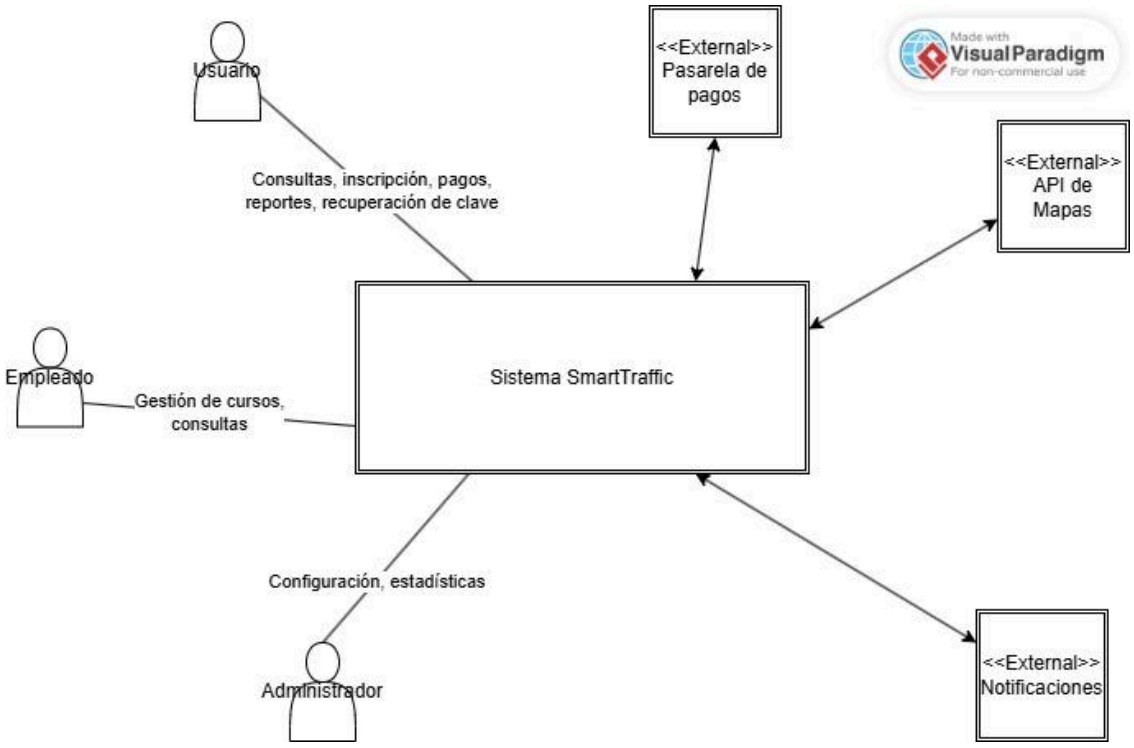

Curso
Id
Nombre
Categoria
Estado
Usuario
Id
Nombre:
Identificación (único)
Email (único)
rol : String
Vehiculo
Id
Tipo
Placa (único)
Año modelo
Id_usuario
Socio
Id
Nombre
NIT (único)
Ciudad
Tipo
Servicio
Id
Id_Usuario
Id_socio
TIpo
Id_curso
Id_tarifa
Precio
Fecha creacion
Pago
Id
Id_servicio
Cantidad
Estado
Fecha pago
0..*
1
0..1
Crear Servicio
Confirmar Pago
Cancelar Servicio
Aplicar Tarifa
Iniciar Pago
Administrador
Usuario
Empleado
Pasarela de Pagos

El siguiente diagrama integra lo que es el modelo de información con el ciclo de vida de los datos. Las cajas grises representan las entidades y sus conexiones con multiplicidad indica la cardinalidad entre ellas mismas para entender las cantidades del negocio.
Las cajas azules refieren a los actores involucrados y las cajas amarillas refieren a las tareas y procesos, dándose un enfoque al tema de los servicios principalmente (se ignoran temas como creación de usuario y demás debido a que la mayor transaccionalidad gira en torno al tema de servicios). La idea del diagrama muestra el flujo de acciones donde se reciben, pactan y/o eliminan datos o registros frente a las entidades, las acciones y los actores que participan en el proceso
Por ejemplo, **Crear Servicio** recibe datos desde **Usuario** los datos del curso y persiste la orden en *Servicio*, **Aplicar Tarifa** lee *Tarifa* y fija el precio pactado en *Servicio*, y así sucesivamente para las demás acciones que refieren en el diagrama.

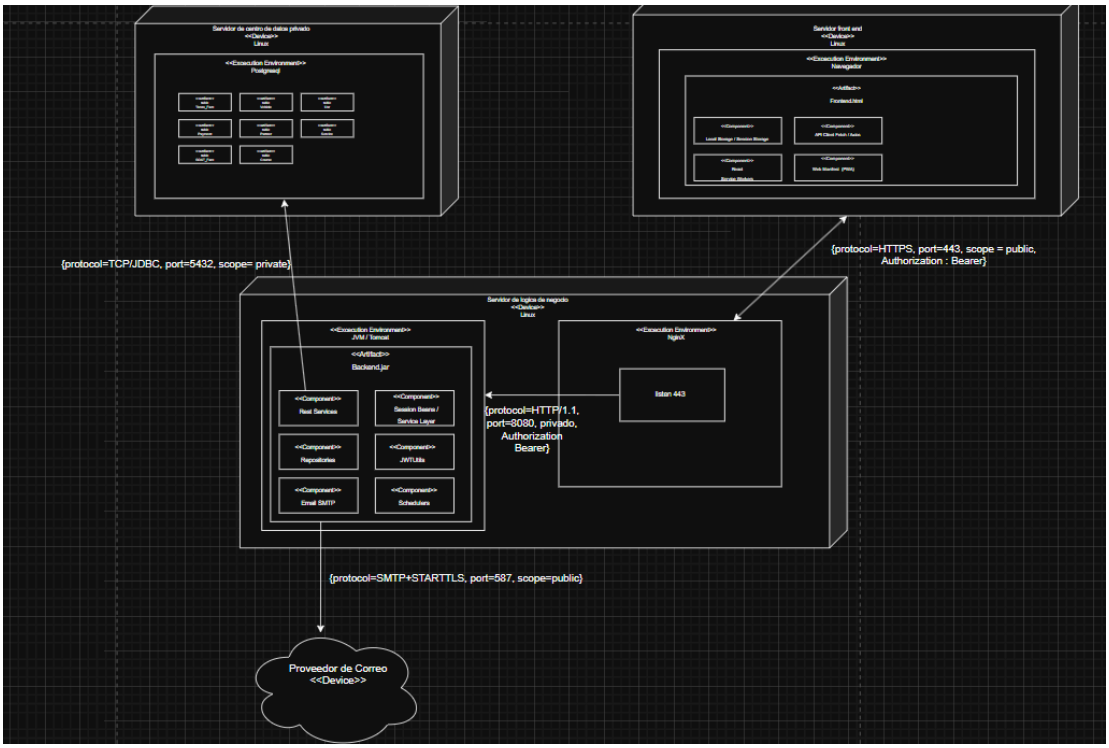
-Funcional



-Contexto



-Despliegue



https://app.diagrams.net/#G1arons33pHVYpr-TgPInmcjGQjTphy5hJ#%7B%22pageId%22%3A%226bm_DVs9BIqvOxrU7L4E%22%7D

3.3 Decisiones de arquitectura

Continuando con la estrategia propuesta anteriormente (Definición de Estrategia) de basarse principalmente en tres pilares: modularización, asignación clara de responsabilidades y dividir y conquistar, se integra de nuevo bajo el contexto de diseño. En este caso, se hace énfasis en la modularización con funciones debidamente repartidas para el bajo acoplamiento y alta cohesión, además de dividir y conquistar, en donde se separan las responsabilidades de cada componente para lograr ofrecer un producto correctamente distribuido así como se muestra en las vistas de arquitectura. Al verse distribuido cada componente se hace referencia a empaquetar o agrupar las funcionalidades en un mismo rubro con el fin de tener la dependencia necesaria a las funciones que así se requiera, obteniendo así una solución óptima, robusta, escalable y viable.

