DISEÑO DE LA SOLUCIÓN

ALEJANDRO HERNANDEZ

DANIEL VELOZA

FICHA 3147238

SENA



Centro de Gestión de Mercados, Logística y Tecnologías de la Información.

SENA – SOFT 2025 – DESARROLLO LIBRE

21/10/2025

**ANALISIS DE REQUERIMIENTOS**

El sistema web 'Pragma' busca automatizar el proceso de compra de tiquetes aéreos, ofreciendo una experiencia intuitiva, segura y eficiente para los pasajeros. El objetivo principal es integrar en una misma plataforma la búsqueda de vuelos, la reserva de asientos, el registro de pasajeros y la generación de tiquetes electrónicos.

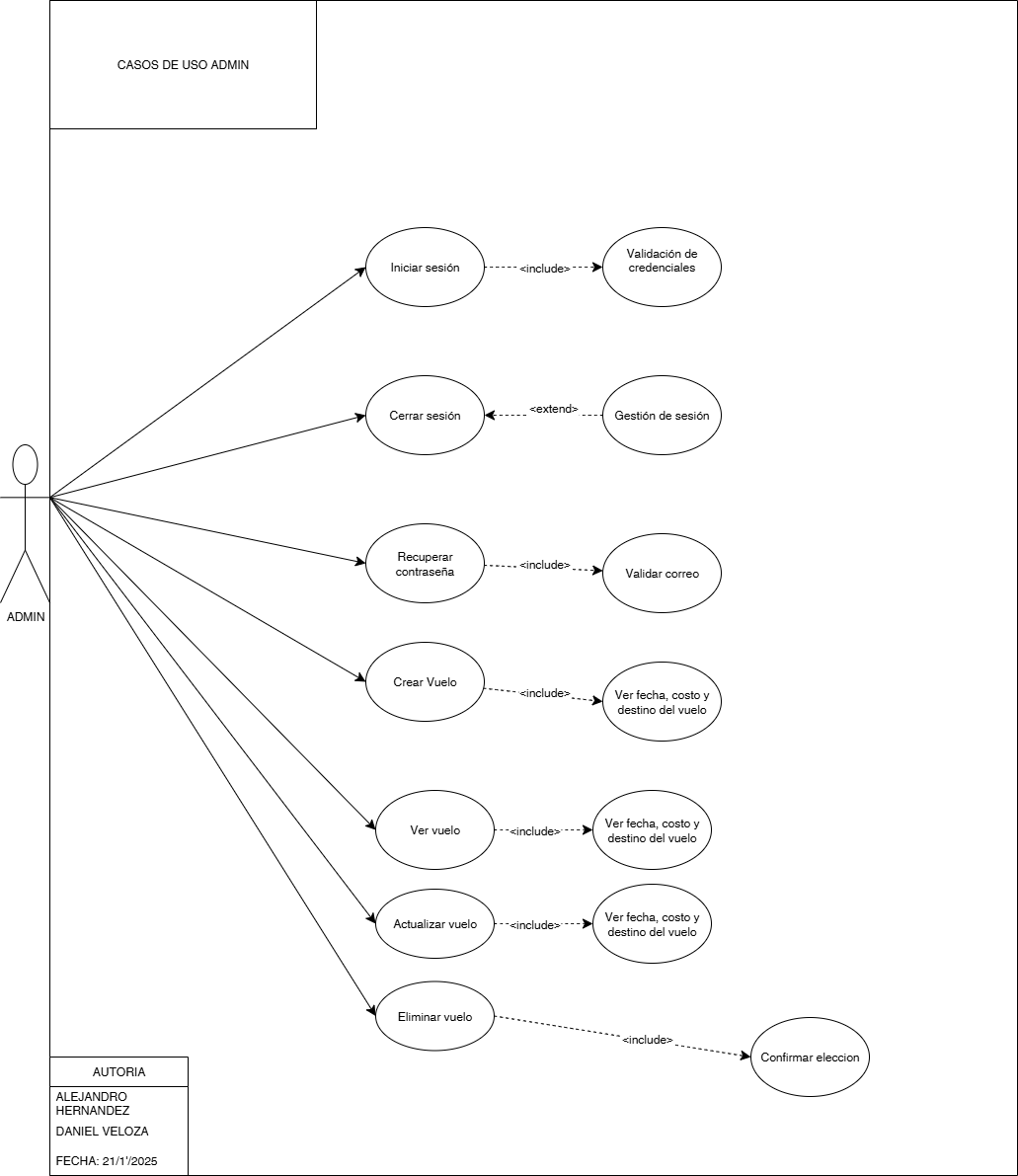
Entre los requisitos funcionales más relevantes se encuentran: la autenticación de usuarios, el control de concurrencia en la reserva de asientos, la simulación de pago, y la generación de códigos de reserva únicos. En cuanto a los requisitos no funcionales, se prioriza la usabilidad, seguridad, rendimiento y compatibilidad con navegadores modernos.

El sistema contará con dos actores principales: el pasajero (usuario final), quien podrá realizar la compra de tiquetes; y el administrador, encargado de gestionar la información de vuelos y mantener actualizada la disponibilidad del sistema.

