

# INSTITUTO TECNOLÓGICO DE PUEBLA

UNIDAD ISISTEMAS

OPERATIVOS II

Documentación del Proyecto: Sistema de Reserva de Vuelos

Docente:

Alejandra Reyes Vélez

Alumnos:

Héctor Ortiz Morales

Julián Jiménez Reyes

Giovany Félix Sotero

Fecha de Entrega:

22 de septiembre de 2025

## **Planteamiento del Problema**

Actualmente, muchas aerolíneas enfrentan problemas de consistencia y concurrencia al momento de gestionar las reservas de vuelos. Esto puede ocasionar que dos usuarios reserven el mismo asiento al mismo tiempo, generados conflictos pérdidas económicas y mala experiencia para el cliente.

### Escenario propuesto

* Varias personas buscan y reservan asientos de un vuelo.
* Ningún asiento debe venderse dos veces.
* Muchos usuarios pueden consultar al mismo tiempo.
* Al tener a un usuario seleccionando los asientos, estos deben estar en reserva, lo que conlleva a que dos usuarios no deben seleccionar al mismo tiempo un asiento.
* El usuario puede elegir más de un asiento con las siguientes categorías: Adulto $65950, Niño $60500, Tercera Edad $50000, considerar los valores para el pago. Los asientos de primera clase $120000.
* Hasta que la compra se realice se pone como deshabilitado o no disponible el asiento.
* Se despliega un formato de la compra que se realizó por parte del usuario, indicando: Numero de vuelo, origen, destino, fecha, hora, número de asientos y lugar de salida.

### Componentes

1. **Servicio de búsqueda de vuelos**

Permite consultar vuelos, horarios, asientos disponibles.

1. **Servicio de reservas**

Gestiona el bloqueo temporal y confirmación de asientos.

1. **Base de datos central replicada**

Almacena información de vuelos, asientos y reservas.

1. **Mecanismo de concurrencia**

Evita que dos usuarios tomen el mismo asiento (locks o transacciones).

### Objetivos distribuidos

* **Consistencia:** Cuando un asiento se marca como reservado, debe reflejarse inmediatamente en todos los nodos.
* **Concurrencia:** Muchos pueden leer y reservar a la vez sin interferir.

## **Objetivos del Proyecto**

### Objetivo general

Desarrollar un sistema de reserva de vuelos que garantice la consistencia y concurrencia en la gestión de asientos, permitiendo a múltiples usuarios consultar y reservar en tiempo real sin duplicidad de ventas, ofreciendo al mismo tiempo una experiencia segura, eficiente y confiable.

### Objetivos específicos

1. Implementar un servicio de búsqueda de vuelos que permita consultar horarios, origen y destino, disponibilidad de asientos y precios en tiempo real.
2. Diseñar un mecanismo de reservas temporales que asegure que ningún asiento pueda ser seleccionado por más de un usuario al mismo tiempo, evitando conflictos de concurrencia.
3. Integrar reglas de negocio para categorías de pasajeros (adulto, niño, tercera edad y primera clase) con sus precios correspondientes y cálculo automático del costo total de la compra.
4. Desarrollar un sistema de bloqueo y confirmación de asientos que mantenga los asientos en estado reservado hasta que la compra sea finalizada o hasta que expire el tiempo límite de la reserva.
5. Generar un comprobante digital de la compra con la información del vuelo: número de vuelo, origen, destino, fecha, hora, número de asientos adquiridos y sala de salida.

## **Solución Propuesta**

Como solución propuesta, creamos una aplicación compuesta por un frontend estático usando HTML y Javascript, un backend HTTP stateless en Java y una base de datos PostgreSQL. El backend expone APIs REST para búsqueda, visualización y gestión de reservas y compras. La lógica de estado de asientos se obtiene por combinación de tablas estáticas y tablas dinámicas.