4 Diseño detallado

4.1 Decisiones de diseño

Problema/ Preocupación de diseño	**Estilo arquitectónico elegido** Se requiere de una arquitectura bien definida para mantener unida la cohesión de la lógica de negocio, aislar aquellas dependencias externas y facilitar la evolución del propio sistema hacia modelos más distribuidos sin comprometer la calidad de las transacciones. Deben de separarse correctamente las responsabilidades y que la comunicación entre los distintos componentes sea de manera controlada
Orientación de la decisión	Se adopta una arquitectura monolítica estructurada en capas, basado en el modelo MVC (Modelo-Vista-Controlador) y complementada con distintas capas de servicios y repositorios. Este diseño está apoyado en principios de arquitectura hexagonal para aislar los componentes volátiles y así, poder asegurar la independencia en el dominio y las integraciones externas. A continuación se hacen unas determinadas especificaciones: - Los controladores REST actúan como los puertos de entrada, recibiendo y exponiendo las funcionalidades del sistema - Los servicios representan los casos de uso, donde se orquesta la lógica de negocio y las reglas transaccionales - Los repositorios y adaptadores (DTOs) funcionan como los puertos de salida, encargándose de persistir los datos guardados en PostgreSQL, la integración con la pasarela de pagos y la comunicación con la API de mapas en el frontend. Estas orientaciones permiten desarrollar un código mantenible, modular y con una óptima proyección futura
Supuestos	Se contemplan los siguientes supuestos frente a este problema de diseño: - El sistema operará con transacciones locales sobre una base de datos relacional centralizada - Las integraciones externas (pasarela de pagos) funcionarán mediante servicios HTTP/REST bajo tuberías seguras, por ejemplo, bajo canales NGROK durante las pruebas

	- La lógica de negocio está netamente centralizada en el backend y el frontend va a actuar únicamente con el consumidor, siendo la vista del mismo - El equipo tiene conocimiento en las tecnologías propuestas para el desarrollo
Restricciones	Se contemplan las siguientes restricciones frente a este problema de diseño: - La arquitectura debe de mantenerse monolítica durante el ciclo, sin despliegues distribuidos ni ninguna decisión que altere su estructura - La base de datos principal es PostgreSQL, por ende, deben evitarse dependencias de motores NoSQL - En este ciclo se levanta la restricción de la lógica de encolar cursos para su correspondiente inscripción - Toda comunicación entre los componentes debe de realizarse mediante API REST con autenticación JWT
Requerimientos relacionados	RF-01: Registro e inicio de sesión por parte de todos los usuarios. RF-02: Inscripción de usuarios a cursos de conducción. RF-04: Recomendación inteligente de oficina de trámites cercana (Tecnomecánica). RF-05: Pasarela de pagos para servicios de la aplicación (Tecnomecánica, cursos de conducción y Soat). RF-06: Historial de Pagos. RF-07: Gestión de cursos de conducción y comparendos. RF-08: Creación de interfaces por rol (Administrador, Empleado y Cliente). RF-09: Estadísticas para la interfaz de rol Administrador. RF-10: Notificar al usuario cuando sus documentos de tránsito estén vencidos. RF-11: Recuperar contraseña
Alternativas consideradas	**Alternativa 1:** Monolito modular basado en MVC con principios hexagonales **Alternativa 2:** Arquitectura de microservicios independientes desde la primera versión **Alternativa 3:** Arquitectura basada en eventos con implementación de colas intermedias

Evaluación de alternativas	**Alternativa 1:** Monolito modular basado en MVC con principios hexagonales - Ventajas: Facilita el mantenimiento, reduce la complejidad inicial y permite evolucionar modelos distribuidos de manera gradual - Desventajas: Puede presentarse una escalabilidad limitada si se presenta un incremento masivo en el tráfico del servicio y número de usuarios **Alternativa 2:** Arquitectura de microservicios independientes desde la primera versión - Ventajas: Alta escalabilidad y despliegue independiente - Desventajas: Requiere una mayor infraestructura, sincronización compleja y aumento en las fases tempranas frente a sobrecargas operativas innecesarias **Alternativa 3:** Arquitectura basada en eventos con implementación de colas intermedias - Ventajas: Excelente desacoplamiento y resiliencia - Desventajas: Presenta una alta complejidad técnica y una elevada curva de aprendizaje del equipo
Alternativa seleccionada y Justificación	Se selecciona la Alternativa 1: monolito modular basado en MVC con principios de arquitectura hexagonal. Esta elección equilibra simplicidad, mantenibilidad y preparación para la evolución futura. Permite encapsular la lógica de negocio, aislar integraciones volátiles (como MercadoPago) y conservar una estructura clara de capas. Además, su uso con Spring Boot ofrece soporte nativo a API REST y transacciones locales, lo que garantiza una comunicación segura y una base sólida para un crecimiento posterior.
Decisiones relacionadas	La arquitectura previamente mencionada es la que se maneja desde el ciclo I, por ende, no se realizaron decisiones post-modelo del estilo arquitectónico para el proyecto
Comentarios	El equipo entiende la arquitectura propuesta y hasta el momento no se han presentado inconvenientes o alteraciones con la misma

Problema/ Preocupación de diseño	Lenguajes y frameworks principales: Definir lenguajes de programación y frameworks que permitan la integración eficiente del sistema SmartTraffic, garantizando mantenibilidad, escalabilidad y cumplimiento de los requisitos funcionales y no funcionales definidos.
Orientación de la decisión	Seleccionar un entorno de desarrollo robusto y con amplio soporte de comunidad que facilite la creación de servicios REST, la persistencia de datos y el consumo desde una interfaz web moderna.
Supuestos	● El equipo cuenta con experiencia previa en desarrollo con Java y React. ● Se requiere compatibilidad con PostgreSQL y soporte para API REST. ● La arquitectura base del proyecto es monolítica en capas con patrón MVC.
Restricciones	● Uso obligatorio de tecnologías open source. ● Cumplimiento de lineamientos institucionales para desarrollo web. ● Restricción de tiempo (3.5 meses) que limita la adopción de tecnologías experimentales.
Requerimientos relacionados	RF-01: Registro e inicio de sesión por parte de todos los usuarios. RF-02: Inscripción de usuarios a cursos de conducción. RF-04: Recomendación inteligente de oficina de trámites cercana (Tecnomecánica). RF-05: Pasarela de pagos para servicios de la aplicación (Tecnomecánica, cursos de conducción y Soat). RF-06: Historial de Pagos. RF-07: Gestión de cursos de conducción y comparendos. RF-08: Creación de interfaces por rol (Administrador, Empleado y Cliente). RF-09: Estadísticas para la interfaz de rol Administrador.

	RF-10: Notificar al usuario cuando sus documentos de tránsito estén vencidos. RF-11: Recuperar contraseña.
Alternativas consideradas	**Alternativa 1** Spring Boot + React **Alternativa 2** Node.js + Vue **Alternativa 3** Django + React
Evaluación de alternativas	**Alternativa 1** Spring Boot + React: Ventajas: robustez, modularidad, integración con JPA, excelente manejo de seguridad y soporte REST. Desventajas: curva de aprendizaje moderada y mayor uso de memoria. **Alternativa 2** Node.js + Vue: Ventajas: desarrollo ágil y unificado en JavaScript. Desventajas: menor robustez en entornos de producción empresarial. **Alternativa 3** Django + React:

	Ventajas: rapidez de desarrollo, ORM integrado. Desventajas: menor experiencia del equipo y sobrecarga de configuración para API REST.
Alternativa seleccionada y Justificación	Se selecciona Java con Spring Boot para el backend, debido a su estructura modular, integración nativa con JPA, herramientas de seguridad (Spring Security y JWT) y compatibilidad con PostgreSQL. En el frontend se adopta React por su flexibilidad, modularidad de componentes y amplio ecosistema. Esta combinación permite un flujo cliente-servidor optimizado bajo el estándar RESTful.
Decisiones relacionadas	• Estilo arquitectónico (Monolito modular MVC). • Mecanismo de persistencia (PostgreSQL con JPA). • Gestión de comunicación e integración(REST)
Comentarios	Sin ningún comentario adicional.
Problema/ Preocupación de diseño	Mecanismos de persistencia y base de datos Definir un mecanismo de almacenamiento que garantice consistencia transaccional, trazabilidad, rendimiento, seguridad y simplicidad operativa dentro de un monolito Spring Boot, integrando pagos vía webhooks y consultas externas (mapas)
Orientación de la decisión	• Motor y acceso: PostgreSQL más Spring Data JPA/Hibernate • transacciones: locales con límites por caso de uso, aislamiento por defecto y retry cuando aplique • Concurrencia: optimista con versión y bloqueo pesimista solo en puntos críticos • Idempotencia pagos: clave única sobre las referencias externas más estado; registro de eventos

	● Trazabilidad: persistir precio en cada orden/servicio y referencia a tarifa vigente ● respaldos/recuperación: respaldos automáticos diarios, retención ≥ 30 días, procedimientos de restauracion probados para cunmplir RTO < 30 min y RPO < 15 min
Supuestos	● El equipo domina JPA/Hibernate ● Las integraciones externas son vía HTTP/REST y webhooks ● El frontend solo consume APIS, la lógica de negocio reside en el backend
Restricciones	● Mantener monolito (no microservicios en este ciclo) ● Evitar dependencias NoSQL ● Cumplimiento LEY 1581/2012 e ISO/IEC 27001: No almacenar datos sensibles de tarjeta ● Objetivos de rendimiento: p95 de lecturas ≤ 500 ms, operaciones con terceros < 1200 ms y disponibilidad objetivo ≥ 99,5% mensual;
Requerimientos relacionados	● RF-02 (inscripción de cursos) ● RF-04 (Pagos + webhooks) ● RF-05 (Historial de pagos) ● RF-06 (CRUD de cursos) ● RF-07 - RF-09 (Estadísticas) ● RF-10 (recordatorios de vencimientos) ● RF-11 (Recuperación de contraseña)
Alternativas consideradas	**Alternativa 1** - PostgreSQL + JPA/hibernate (enfoque estándar Spring) **Alternativa 2** - PostgreSQL + Mybatis (mapeo SQL explícito) + Hikari CP **Alternativa 3 -** PostgreSQL + Outbox/Eventos + cola para consistencia