**DIAGRAMAS UML**

**DIAGRAMA DE CASOS DE USO**

1. **DIAGRAMA DE CASOS DE USO ADMIN:**

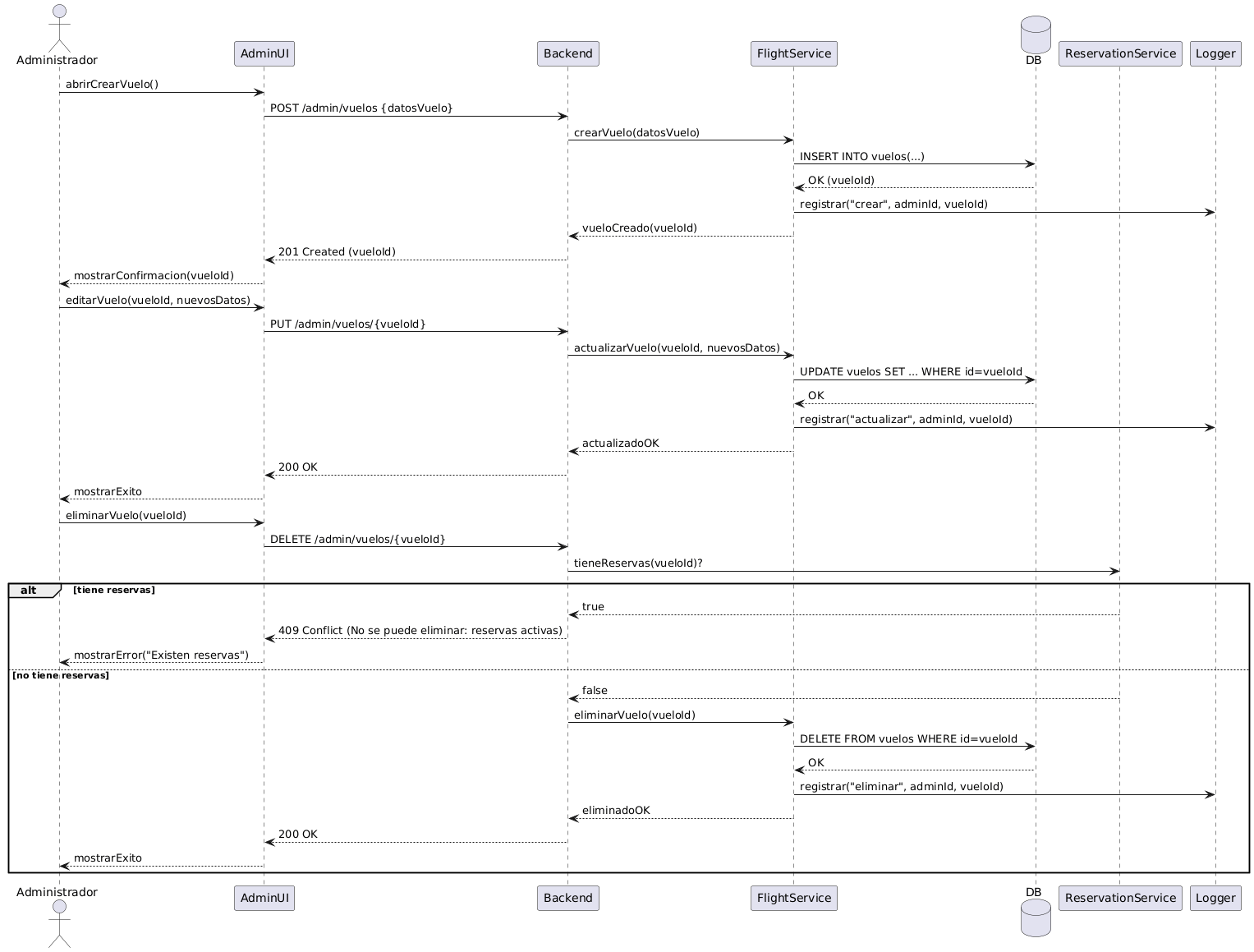


1. **DIAGRAMA DE CASOS DE USO PASAJERO:**