### Tecnologías Escogidas

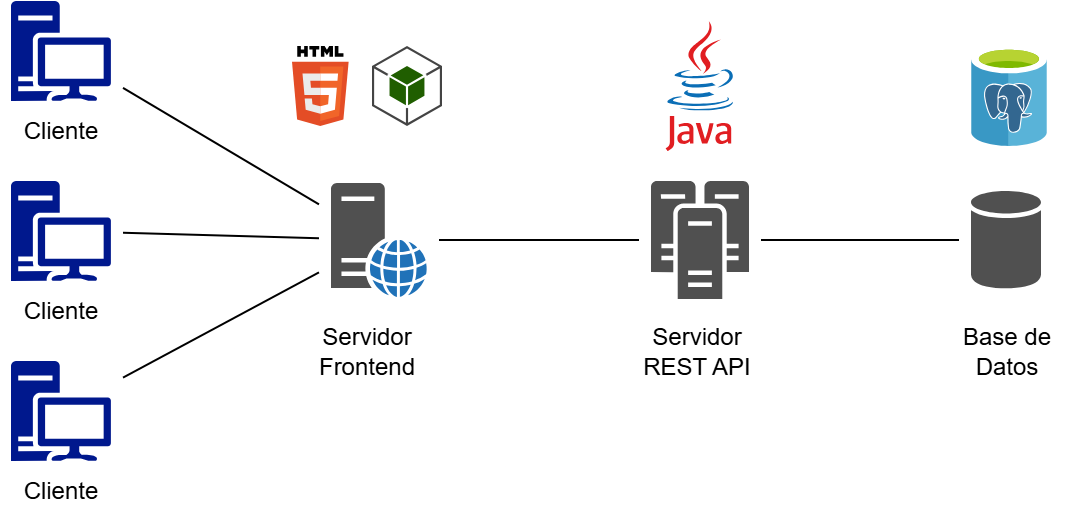
* **Java HTTP Server:** Java para control total, sin frameworks innecesarios para el scope del curso. Fácil de depurar y ligero.
* **PostgreSQL:** ACID, restricciones, transacciones y buen soporte SQL. Ideal para garantizar consistencia.
* **HTML y Javascript**. Un frontend simple y suficiente para prototipo.

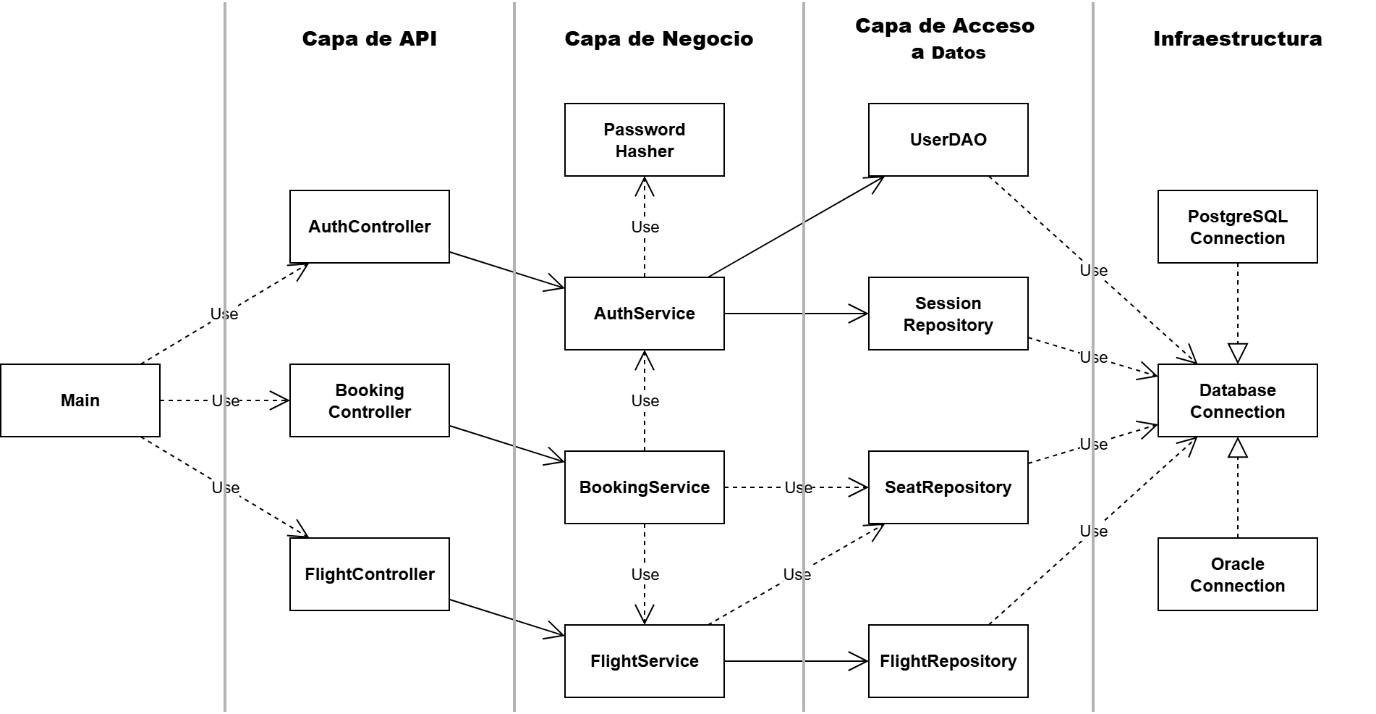
### Arquitectura

Para el sistema se ha escogido una arquitectura por capas. Esto permite separar las instancias en el punto clave para su escalado.