Evaluación de alternativas	**Alternativa 1** ● Ventajas: Alta productividad, integración nativa con Spring, transacciones locales, migraciones controladas, curva de aprendizaje favorable ● Desventajas - Exige modelado cuidadoso para alto rendimiento **Alternativa 2** ● Ventajas: Control fino del SQL y planes de ejercicio en consultas complejas ● Desventajas: Menos velocidad de entrega y mayor costo de mantenimiento **Alternativa 3** ● Ventajas: Resiliencia y desacoplamiento elevados para integraciones intersivas ● Desventajas: Complejidad operativa innecesaria en este ciclo
Alternativa seleccionada y Justificación	Se selecciona la alternativa 1 por ofrecer el mejor equilibrio entre simplicidad, velocidad de desarrollo, consistencia e integración nativa con Spring Boot, permitiendo cumplir trazabilidad, idempotencia en pagos y objetivos de rendiciempo sin sobrecargar la operación
Decisiones relacionadas	● estilo: Monolito modular MVC ● integración: Spring security + JWT, contraseñas BCrypt, HTTPS end to end ● No funcionales: disponibilidad y métricas de desempeño
Comentarios	A futuro, se prevé la implementación de una réplica asíncrona de base de datos para mejorar el rendimiento en consultas de lectura y utilización de vistas materializadas para optimizar el cálculo de estadísticas. Asimismo, se considera viable la adopción del patrón Outbox en ciclos posteriores si el sistema evoluciona hacia una arquitectura distribuida o requiere mayor integración con servicios externos

Problema/ Preocupación de diseño	Estrategia de identificadores y claves de negocios (IDs, referencias externas y correlación) Se requiere un modelo consistente para identificar entidades internas, garantizar unicidad claves de negocio, correlacionar interacciones con terceros (pasarela de pagos) y evitar colisiones o dependencias del autoincremento. Además, se necesitan IDs seguros para exposición pública, soporte de idempotencia y buen desempeño en búsquedas y joins
Orientación de la decisión	Se adopta una estrategia basada en UUID v7 como identificador principal para todas las entidades, acompañadas de claves naturales únicas según el dominio y campos de correlación para rastrear transacciones y garantizar independencia. Esta convención asegura independencia del motor, seguridad en la exposición de datos y consistencia en las relaciones entre módulos
Supuestos	● PostgreSQL ≥ 13 con soporte para funciones de generación de UUID ● Acceso mayoritario por PK, pero búsquedas frecuentes ● El PSP siempre envía una referencia única por transacción
Restricciones	● Despliegue local (Sin generadores distribuidos) ● No se expondrán claves naturales sensibles como identificadores en la API ● Migraciones deberán introducir índices/constraints sin bloquear operaciones críticas
Requerimientos relacionados	● RF-04/RF-05: idempotencia y colisión ● RF-06/RF-07/RF-09: Búsquedas por claves de negocio ● No funcionales: Seguridad, mantenibilidad, rendimiento en lookups

Alternativas consideradas	**Alternativa 1** - UUID v7 como PK + claves naturales con referencias externas y correlaciones **Alternativa 2** - BIGSERIAL como PK + claves naturales únicas, UUID solo para exposición **Alternativa 3 -** Generadores distribuidos tipo Snowflake
Evaluación de alternativas	**Alternativa 1** ● Ventajas: Ordenable temporalmente, seguro para exposición, desacopla de la DB, ideal para correlación ● Desventajas: PK más ancha que BIGSERIAL **Alternativa 2** ● Ventajas: simple y compacto, rendimiento muy bueno en PK ● Desventajas: exposición problemática, acoplamiento a la DB, colisiones entre entornos al replicar **Alternativa 3** ● Ventajas: alta escalabilidad y orden temporal ● Desventajas: complejidad innecesarias en despliegue local, requiere coordinar workers
Alternativa seleccionada y Justificación	se elige la alternativa 1 por ofrecer trazabilidad robusta, seguridad al exponer IDs, idempotencia clara con el PSP y buen ordenamiento para índices y consultas temporales, sin depender de infraestructura adicional
Decisiones relacionadas	● mecanismo de persistencia: se apoya en PostgreSQL/JPA sin re-definir ORM ● Auditoría/observabilidad: unifica logs/metricas ● Rendimiento: índices compuestos en referencias externas, emails

Comentarios	● Añadir test de integración que verifiquen unicidad y idempotencia al recibir webhook ● Documentar README la política de IDs y ejemplos de URLs ● en migraciones, planificar la transición gradual si alguna tabla ya se usa
Problema/ Preocupación de diseño	Gestión de comunicación e integración : Definir el método de comunicación entre las aplicaciones de frontend y backend, para este proyecto se usará la REST como estilo arquitectónico para la comunicación
Orientación de la decisión	Creación de Servicios REST, estilo que todo el equipo domina a un nivel aceptable, los componentes que se verán afectados son todos los componentes que consuman servicios de nuestra aplicación backend como frontend, postman (pruebas), jmeter (pruebas)
Supuestos	● El equipo sabe cómo implementar servicios REST en la aplicación ● Todos los componentes de la aplicación utilizan este estilo para la comunicación. ● Se definen objetos con atributos no nulos para ser enviados usando JSON.
Restricciones	● Transacciones que no sean nativas como por ejemplo la pasarela de pagos. ● No es funcional, si se llega a necesitar actualizaciones en tiempo real
Requerimientos relacionados	RF-01: Registro e inicio de sesión por parte de todos los usuarios. RF-02: Inscripción de usuarios a cursos de conducción. RF-04: Recomendación inteligente de oficina de trámites cercana (Tecnomecánica). RF-05: Pasarela de pagos para servicios de la aplicación (Tecnomecánica, cursos de conducción y Soat). RF-06: Historial de Pagos. RF-07: Gestión de cursos de conducción y comparendos. RF-08: Creación de interfaces por rol (Administrador, Empleado y Cliente). RF-09: Estadísticas para la interfaz de rol Administrador. RF-10: Notificar al usuario cuando sus documentos de tránsito estén vencidos. RF-11: Recuperar contraseña.

Alternativas consideradas	**Alternativa 1** REST **Alternativa 2** SOAP **Alternativa 3** GraphQL
Evaluación de alternativas	**Alternativa 1** REST Ventajas : Fácil de implementar, amplio soporte, uso eficiente de HTTP y flexible en formatos de envío de información. Desventajas: a veces recibe más o menos información de la necesaria, no tiene tipado, no es ideal para operaciones complejas, no soporta operaciones en tiempo real y difícil versionamiento en APIs grandes. **Alternativa 2** SOAP Ventajas : Garantiza estructura y tipos, soporta cifrado, firma digital, autenticación y tokens, soporta operaciones ACID a nivel empresarial, permite crear encabezados personalizados y está estandarizado para entornos empresariales Desventajas:los mensajes XML son pesados, más difícil de implementar y mantener que REST o GraphQL, menor compatibilidad con navegadores y apps móviles, dependencia de XML **Alternativa 3**

	GraphQL Ventajas : Enviar over/under - fetching, es decir, que el cliente solo pide exactamente los datos que necesita, optimiza el rendimiento, fuertemente tipado, ideal para frontends complejos. Desventajas: Requiere definir esquemas, difícil de cachear con HTTP, riesgo a consultas muy costosas, menor soporte nativo, no es ideal para operaciones simples
Alternativa seleccionada y Justificación	Para el proyecto se utilizará el estilo REST ya que consideramos que en el alcance del proyecto no se contemplan procesos tan exigentes, por lo tanto se podría usar este estilo cubre las necesidades del proyecto y es fácil de implementar considerando el tiempo de desarrollo del proyecto
Decisiones relacionadas	• Estilo arquitectónico (Monolito modular MVC). • Mecanismo de persistencia (PostgreSQL con JPA). • Seguridad
Comentarios	Sin comentarios adicionales.