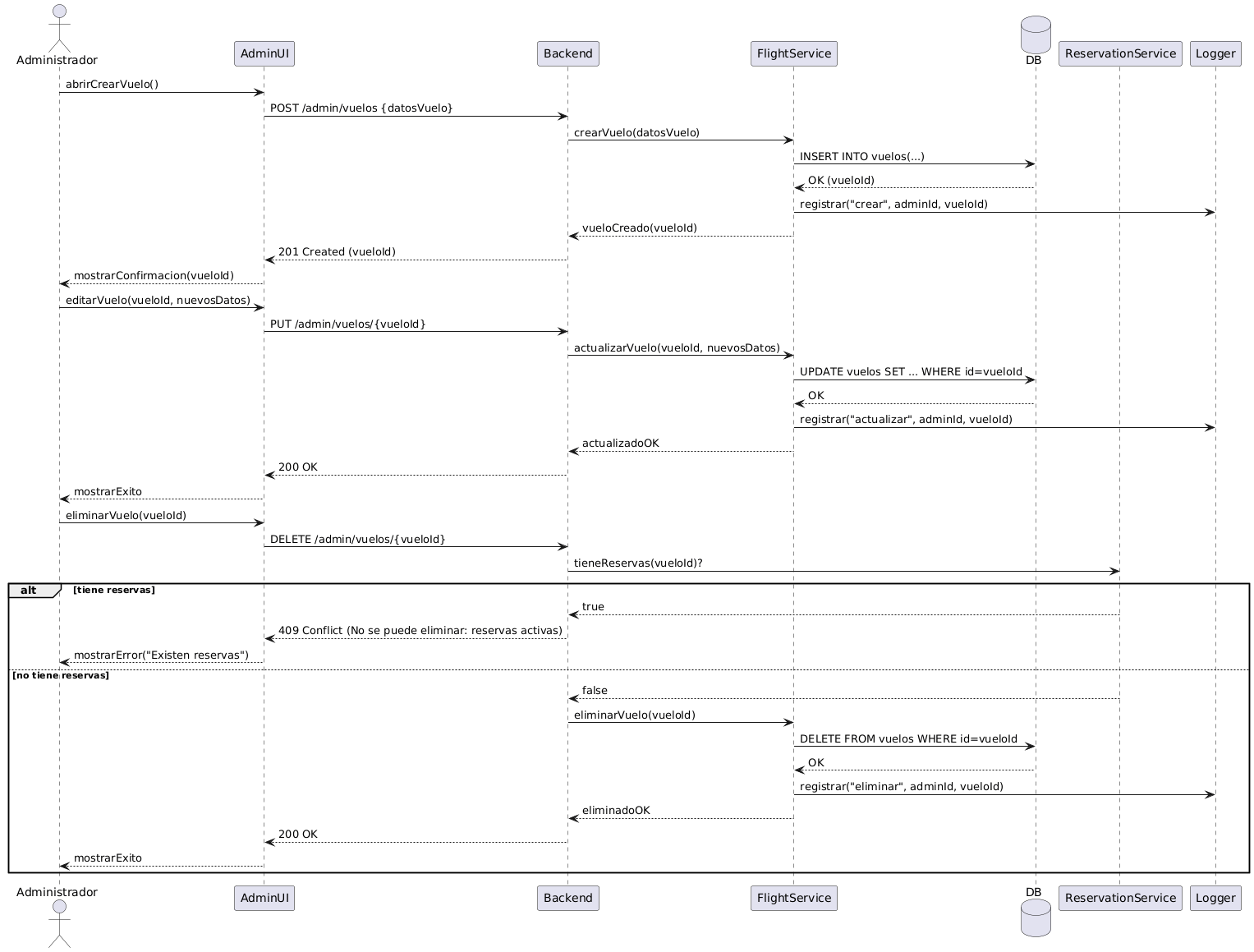


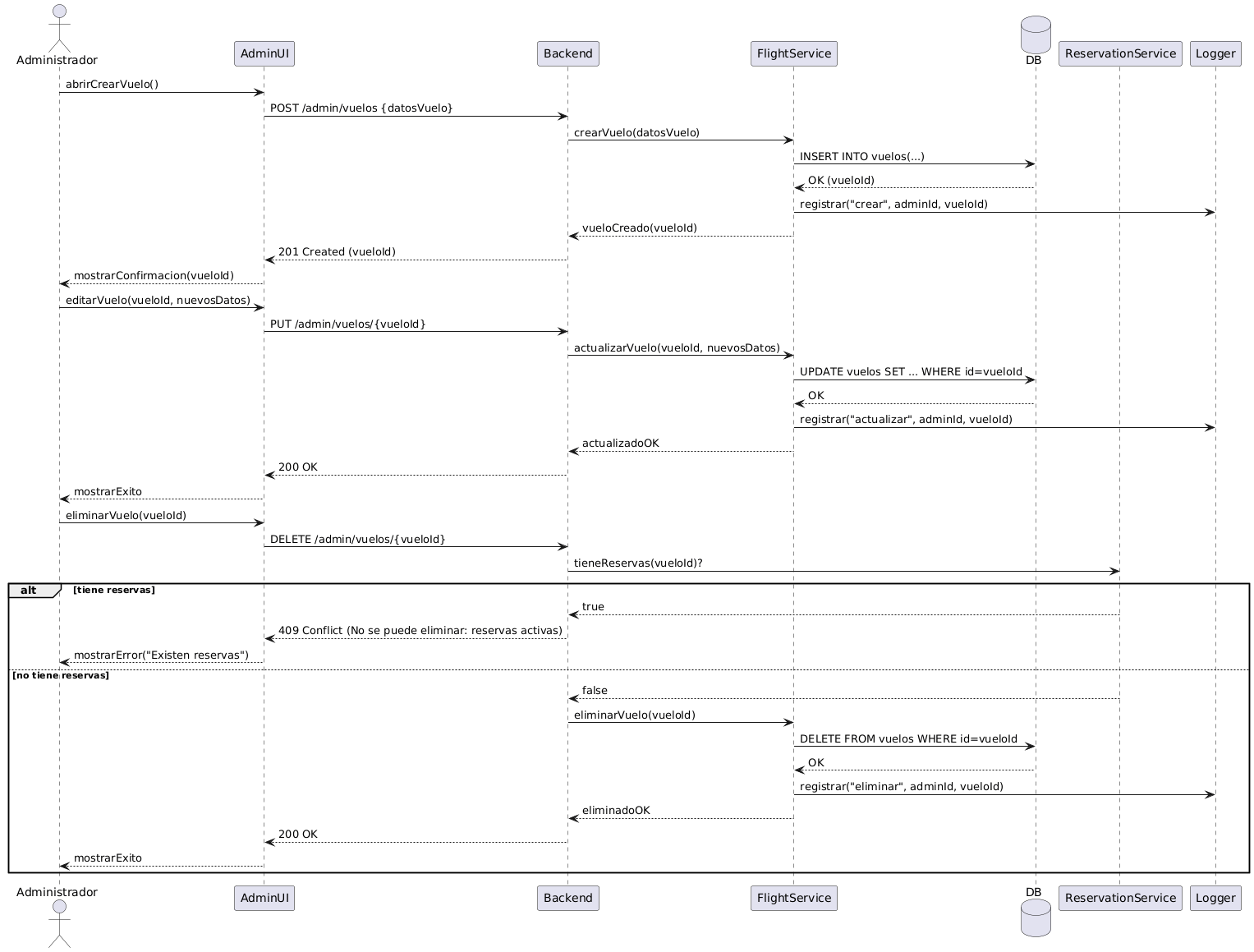
1. **DIAGRAMA DE SECUENCIAS**

**DIAGRAMA DE SECUENCIAS DE ADMIN**