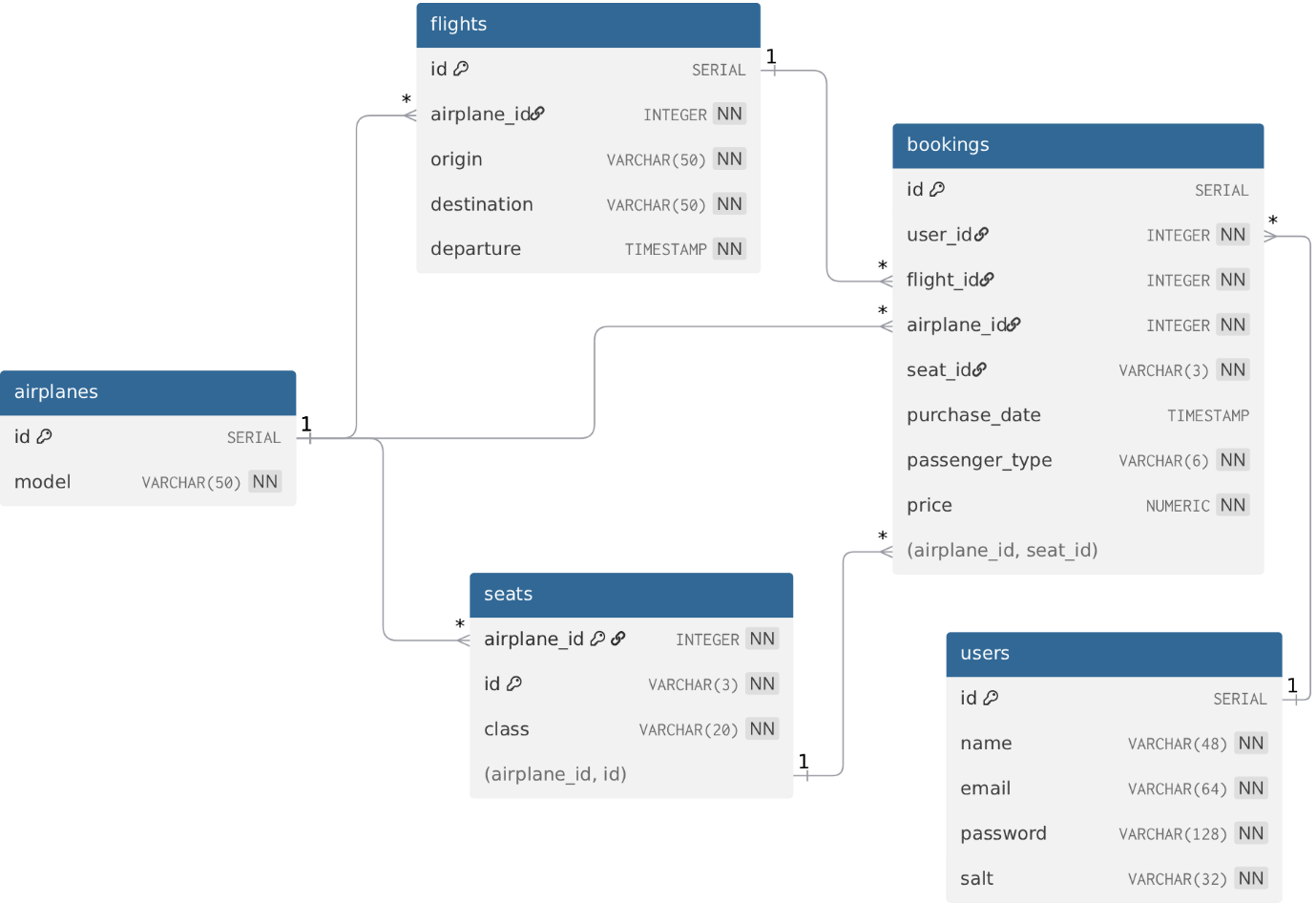
1. **Presentación**
   * HTML + Javascript que consume la API REST.
   * Responsable de renderizar el diagrama del avión y gestionar interacciones.
2. **API**
   * Servidor HTTP stateless con Java
   * Controllers/Handlers: exponen los endpoints, parsean requests/responses.
   * Services: reglas de negocio.
   * Repositories: acceso a la base de datos.
   * Models: endidades de dominio y responses.
3. **Persistencia**
   * Base de datos PostgreSQL
   * Tablas estáticas para datos del negocio.
   * Tablas dinámicas para datos temporales como sesiones y reservas temporales.