Problema/ Preocupación de diseño	Seguridad, garantizar la protección de la información sensible y la autenticación segura de los usuarios dentro de la plataforma SmartTraffic, evitando accesos no autorizados y cumpliendo con la Ley 1581 de 2012 (Habeas Data).
Orientación de la decisión	Implementar un modelo de autenticación y autorización basado en JSON Web Tokens (JWT) gestionado desde Spring Security, complementado con encriptación BCrypt para contraseñas y tráfico.
Supuestos	● El sistema opera sobre HTTPS y no almacena información sensible de tarjetas. ● La autenticación debe ser sin estado para permitir escalabilidad. ● El backend ya está implementado sobre Spring Boot.
Restricciones	● Cumplimiento estricto de la Ley 1581 de 2012 e ISO 27001. ● No se permitirá persistir información de autenticación fuera del token. ● Los usuarios deben autenticarse mediante correo y contraseña validados.
Requerimientos relacionados	RF-01: Registro e inicio de sesión por parte de todos los usuarios. RF-11: Recuperar contraseña.
Alternativas consideradas	**Alternativa 1** JWT + BCrypt **Alternativa 2** OAuth2

Evaluación de alternativas	**Alternativa 1** JWT + BCrypt: Ventajas: escalable, seguro, compatible con API REST, sin estado. Desventajas: requiere manejo cuidadoso de expiración y renovación de tokens. **Alternativa 2** OAuth2: Ventajas: autenticación federada. Desventajas: complejidad adicional y dependencia de servicios externos.
Alternativa seleccionada y Justificación	Se adopta JWT + BCrypt, asegurando autenticación sin estado, verificación de roles y cifrado fuerte. Esta configuración mejora la seguridad de las rutas protegidas y evita vulnerabilidades comunes
Decisiones relacionadas	• Estilo de arquitectura (API RESTful). • Lenguajes y frameworks (Spring Boot).
Comentarios	La configuración permite extender la seguridad hacia futuras integraciones (OAuth2 o multifactor)
Problema/ Preocupación de diseño	**Despliegue y entornos**

	Definir el entorno de despliegue local adecuado para garantizar la disponibilidad, estabilidad y correcto funcionamiento de la aplicación SmartTraffic, asegurando compatibilidad con los componentes tecnológicos utilizados (Spring Boot, React, PostgreSQL) y posibilitando una entrega continua y segura en entornos de pruebas y producción.
Orientación de la decisión	El despliegue del sistema se realizará únicamente de forma local, manteniendo todos los componentes dentro del entorno de desarrollo del equipo. El backend (Spring Boot), el frontend (React) y la base de datos PostgreSQL se ejecutarán en la misma máquina o red local mediante configuraciones definidas en los archivos .env y application.properties. Las pruebas se realizarán sobre este entorno único, garantizando la correcta interacción entre módulos antes de cualquier futura publicación o migración a nube. Durante este ciclo, no se contemplan entornos de staging ni producción remota; el despliegue local facilita las pruebas controladas, reduce costos y simplifica la integración continua dentro del equipo de desarrollo.
Supuestos	● Todos los integrantes del equipo tienen instalado el entorno de desarrollo necesario: Java 17+, Node.js, PostgreSQL y Git. ● El archivo de configuración local (.env / .yaml) contiene las credenciales y rutas correctas para la conexión a la base de datos, conexión a backend y API de google maps. ● Las pruebas funcionales, unitarias y de integración se realizan dentro del mismo entorno local. ● Se cuenta con dos repositorios compartidos en GitHub que centraliza el código fuente y permite replicar fácilmente el entorno en cada máquina. Uno para el front y otro para el backend.
Restricciones	● No se realizará despliegue en servidores externos ni servicios en la nube debido al costo que este genera en adición con no tener algún patrocinador que financie el despliegue. ● Las pruebas de rendimiento estarán limitadas al hardware disponible en los equipos de desarrollo de cada uno de los integrantes del equipo. ● Los entornos locales deben mantener versiones unificadas de dependencias (Spring Boot, React, PostgreSQL).
Requerimientos relacionados	● RF-01: Registro e inicio de sesión de usuarios. ● RF-02: Inscripción a cursos de conducción. ● RF-05: Pasarela de pagos (validación en entorno local). ● RF-06: Historial de pagos. ● RF-07: Gestión de cursos de conducción y comparendos. ● RF-09: Estadísticas para el rol Administrador. ● RF-10: Recordatorios de documentos vencidos.

Alternativas consideradas	• Alternativa 1: Despliegue totalmente local en los equipos de desarrollo (Spring Boot + React + PostgreSQL). • Alternativa 2: Despliegue híbrido con base de datos remota y frontend local. • Alternativa 3: Despliegue en nube con instancias gratuitas (Render, Railway o Vercel).
Evaluación de alternativas	• Alternativa 1 – Local completo: Ventajas: control total del entorno, sin costos de infraestructura, mayor facilidad para depurar y probar. Desventajas: requiere configuración inicial manual en cada equipo. • Alternativa 2 – Híbrido: Ventajas: menor carga local en los equipos. Desventajas: dependencia de conexión a internet y configuración más compleja. • Alternativa 3 – Nube gratuita: Ventajas: accesibilidad remota. Desventajas: riesgo de interrupciones, límite de uso y mayor complejidad de configuración.
Alternativa seleccionada y Justificación	Se selecciona la Alternativa 1 (Despliegue local completo), ya que satisface las necesidades actuales del proyecto: bajo costo, simplicidad y control total del entorno. Permite ejecutar el backend, frontend y base de datos dentro del mismo sistema operativo sin depender de servicios externos. Además, facilita las pruebas, el mantenimiento y la detección temprana de errores durante el ciclo de desarrollo.
Decisiones relacionadas	• Estilo arquitectónico: Monolito modular MVC. • Lenguajes y frameworks: Spring Boot (Java) y React (TypeScript). • Mecanismo de persistencia: PostgreSQL con JPA. • Seguridad: Autenticación JWT sobre HTTPS local.
Comentarios	Se recomienda estandarizar las rutas, puertos y variables de entorno en un documento compartido para evitar inconsistencias entre máquinas. En ciclos futuros se podrá migrar el entorno a un servicio cloud (como AWS o Render) si el sistema requiere disponibilidad pública o escalabilidad.

Problema/ Preocupación de diseño	**Decisiones no funcionales clave.** Frente a la disponibilidad y la recuperación, el sistema debe de mantenerse disponible de forma continua para los usuarios y así, garantizar una recuperación oportuna ante fallas del software, errores humanos o incluso, pérdida de datos. Se propone minimizar el tiempo de inactividad (RTO) y la pérdida de información (RPO) sin incrementar de manera significativa la complejidad operativa
Orientación de la decisión	Se establecen objetivos de Recuperación ante Desastres (DR) y Continuidad del Servicio (BCP) con los siguientes valores de referencia: - RTO <30 minutos, donde el sistema debe restablecer su funcionamiento con normalidad en menos de media hora tras una determinada interrupción - RPO <15 minutos, donde los datos críticos no deben representar pérdidas mayores a los últimos 15 minutos de operaciones registradas
Supuestos	Se contemplan los siguientes supuestos frente a este problema de diseño: - El sistema debe de operar en un único servidor físico o máquina virtual, con capacidad para mantener una instancia secundaria sincronizada - El almacenamiento debe permitir conservar copias de respaldo al menos durante 30 días - Los respaldos y procedimientos de restauración deben ser probados de forma periódica para verificar su efectividad - El equipo cuenta con las credenciales de carácter administrativo al entorno de producción en caso de presentar una emergencia
Restricciones	Se contemplan las siguientes restricciones frente a este problema de diseño: - El proyecto no cuenta con presupuesto para infraestructura de carácter redundante - Las tareas de respaldo deben de ejecutarse sin afectar el rendimiento que los usuarios puedan percibir - Los procesos de recuperación deben poder realizarse durante una ventana de mantenimiento planificada corta

	- Los respaldos deben cumplir con las políticas de confidencialidad y manejo de datos establecidas por la organización
Requerimientos relacionados	RF-01: Registro e inicio de sesión por parte de todos los usuarios. RF-02: Inscripción de usuarios a cursos de conducción. RF-04: Recomendación inteligente de oficina de trámites cercana (Tecnomecánica). RF-05: Pasarela de pagos para servicios de la aplicación (Tecnomecánica, cursos de conducción y Soat). RF-06: Historial de Pagos. RF-07: Gestión de cursos de conducción y comparendos. RF-08: Creación de interfaces por rol (Administrador, Empleado y Cliente). RF-09: Estadísticas para la interfaz de rol Administrador. RF-10: Notificar al usuario cuando sus documentos de tránsito estén vencidos. RF-11: Recuperar contraseña
Alternativas consideradas	**Alternativa 1:** Copias de seguridad automáticas con réplica asíncrona y procedimientos de restauración documentados **Alternativa 2:** Configuración de un entorno de carácter activo.pasivo con failover automático **Alternativa 3:** Implementación de una infraestructura activo.activo distribuida en distintas regiones
Evaluación de alternativas	**Alternativa 1:** Copias de seguridad automáticas con réplica asíncrona y procedimientos de restauración documentados - Ventajas: Bajo costo de implementación, cumple con los objetivos de recuperación planteados, es fácil de mantener y escalar de manera gradual - Desventajas: Podría implicar una pequeña interrupción del servicio durante la restauración manual **Alternativa 2:** Configuración de un entorno de carácter activo.pasivo con failover automático - Ventajas: Reduce de manera significativa el tiempo de recuperación - Desventajas: Mayor complejidad operativa y consumo de recursos, requiere sincronización constante entre nodos

	Alternativa 3: Implementación de una infraestructura activo.activo distribuida en distintas regiones - Ventajas: Resiliencia de carácter máxima y cero pérdida de datos - Desventajas: Costos y requerimientos técnicos elevados, innecesarios para la escala actual del proyecto
Alternativa seleccionada y Justificación	Se adopta la Alternativa 1, basada en copias automáticas, réplica asíncrona y planes de restauración. Esta opción ofrece un balance adecuado entre disponibilidad, costo y simplicidad operativa, garantizando que el sistema pueda restablecerse rápidamente sin infraestructura compleja. La solución permite además evolucionar en el futuro hacia un modelo activo-pasivo si la carga o criticidad del sistema lo justifican.
Decisiones relacionadas	Temporalmente no se han tomado decisiones consecuentes a esta problemática. En la fase de implementación se discutirá de la viabilidad de esta estrategia
Comentarios	Debe de reunirse todo el equipo y analizar esta idea propuesta. Posterior a esa reunión, se decidirá cómo abarcar e implementar esta estrategia si realmente no representa alterar una gran cantidad del código base elaborado
Problema/ Preocupación de diseño	**Rendimiento:** asegurar tiempos de respuesta consistentes bajo concurrencia moderada y cargas pico, evitando cuellos de botella en API, base de datos y render del frontend
Orientación de la decisión	Optimizar servicios REST, teniendo un buen diseño de consultas.