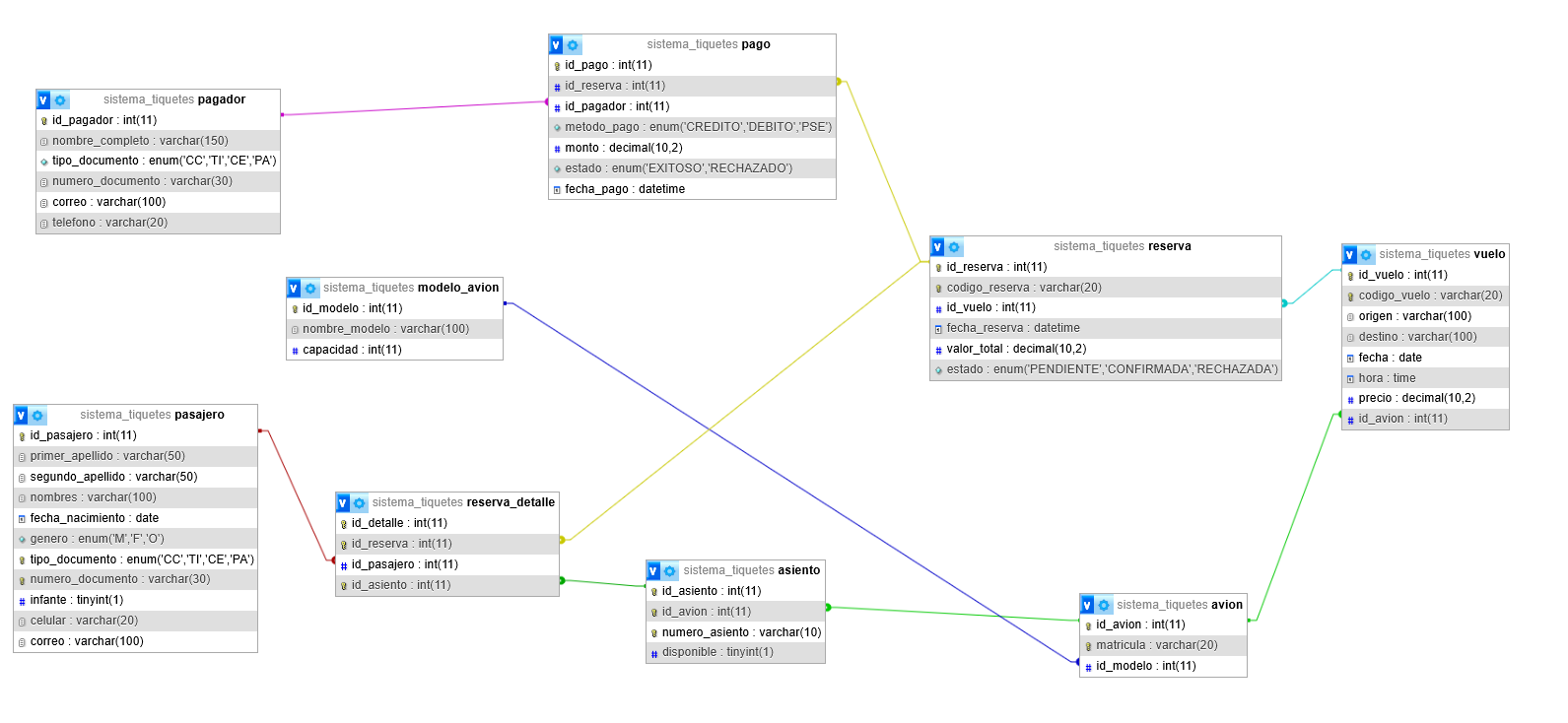
**DIAGRAMA DE SECUENCIAS DE PASAJERO**

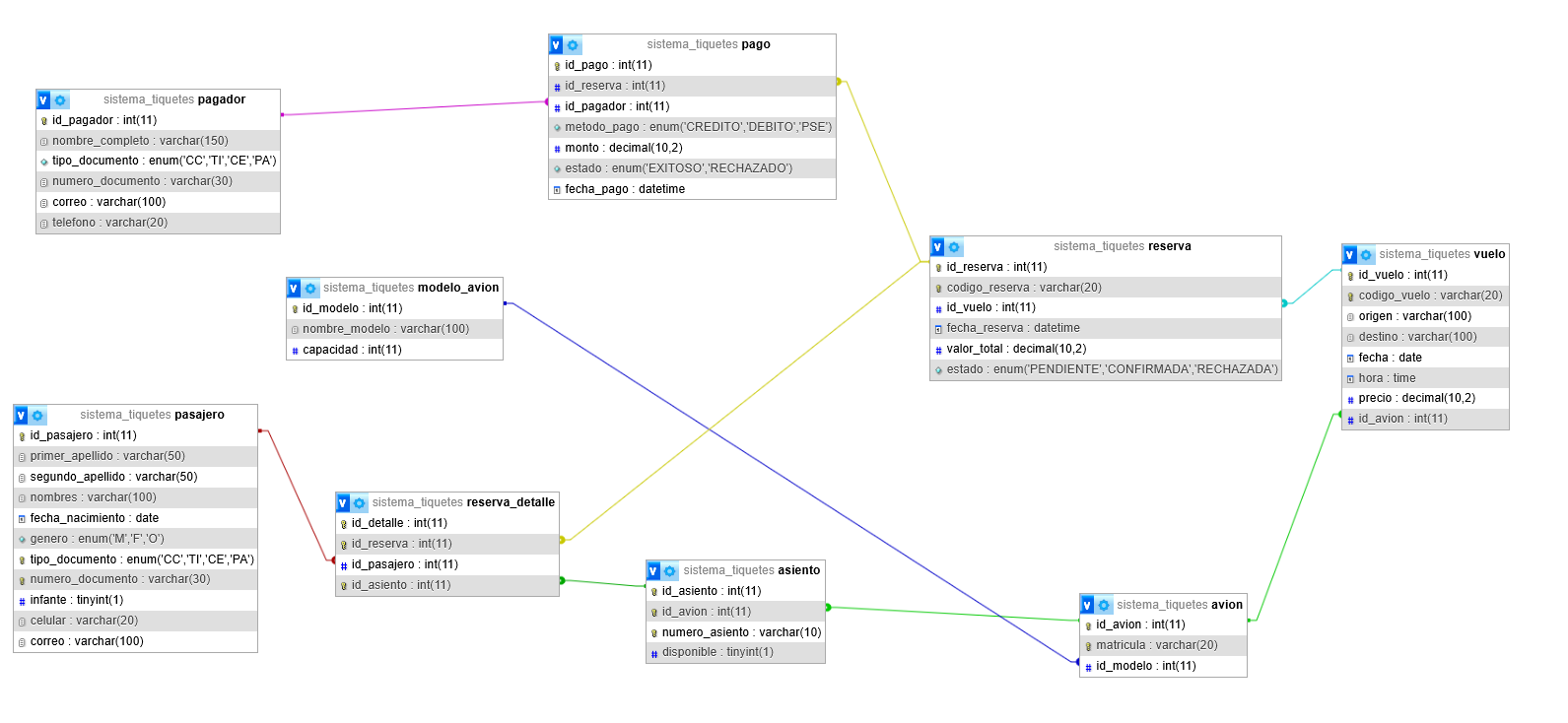
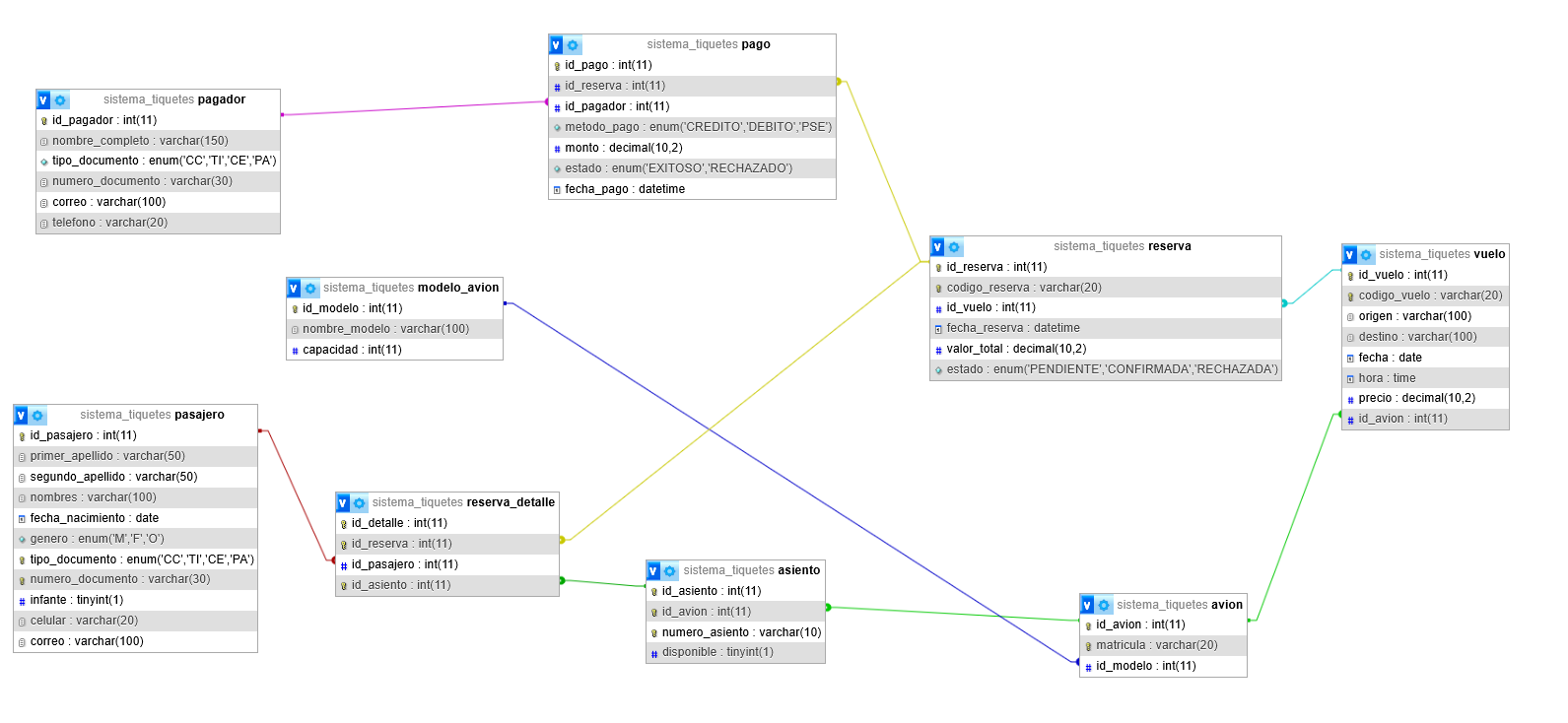