Al ser stateless las instancias del servidor HTTP, las sesiones y reservas temporales son compartidas entre instancias ya que se guardan en la Base de Datos. Esto permite escalar horizontalmente sin problemas, ya que al arrancar o cerrar una instancia no se pierden las sesiones no reservas.





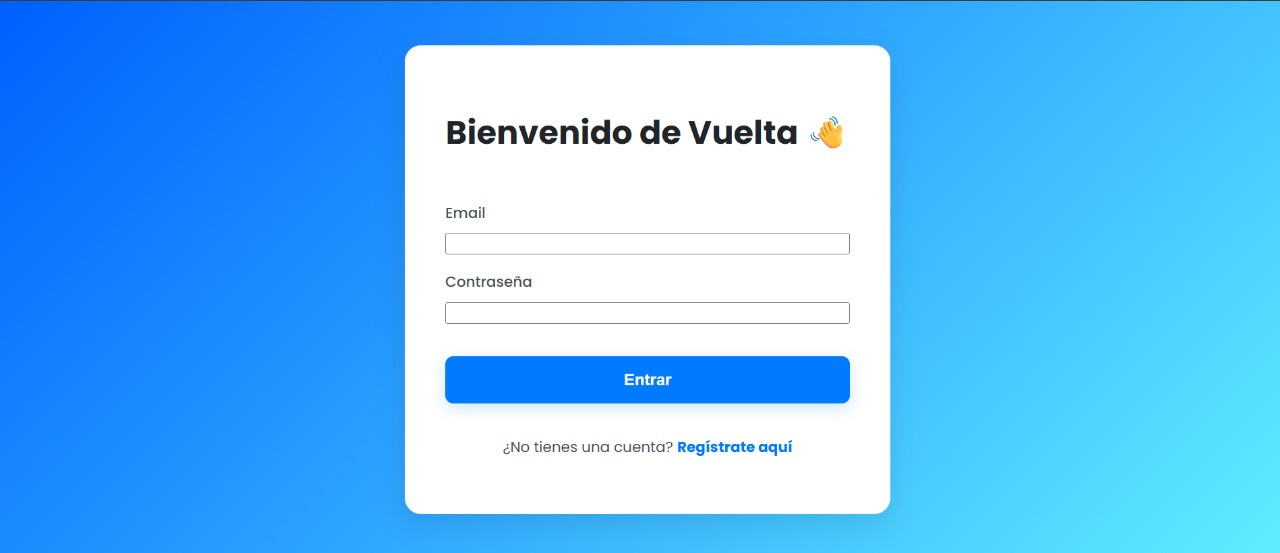
### Diseño de la Base de Datos



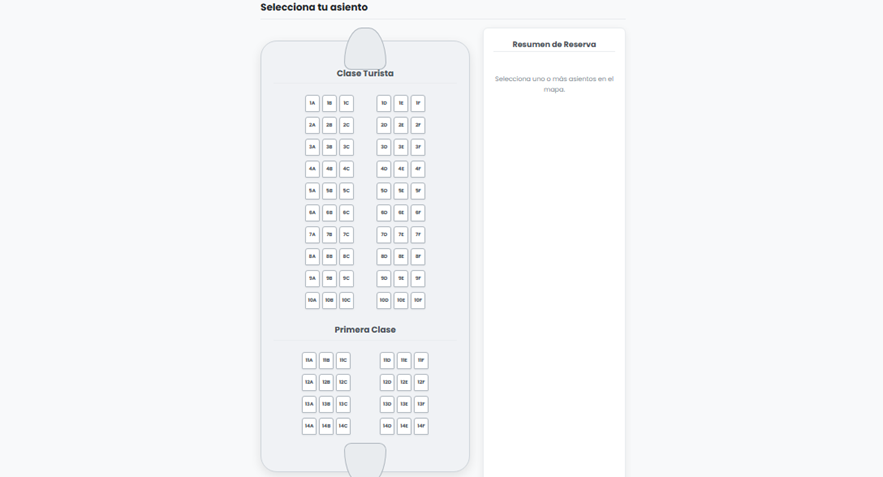
#### Consideraciones y decisiones

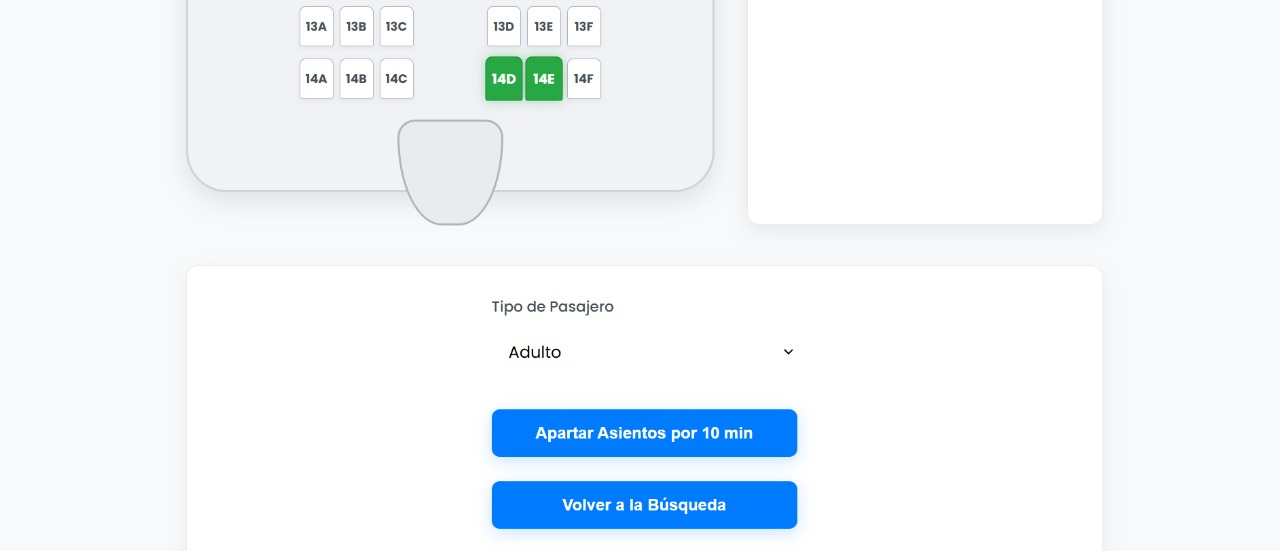
* Seats se relaciona con airplane directamente. Esto permite reutilizar el layout del mismo avión en distintos vuelos.
* El estado del asiento no es una columna. Se calcula con una consulta que combina seats, bookings y reservations.
* Sessions. Son guardadas en la Base de Datos para simplicidad y se validan desde el servidor.
* Reservations temporales. Cuentan con un Timestamp de caducidad y el servicio booking verifica la expiración a la hora de interactuar.