Supuestos	- Tráfico concurrente moderado. - Volumen actual: decenas de miles de filas por identidad. - despliegue local. - Equipo conoce prácticas HikariCP,@Transactional,@EntityGraph, fetch lazy.
Restricciones	- Evitar dependencias complejas. - Mantener monolito modular, difícil de migrar - Memoria limitada
Requerimientos relacionados	RF-01: Registro e inicio de sesión por parte de todos los usuarios. RF-02: Inscripción de usuarios a cursos de conducción. RF-04: Recomendación inteligente de oficina de trámites cercana (Tecnomecánica). RF-05: Pasarela de pagos para servicios de la aplicación (Tecnomecánica, cursos de conducción y Soat). RF-06: Historial de Pagos. RF-07: Gestión de cursos de conducción y comparendos. RF-08: Creación de interfaces por rol (Administrador, Empleado y Cliente). RF-09: Estadísticas para la interfaz de rol Administrador. RF-10: Notificar al usuario cuando sus documentos de tránsito estén vencidos. RF-11: Recuperar contraseña
Alternativas consideradas	**Alternativa 1 – Optimización síncrona "clásica" (seleccionada)** ● Spring MVC + HikariCP ● índices y tuning SQL/ORM **Alternativa 2 – Arquitectura reactiva end-to-end (WebFlux + DB reactive)** **Alternativa 3 – Offloading masivo y caché distribuida** ● Colas (RabbitMQ/Kafka) + Redis amplio + pre-cálculos.

Evaluación de alternativas	**Alternativa 1 (Síncrona optimizada)** ● **Ventajas:** Rápida adopción, instrumentación simple, bajo costo, el equipo está familiarizado con esta arquitectura. ● **Desventajas:** Límite superior menor que reactivo en I/O extremo; requiere disciplina en SQL/ORM. **Alternativa 2 (Reactiva)** ● **Ventajas:** Concurrencia eficiente en I/O; buena base para altas cargas. ● **Desventajas:** Complejidad, mayor esfuerzo de pruebas/observabilidad; beneficio incierto con cuello en DB. **Alternativa 3 (Offloading + caché distribuida)** ● **Ventajas:** Reduce latencia percibida y picos; escalabilidad. ● **Desventajas:** Operación compleja, consistencia eventual.
Alternativa seleccionada y Justificación	Se adopta Alternativa 1 – Optimización síncrona “clásica”. Con el alcance actual y despliegue local, ofrece mejor relación beneficio/esfuerzo, el equipo tiene experiencia usando la alternativa 1 .
Decisiones relacionadas	- Estilo arquitectónico: Monolito modular MVC. - Lenguajes y frameworks: Spring Boot (Java) y React (TypeScript). - Mecanismo de persistencia: PostgreSQL con JPA.

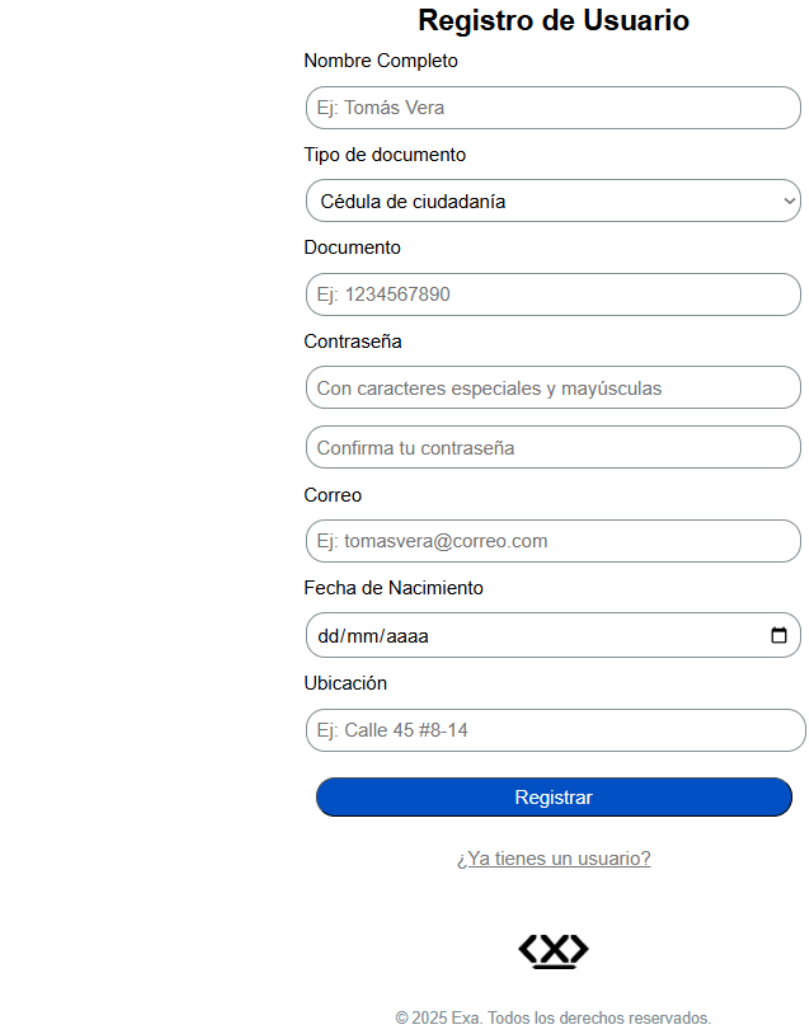
Comentarios	Sin comentarios adicionales
Problema/ Preocupación de diseño	**Usabilidad y experiencia de usuario:** Garantizar una interfaz de usuario intuitiva, accesible y coherente con las funcionalidades del sistema, que permita a los usuarios navegar, registrar y gestionar trámites automotrices (SOAT, tecnomecánica, cursos de conducción) sin dificultad. Se busca que la experiencia sea fluida, clara y uniforme entre los distintos roles (cliente, empleado y administrador), priorizando la simplicidad, legibilidad y consistencia visual.
Orientación de la decisión	La interfaz será desarrollada utilizando React con un enfoque modular basado en componentes reutilizables, aplicando principios de usabilidad y diseño centrado en el usuario (UCD). Se implementarán buenas prácticas de UX/UI, como: ● Diseño limpio y minimalista con jerarquía visual clara. ● Navegación lateral o superior coherente entre roles. ● Formularios simplificados con validaciones en tiempo real. ● Retroalimentación inmediata al usuario (mensajes de éxito, error o advertencia). Además, se usarán librerías y estilos coherentes (TailwindCSS) que garanticen consistencia en colores, tipografía y espaciado.
Supuestos	- Los usuarios no requieren conocimientos técnicos avanzados para usar el sistema. - Las pruebas de usabilidad se realizan internamente con los integrantes del equipo. - Todos los roles del sistema (cliente, empleado y administrador) accederán mediante navegadores modernos. - El diseño de las vistas seguirá los prototipos elaborados en Figma como referencia visual inicial.
Restricciones	- El proyecto no contempla pruebas de usabilidad con usuarios externos o públicos por limitaciones de tiempo. - El diseño visual debe seguir una paleta neutra y profesional definida por el equipo (colores institucionales). - No se implementarán funcionalidades avanzadas de accesibilidad (como traductores, lectores de pantalla o alto contraste) en esta fase. - Se debe garantizar compatibilidad con navegadores modernos (Chrome, Edge, Firefox) y resoluciones estándar de escritorio.