**DIAGRAMA DE CLASES**

**MODELO DE BASE DE DATOS, E R**



El sistema “Pragma” está soportado por una base de datos relacional implementada en MySQL, diseñada bajo criterios de normalización y consistencia referencial.

A continuación, se describen las principales entidades y sus relaciones:

1. Entidad pagador

Contiene la información del titular de la compra o transacción.

* id\_pagador: Identificador primario.
* nombre\_completo, tipo\_documento, numero\_documento, correo, telefono: datos del pagador.
* Se relaciona uno a muchos con la tabla pago.

2. Entidad pago

Registra los pagos realizados (simulados) por los usuarios.

* id\_pago: PK.
* id\_reserva: FK → reserva.id\_reserva.
* id\_pagador: FK → pagador.id\_pagador.
* metodo\_pago: Enum (‘CREDITO’, ‘DEBITO’, ‘PSE’).
* monto: Valor total del pago.
* estado: Enum (‘EXITOSO’, ‘RECHAZADO’).
* fecha\_pago: Fecha y hora del pago.

Cada reserva puede tener un pago asociado, pero un pagador puede tener varios pagos (relación 1:N).

3. Entidad reserva

* Registra las reservas de vuelos realizadas por los pasajeros.
* id\_reserva: PK.
* codigo\_reserva: Código único generado.
* id\_vuelo: FK → vuelo.id\_vuelo.
* fecha\_reserva: Fecha de creación.
* valor\_total: Monto total.
* estado: Enum (‘PENDIENTE’, ‘CONFIRMADA’, ‘RECHAZADA’).

Se relaciona uno a muchos con reserva\_detalle.

4. Entidad reserva\_detalle

Representa los pasajeros y asientos incluidos en una reserva.

* id\_detalle: PK.
* id\_reserva: FK → reserva.id\_reserva.
* id\_pasajero: FK → pasajero.id\_pasajero.
* id\_asiento: FK → asiento.id\_asiento.

Relación N:M entre pasajeros y asientos, gestionada mediante esta tabla intermedia.

5. Entidad pasajero

Contiene la información de cada pasajero registrado.

* id\_pasajero: PK.
* primer\_apellido, segundo\_apellido, nombres.
* fecha\_nacimiento, genero, tipo\_documento, numero\_documento.
* infante (booleano).
* celular, correo.

Un pasajero puede estar asociado a múltiples reservas (a través de reserva\_detalle).

6. Entidad vuelo

Define los vuelos disponibles.

* id\_vuelo: PK.
* codigo\_vuelo, origen, destino, fecha, hora, precio.
* id\_avion: FK → avion.id\_avion.

Un vuelo pertenece a un avión, y puede tener muchas reservas.

7. Entidad avion

Identifica los aviones registrados.

* id\_avion: PK.
* matricula: Identificador único del avión.
* id\_modelo: FK → modelo\_avion.id\_modelo.

8. Entidad modelo\_avion

Define los diferentes tipos de avión.

* id\_modelo: PK.
* nombre\_modelo.
* capacidad: Número máximo de asientos.

Un modelo de avión puede estar asociado a varios aviones.

9. Entidad asiento

Representa cada asiento físico dentro de un avión.