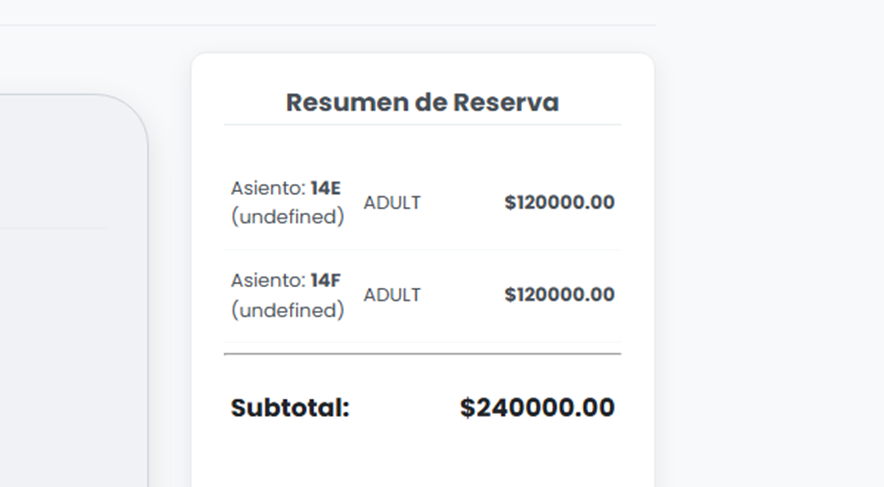
## **Evidencia de Código y Ejecución**











|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38 | **public** **class** **FlightService** {  **private** **final** FlightRepository flightRepository;  **private** **final** SeatRepository seatRepository;  **public** **FlightService**(FlightRepository flightRepository, SeatRepository seatRepository) {  **this**.flightRepository = flightRepository;  **this**.seatRepository = seatRepository;  }  **public** Result<List<Flight>> **search**(Map<String, String> queryParams) {  **return** Result.of(() -> {  FlightFilter filter = FlightFilter.fromQuery(queryParams);    **return** flightRepository.findFlights(filter)  .orElseThrow()  .orElseGet(List::of);  });  }  **public** Result<Optional<FlightDetails>> **getFlightDetails**(**int** flightId) {  **return** Result.of(() -> {  Optional<Flight> flightOpt = flightRepository.getFlight(flightId).orElseThrow();  **if** (flightOpt.isEmpty()) **return** Optional.empty();  Flight flight = flightOpt.get();  Optional<List<Seat>> seats = seatRepository.getSeatsForFlight(flight.id()).orElseThrow();  **return** Optional.of(  **new** FlightDetails(flight, seats.get())  );  });  }  } |

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49 | **public** Result<Optional<Booking>> **bookSeat**(**int** flightId, String seatId, PassengerType passengerType, String sessionId) {    **return** Result.of(() -> {  Optional<User> userOpt = authService.validateSession(sessionId).orElseThrow();  **if** (userOpt.isEmpty())  **return** Optional.empty();  User user = userOpt.get();  Optional<FlightDetails> flightOpt = flightService.getFlightDetails(flightId).orElseThrow();  **if** (flightOpt.isEmpty())  **return** Optional.empty();  FlightDetails flightDetails = flightOpt.get();  Optional<Seat> seatOpt = flightDetails.seats().stream()  .filter(seat -> seat.id().equals(seatId))  .findAny();  **if** (seatOpt.isEmpty())  **return** Optional.empty();  Seat seat = seatOpt.get();  **if** (seat.status() == Status.BOOKED)  **return** Optional.empty();  Optional<Integer> ownerIdOpt = seatRepository.getReservationOwnerId(flightId, seatId).orElseThrow();  **if** (ownerIdOpt.isPresent() && ownerIdOpt.get() != user.id())  **return** Optional.empty();  Booking booking = **new** Booking(  user.id(),  flightId,  flightDetails.flight().airplaneId(),  seatId,  LocalDateTime.now(),  passengerType,  seat.seatClass() == Seat.Class.FIRST? **new** **BigDecimal**("120000.00") : passengerType.price  );  seatRepository.book(booking).orElseThrow();  **return** Optional.of(booking);  });  } |

## **Conclusiones**

Según lo visto a lo largo del desarrollo, el sistema de reservas de vuelos logró dar solución a un problema central en los sistemas distribuidos: la gestión de concurrencia y consistencia en escenarios de alta demanda. El uso de mecanismos de bloqueo temporal junto con transacciones en PostgreSQL permitió garantizar que ningún asiento fuera asignado dos veces, resolviendo así un conflicto común en entornos multiusuario.

Al recopilar lo mencionado sobre la arquitectura, la división en capas dentro del servidor HTTP en Java proporcionó claridad en la organización del código y un mayor control sobre la lógica del negocio. La decisión de mantener el backend en un estado stateless, apoyado en la base de datos para sesiones y reservas, resultó clave para permitir la escalabilidad horizontal y la tolerancia a fallos.

Considerando los objetivos planteados, la integración de reglas de negocio para categorías de pasajeros y precios diferenciales fortaleció el realismo del sistema y permitió validar el correcto funcionamiento de la lógica de reservas. El frontend, aunque sencillo, cumplió con el propósito de ofrecer una interfaz efectiva para la interacción con los usuarios.

En conjunto, los resultados obtenidos demuestran que la solución implementada no solo satisface los requerimientos académicos, sino que también refleja prácticas aplicables en entornos reales, resaltando la importancia de un diseño modular, el uso adecuado de transacciones y la correcta separación de responsabilidades en la arquitectura de software.