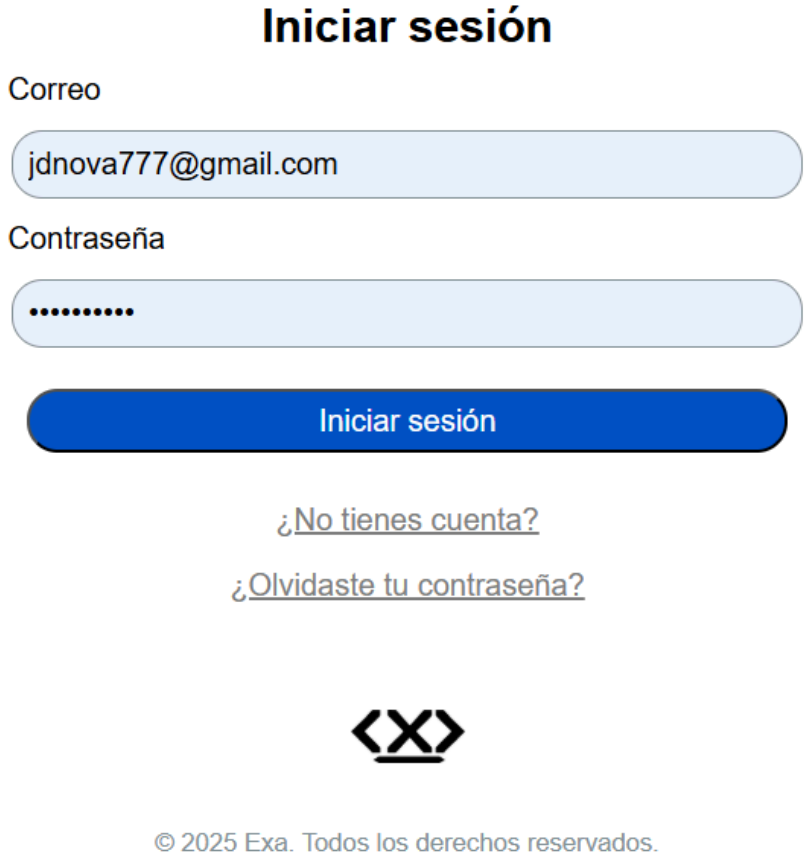
Requerimientos relacionados	RF-01: Registro e inicio de sesión por parte de todos los usuarios. RF-02: Inscripción de usuarios a cursos de conducción. RF-04: Recomendación inteligente de oficina de trámites cercana (Tecnomecánica). RF-05: Pasarela de pagos para servicios de la aplicación (Tecnomecánica, cursos de conducción y Soat). RF-06: Historial de Pagos. RF-07: Gestión de cursos de conducción y comparendos. RF-08: Creación de interfaces por rol (Administrador, Empleado y Cliente). RF-09: Estadísticas para la interfaz de rol Administrador. RF-10: Notificar al usuario cuando sus documentos de tránsito estén vencidos. RF-11: Recuperar contraseña
Alternativas consideradas	• **Alternativa 1:** Diseño de interfaz manual con CSS personalizado. • **Alternativa 2:** Uso de librerías de componentes (TailwindCSS / Material UI). • **Alternativa 3:** Implementación con frameworks de diseño profesional (Ant Design o Bootstrap).
Evaluación de alternativas	**Alternativa 1:** CSS personalizado: Ventajas: control total del estilo visual, sin dependencias externas. Desventajas: alto tiempo de desarrollo y posible inconsistencia visual entre componentes. **Alternativa 2:** TailwindCSS / Material UI: Ventajas: componentes estandarizados, diseño limpio, consistencia visual y agilidad en el desarrollo. Desventajas: curva de aprendizaje inicial para el equipo y limitaciones en personalización avanzada. **Alternativa 3:** Ant Design / Bootstrap: Ventajas: plantillas predefinidas y soporte extendido. Desventajas: diseño visual genérico y menor flexibilidad para personalización del branding.
Alternativa seleccionada y Justificación	Se selecciona la Alternativa 2 (TailwindCSS / Material UI) por ofrecer un balance adecuado entre rapidez de desarrollo, consistencia visual y flexibilidad. Permite mantener una identidad visual uniforme, reducir el esfuerzo en diseño desde cero y asegurar una experiencia agradable para los usuarios finales. Además, facilita la creación de componentes reutilizables y escalables, coherentes con la arquitectura modular del frontend en React.

Decisiones relacionadas	- Lenguajes y frameworks principales: React (TypeScript). - Estilo arquitectónico: Monolito modular MVC. - Seguridad: Autenticación JWT con sesiones seguras. - Despliegue y entornos: Ejecución local del frontend conectado al backend Spring Boot.
Comentarios	El equipo considera prioritaria la mejora continua de la experiencia de usuario, por lo que se prevé realizar una futura iteración con pruebas de usabilidad externas y posibles mejoras de accesibilidad (WCAG 2.1). Asimismo, se recomienda documentar los lineamientos visuales (colores, fuentes, espaciado, iconografía) en una guía de estilo común del proyecto.

4.2 Especificación de interfaz de usuario

Para todos los roles se presenta al mismo inicio de sesión y registro:


Registro de Usuario
Nombre Completo
Ej: Tomás Vera
Tipo de documento
Cédula de ciudadanía
Documento
Ej: 1234567890
Contraseña
Con caracteres especiales y mayúsculas
Confirma tu contraseña
Correo
Ej: tomasvera@correo.com
Fecha de Nacimiento
dd/mm/aaaa
Ubicación
Ej: Calle 45 #8-14
Registrar
¿Ya tienes un usuario?
© 2025 Exa. Todos los derechos reservados.


Iniciar sesión
Correo
jdnova777@gmail.com
Contraseña
••••••••••
Iniciar sesión
¿No tienes cuenta?
¿Olvidaste tu contraseña?
© 2025 Exa. Todos los derechos reservados.

Cada usuario contará con distintos inicios / portales al iniciar sesión:

SmartTraffic

Gestión de cursos de conducción y comparendos

Estadísticas de Estudiantes

SmartTraffic

Para información más detallada accede a tu perfil

Realiza consultas al simit de tus vehiculos

SmartTraffic

Gestión de cursos de conducción y comparendos

Estadísticas de usuarios

A su vez los usuarios con rol de cliente contarán con la interfaz para el resto de funcionalidades como recomendación de oficinas de trámites, consultas a, pagos, servicios, perfil, registro de vehiculo, recuperar contraseña y demás especificadas en el documento de detalle de casos de uso:


Solicitar Servicio
Tipo de Servicio
Curso de Conducción
Tipo de Curso
Curso de Conducción A1
Buscar Oficinas y tramitar servicio
© 2025 Exa. Todos los derechos reservados.

Solicitar Servicio

Tipo de Servicio

SOAT

Placa

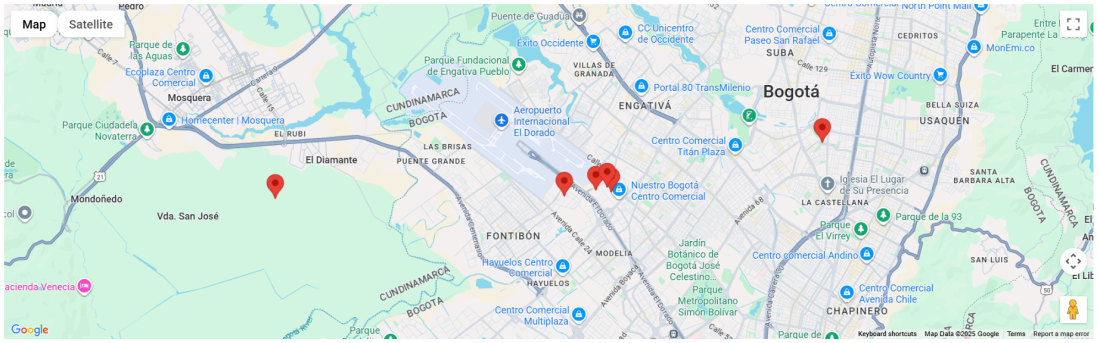
JLR654

Buscar Oficinas y tramitar servicio

SmartTraffic Inicio Servicios Pagos Perfil

Oficinas de trámite cercanas

Por favor, elige una ubicación



SmartTraffic Inicio Servicios Pagos Perfil

Correo Electrónico: jdnova777@gmail.com

Rol: Cliente

Ubicación:

Editar Perfil

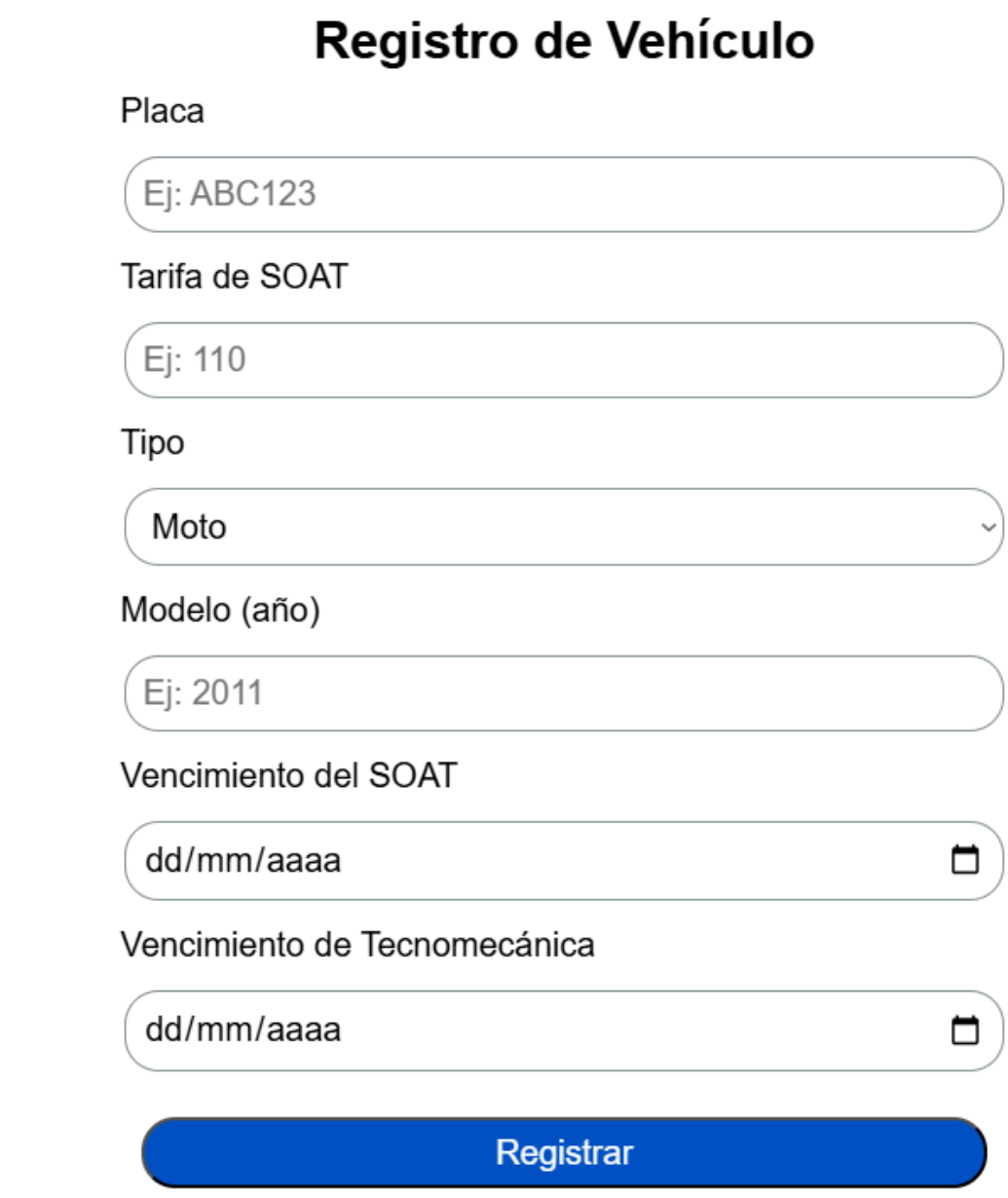
¿Quieres actualizar tu contraseña?

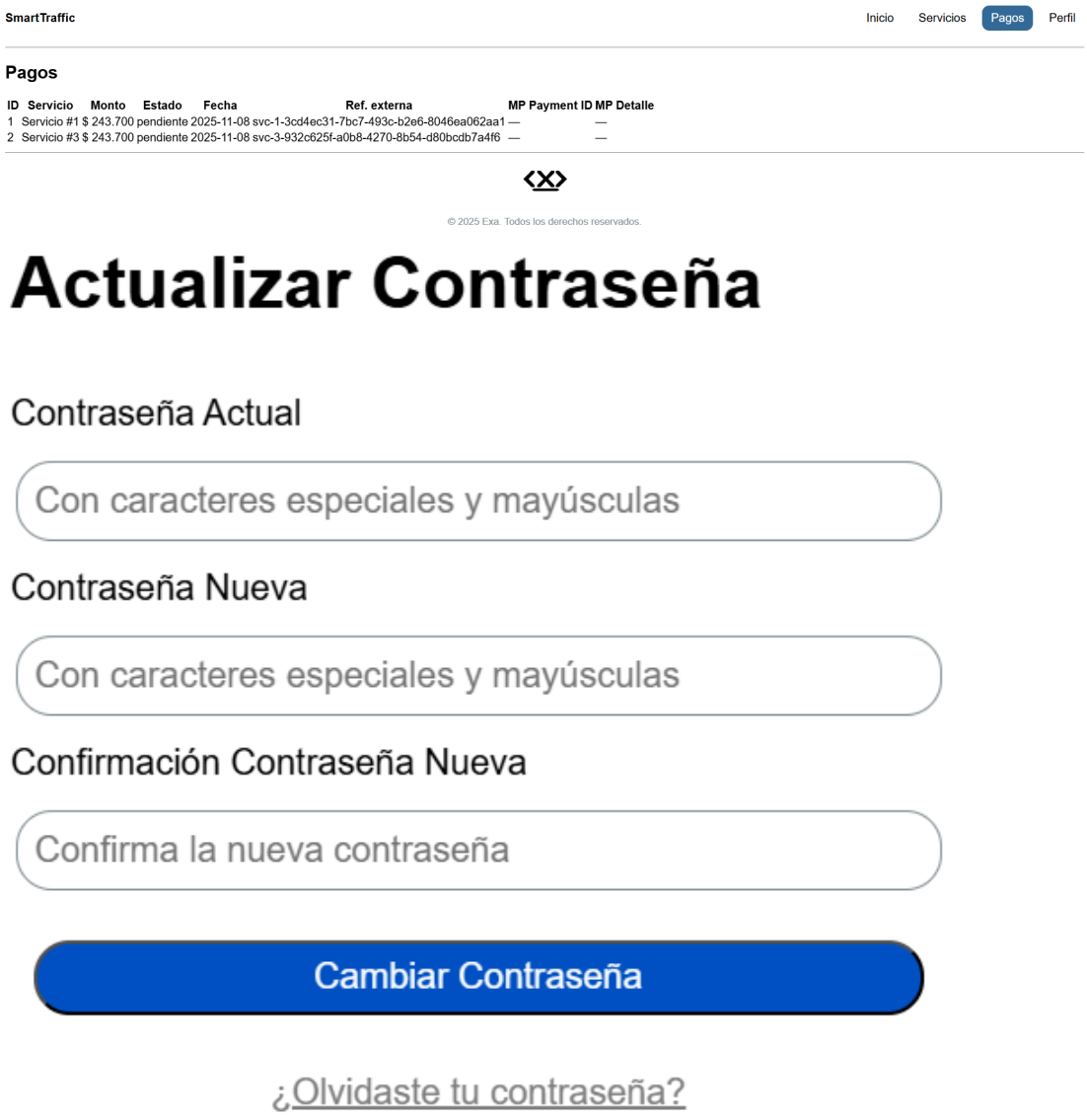
Vehículos

Tipo / Placa	Modelo	SOAT (tipo)	Vencimientos	Acciones	
Liviano Privado · ABC123	2009	SOAT (110)	SOAT: 2025-12-16 · Tecno: 2025-12-31	Eliminar	Editar
Pesado Particular · URT505	2011	SOAT (110)	SOAT: 2025-12-24 · Tecno: 2026-04-28	Eliminar	Editar

Registrar Vehículo

Cerrar Sesión Eliminar Cuenta


Registro de Vehículo
Placa
Ej: ABC123
Tarifa de SOAT
Ej: 110
Tipo
Moto
Modelo (año)
Ej: 2011
Vencimiento del SOAT
dd/mm/aaaa
Vencimiento de Tecnomecánica
dd/mm/aaaa
Registrar


SmartTraffic
Inicio
Servicios
Pagos
Perfil
Pagos
ID Servicio Monto Estado Fecha Ref. externa MP Payment ID MP Detalle
1 Servicio #1 $ 243.700 pendiente 2025-11-08 svc-1-3cd4ec31-7bc7-493c-b2e6-8046ea062aa1 — —
2 Servicio #3 $ 243.700 pendiente 2025-11-08 svc-3-932c625f-a0b8-4270-8b54-d80bcdb7a4f6 — —
© 2025 Exa. Todos los derechos reservados.
Actualizar Contraseña
Contraseña Actual
Con caracteres especiales y mayúsculas
Contraseña Nueva
Con caracteres especiales y mayúsculas
Confirmación Contraseña Nueva
Confirma la nueva contraseña
Cambiar Contraseña
¿Olvidaste tu contraseña?

Reestablecimiento de contraseña

Ingresa tu correo electrónico y te enviaremos un código único para reestablecer tu contraseña.

Correo electrónico:

Ej: tomas@correo.com

Enviar Correo

Actualizar Usuario

Nombre Completo

Juliamcito Pekka

Correo

jdnova777@gmail.com

Actualizar

Mientras que los roles de empleado y administrador cuentan con otras funcionalidades como ver, registrar, editar aliados, vista de cursos de conducción, enrolamiento a cursos, entre otros.

Cursos de Conducción

- **Curso de Conducción**
  Capacidad parcial: 0
  Capacidad total: 25
- **Curso de Conducción**
  Capacidad parcial: 0
  Capacidad total: 25
- **Curso de Conducción**
  Capacidad parcial: 0
  Capacidad total: 25
- **Curso de Comparendo**
  Capacidad parcial: 0
  Capacidad total: 25
- **Curso de Conducción**
  Capacidad parcial: 0
  Capacidad total: 25
- **Curso de Conducción**
  Capacidad parcial: 0
  Capacidad total: 25
- **Curso de Conducción**
  Capacidad parcial: 0
  Capacidad total: 25
- **Curso de Conducción**
  Capacidad parcial: 0
  Capacidad total: 25

Aliados

Registrar Aliado

ID	Nombre	Lat	Lon	SOAT	Techno	Acciones	
1	Punto SOAT Centro	4.685	-74.118	Sí	Sí	Editar	Eliminar
2	Centro Técnico Norte	4.6854	-74.122	Sí	Sí	Editar	Eliminar
3	Estación Técnico Centro	4.6862	-74.119	Sí	Sí	Editar	Eliminar
4	Sucursal Occidente	4.6838	-74.13	Sí	Sí	Editar	Eliminar
5	Punto SOAT Suba	4.6976	-74.064	Sí	Sí	Editar	Eliminar
6	Quiosco SOAT El Dorado	4.6832	-74.204	Sí	Sí	Editar	Eliminar
7	CIA	4.64594945	-74.0776053940335	No	No	Editar	Eliminar

Nuevo Aliado

Registrar Aliado

Nombre

Ej: Quiosco SOAT El Dorad

Ubicación

Ej: Calle 45 #8-14

Selecciona una opción del listado para capturar coordenadas.