* id\_asiento: PK.
* id\_avion: FK → avion.id\_avion.
* numero\_asiento: Identificador del asiento (ej. “12A”).
* disponible: Booleano (1 = libre, 0 = ocupado).

Un avión tiene varios asientos, los cuales se relacionan con las reservas mediante reserva\_detalle.

Relaciones clave en la BD

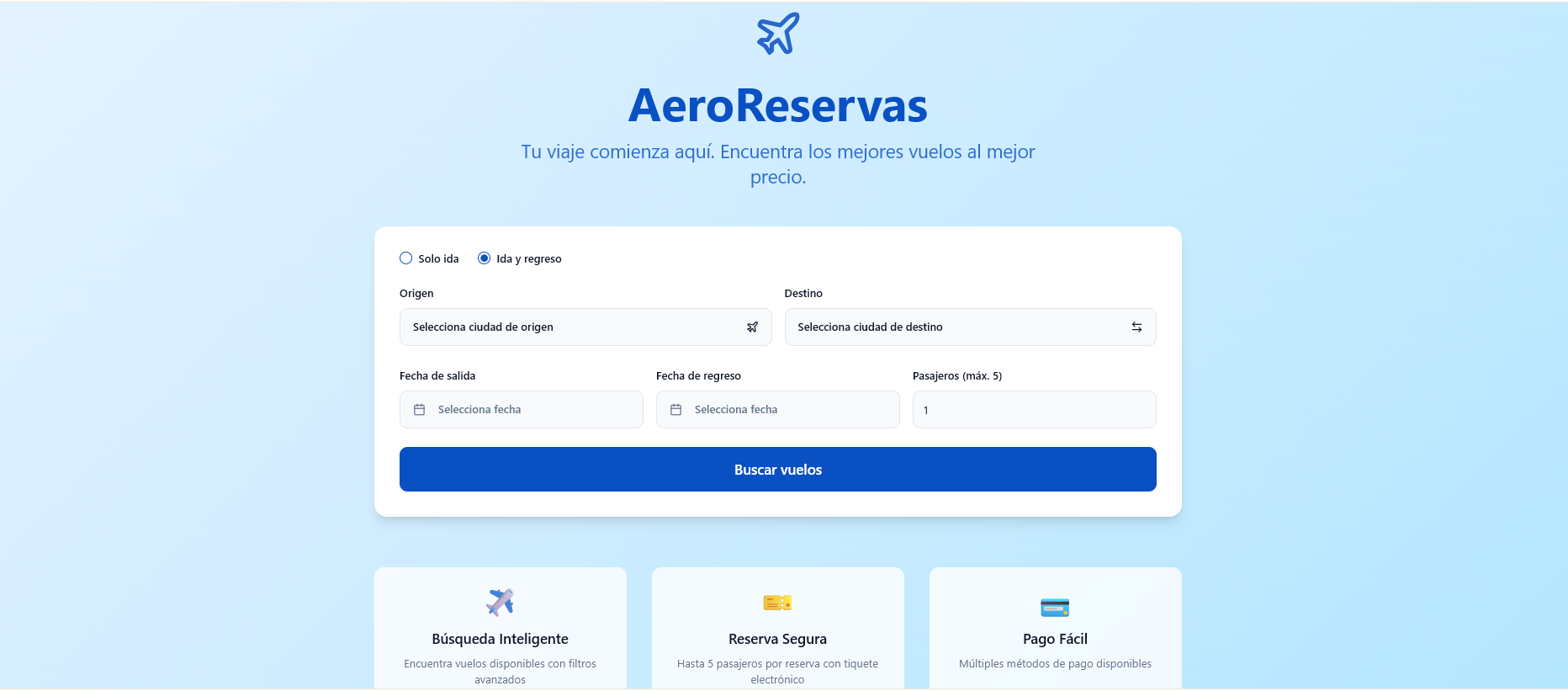
* modelo\_avion (1:N) → avion
* avion (1:N) → asiento
* vuelo (1:N) → reserva
* reserva (1:N) → reserva\_detalle
* reserva\_detalle (N:1) → pasajero
* reserva\_detalle (N:1) → asiento
* reserva (1:1) → pag
* pago (N:1) → pagador

El modelo garantiza integridad referencial, evita duplicidad de datos y permite implementar control de concurrencia en la asignación de asientos.

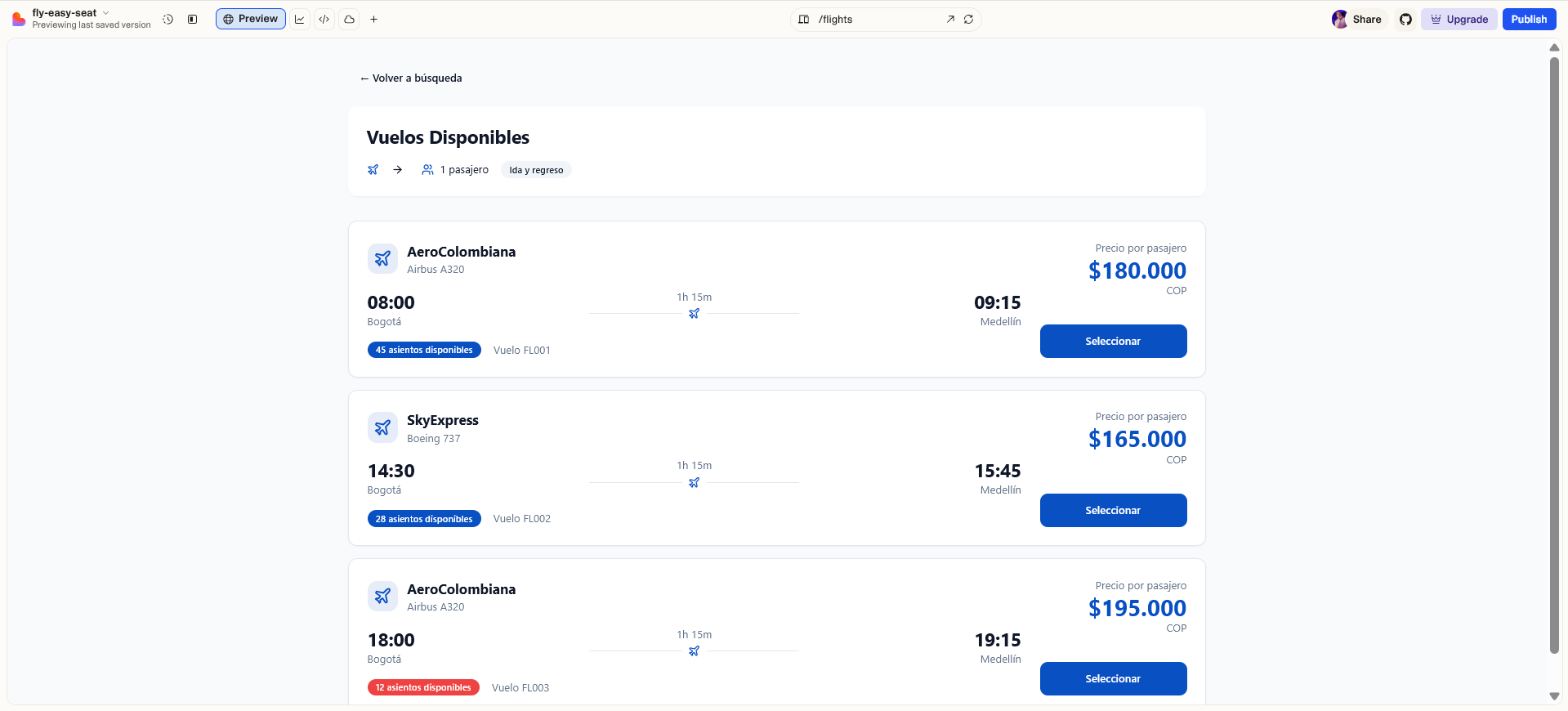
Está preparado para ampliarse con nuevas funcionalidades, como historial de vuelos, facturación electrónica o programas de fidelización.