☐ SOAT

☐ Tecnomecánica

Registrar Volver

Gestión de cursos

Los cursos se cargan automáticamente. Puedes ver inscritos y desinscribir sin buscar email. Para inscribir, primero busca el usuario por email.

Buscar usuario para inscribir

Email usuario@dominio.com

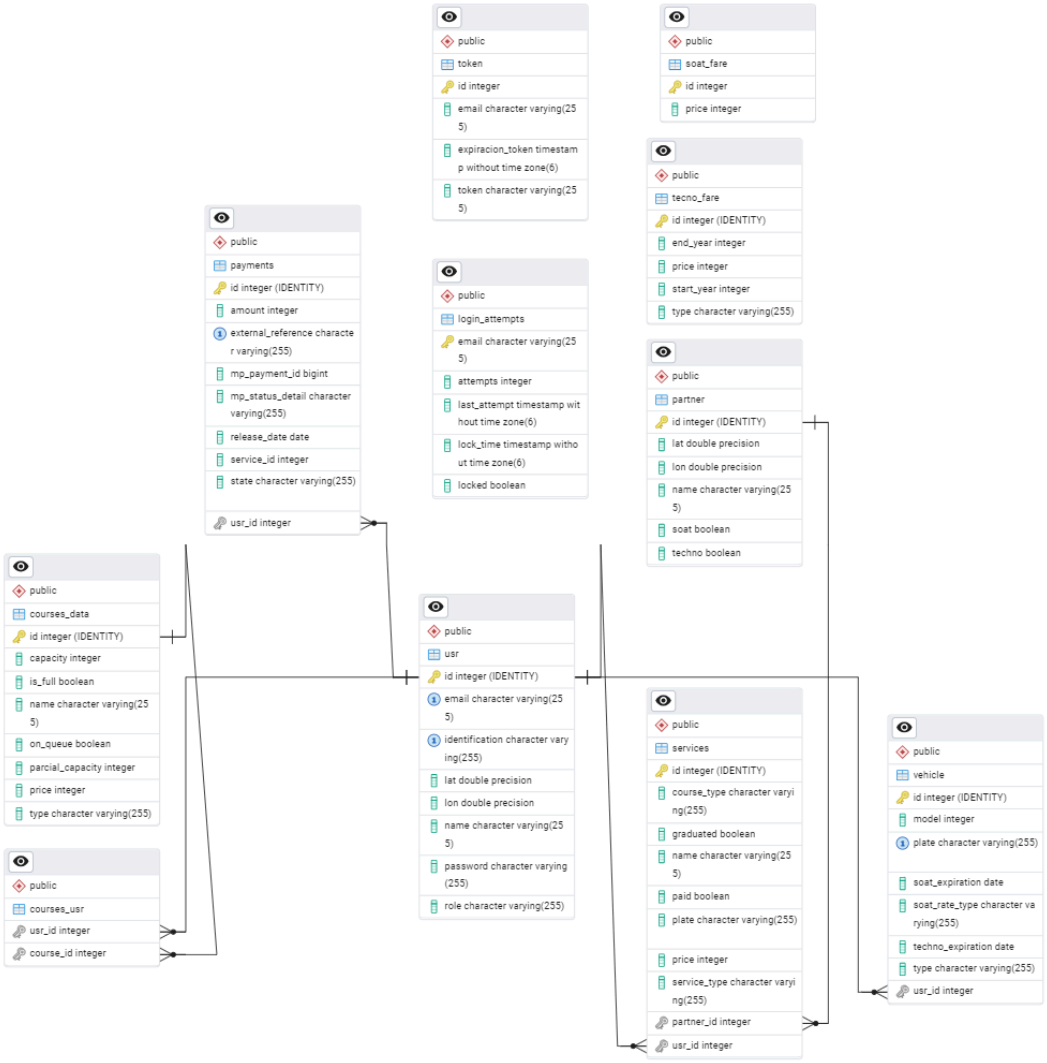
Buscar usuario Limpiar selección de usuario Sin usuario seleccionado

Cursos

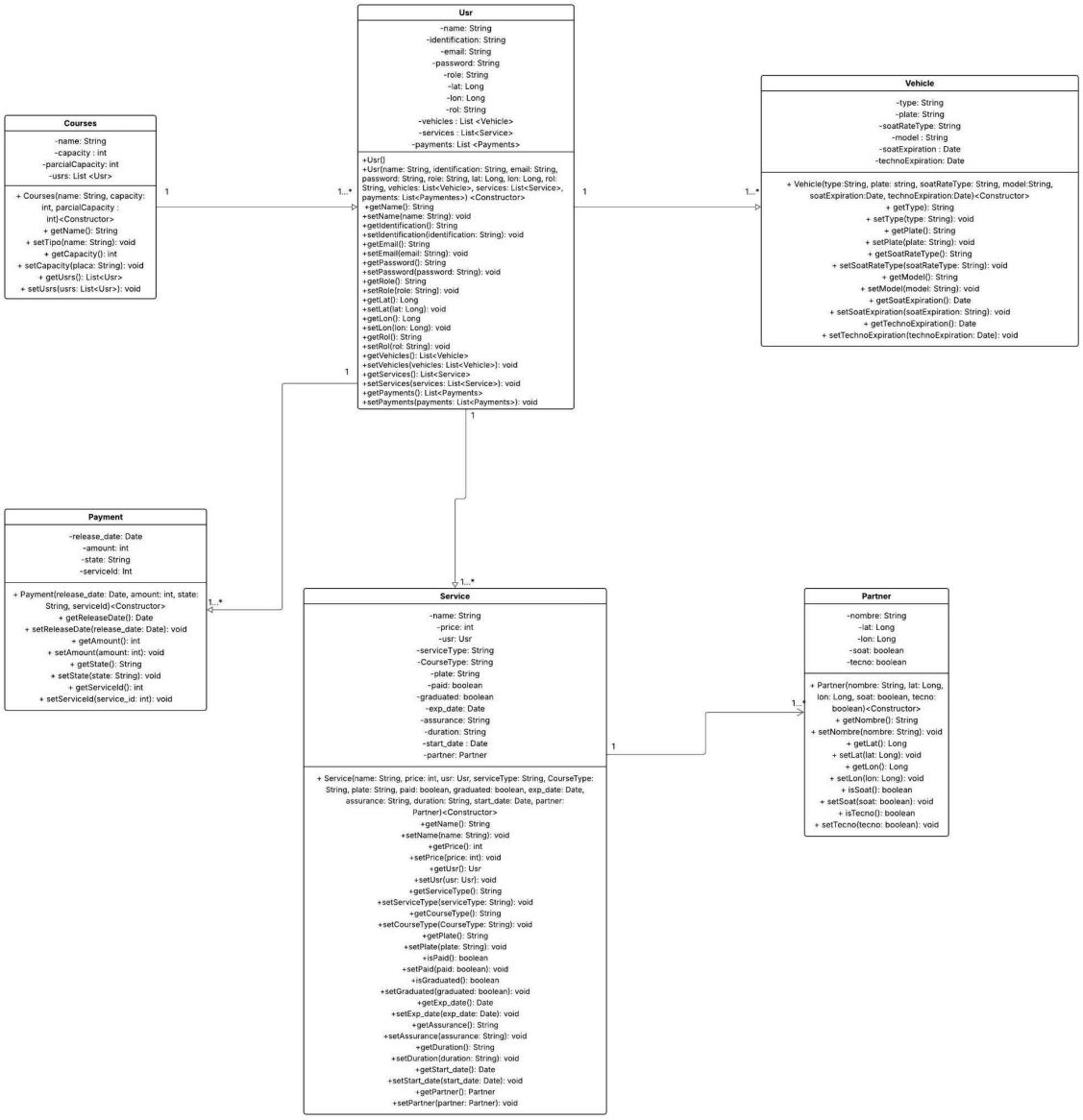
Refrescar

capacity	name	parcial capacity	price	type	acciones	
25	Curso de Conducción	0 (25 cupos)	$ 1.230.000	C1	Inscribir (elige usuario)	Ver inscritos
25	Curso de Conducción	0 (25 cupos)	$ 1.290.000	C2	Inscribir (elige usuario)	Ver inscritos
25	Curso de Conducción	0 (25 cupos)	$ 1.500.000	C3	Inscribir (elige usuario)	Ver inscritos

4.3 Modelo de datos



4.4 Modelo de clases



5 Control de Cambios

CONTROL DE CAMBIOS

Fecha	Descripción	Autor(es)
4/11/2025	Revisión y corrección	Felipe Triviño