MOCKUPS UX/UI

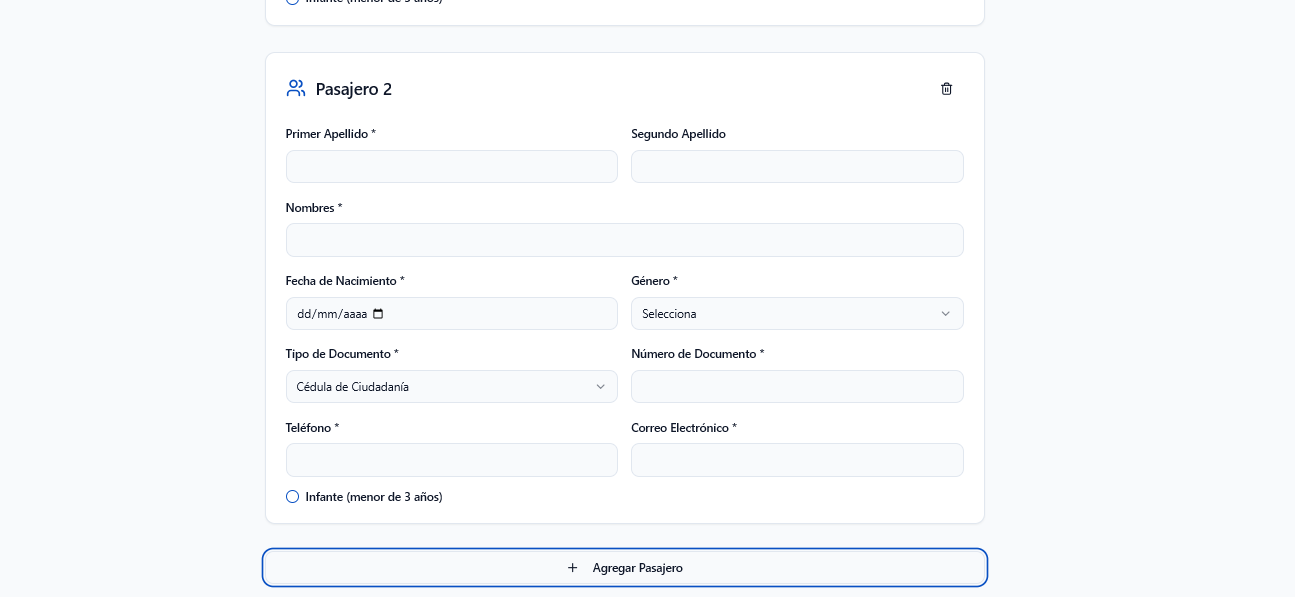
MODULO DEL FORMULARIO PARA CONSULTAR LOS VUELOS

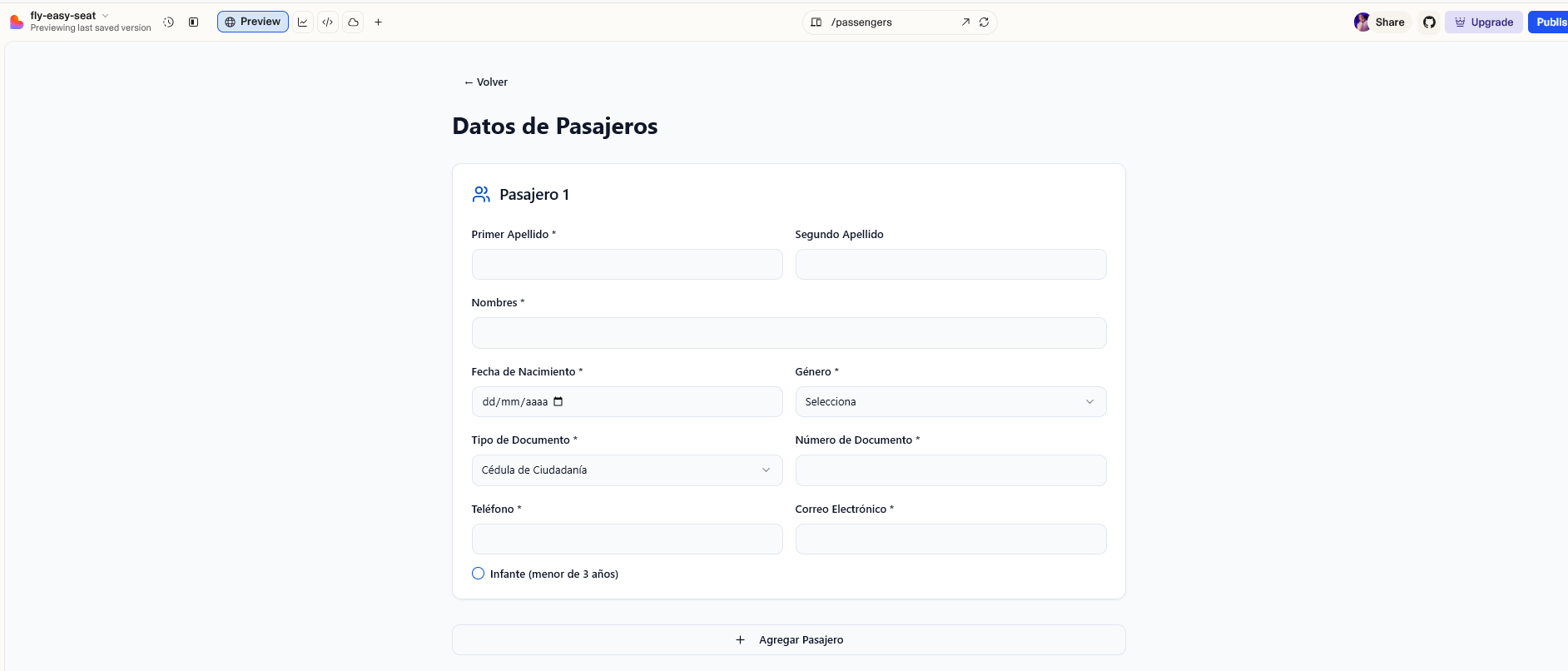


MODULO DE RESULTADOS DE LA CONSULTA DE LOS VUELOS

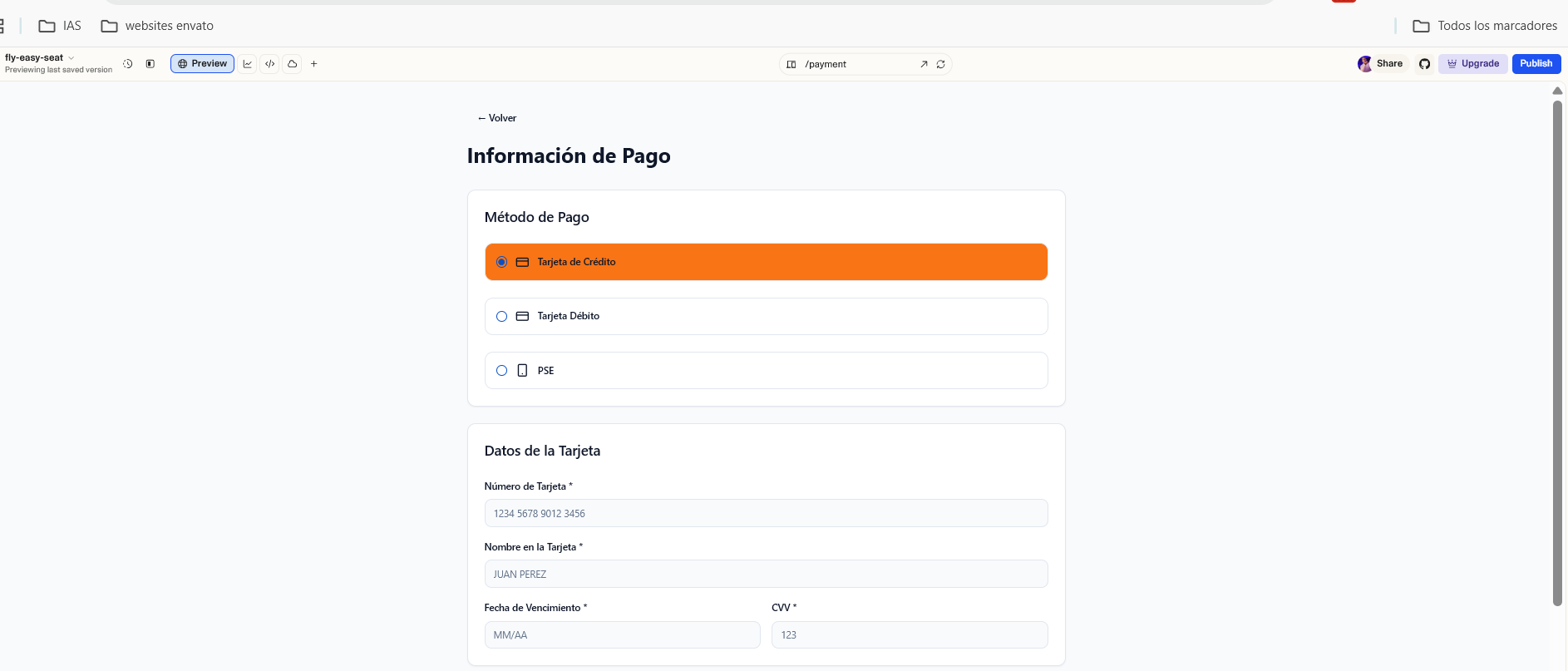


FORMULARIOS DE REGISTRO DE INFORMACION PERSONAL PARA EL VUELO





DE REGISTRO DE INFORMACION PERSONAL PARA EL PAGO



PROPUESTA DE ARQUITECTURA (MVC)

El sistema se implementará bajo el patrón de arquitectura Modelo-Vista-Controlador (MVC), con el fin de garantizar la separación de responsabilidades y una mayor mantenibilidad del código.

• Modelo (Model): Contendrá las clases y estructuras que representan las entidades del sistema, como Usuario, Vuelo, Asiento, Reserva y Pago. Se encargará de la interacción directa con la base de datos MySQL.

• Vista (View): Será el componente encargado de la interfaz de usuario, desarrollada con HTML5, CSS3, Bootstrap y React.js, ofreciendo una experiencia moderna, adaptable y dinámica.

• Controlador (Controller): Gestionará la lógica de negocio y coordinará la comunicación entre las vistas y los modelos. Será implementado con Node.js y Express, utilizando controladores específicos (por ejemplo, AuthController, FlightController, ReservationController).

El sistema estará diseñado para garantizar escalabilidad, reutilización de componentes y fácil integración con servicios externos en futuras versiones.