MANUAL TÉCNICO

Universidad Don Bosco — Escuela de Computación
Asignatura: Desarrollo de Aplicaciones con Web Frameworks
Docente: Mario Alvarado
Fecha: Octubre 2025

Repositorios: https://github.com/pgap22/dwf-catedra

https://github.com/pgap22/dwf-frontend-catedra

Equipo de Trabajo

Carné	Nombre Completo	Github
BS242634	Gerardo Rafael Bonilla Saz	pgap22
JA242663	Fernando Alexander Jiménez Artiga	Ferchofloo
PA242675	Rodrigo Daniel Pineda Ardón	Rodrogas666
VE242685	Leandro Alberto Valencia Escobar	leannsttar

Introducción

Este manual técnico documenta el desarrollo del proyecto **"Sistema de Gestión de Boletos Aéreos"**, una aplicación web diseñada para administrar de forma integral los procesos de venta, reserva y gestión de vuelos comerciales.
El sistema está construido bajo una arquitectura **API REST** desarrollada en **Spring Boot (Java 17)** para el backend y **React + TailwindCSS** para el frontend.

El propósito del documento es servir como guía técnica para la instalación, configuración, mantenimiento y comprensión de los componentes del sistema, cumpliendo con los requisitos establecidos en la rúbrica institucional.

Objetivos

Objetivo general

Desarrollar un sistema de gestión de boletos aéreos basado en una API REST con **Java Spring Boot**, que automatice y optimice los procesos de registro de vuelos, aerolíneas, pasajeros y reservaciones, garantizando integridad de datos, trazabilidad y disponibilidad de información en tiempo real.

Objetivos específicos

- Diseñar una **base de datos relacional** para vuelos, pasajeros, reservas y pagos.
- Definir la **arquitectura de la API REST** con endpoints claros para cada proceso de negocio.
- Implementar una **interfaz moderna e intuitiva** con React y TailwindCSS.
- Asegurar la **seguridad y autenticación** mediante **JWT**.
- Documentar la API con **Swagger/OpenAPI**.

Descripción General del Sistema

El sistema gestiona todas las operaciones relacionadas con la venta de boletos aéreos:

- Administración de aerolíneas, vuelos, aviones y tripulación.
- Gestión de pasajeros, reservas, pagos y cancelaciones.
- Panel de control administrativo con estadísticas.
- Módulos de reclamos, reportes y autenticación por roles.

Se basa en una arquitectura **cliente-servidor**:

- **Frontend:** interfaz para usuarios y administradores (React + Tailwind).
- **Backend:** API REST en Java Spring Boot.
- **Base de datos:** MySQL 8.0.

Arquitectura del Proyecto

Estructura del Backend

Repositorio: https://github.com/pgap22/dwf-catedra

```
src/
 └── main/
    └── java/com/udb/aerovia_api/
       ├── config/       # Configuraciones generales (CORS, seguridad, JWT)
       ├── domain/       # Entidades del modelo (Usuario, Vuelo, Reserva, etc.)
       ├── dto/          # Objetos de transferencia de datos
       ├── exception/    # Manejadores globales de excepciones
       ├── mapper/       # Mapeos entre entidades y DTOs
       ├── repository/   # Interfaces JPA Repository
       ├── security/     # Implementación de Spring Security + JWT
       ├── service/      # Lógica de negocio
       └── web/          # Controladores REST (Endpoints)
```

Archivo principal:
AeroviaApiApplication.java — clase de arranque con @SpringBootApplication.

Estructura del Frontend

Repositorio: https://github.com/pgap22/dwf-frontend-catedra

```
src/
 ├── api/          # Configuración de Axios para conexión con la API
 ├── components/   # Componentes reutilizables (inputs, botones, modales)
 ├── pages/        # Páginas principales (Login, Dashboard, Reservas, etc.)
 ├── store/        # Manejo de estado global (Context o Zustand)
 ├── App.jsx       # Enrutamiento principal
 └── main.jsx      # Punto de entrada del proyecto
```

El proyecto utiliza **Vite** para construcción rápida y **TailwindCSS** para el diseño visual.

Herramientas y Tecnologías

Tipo	Herramienta / Versión
Backend	Java 17 + Spring Boot 3.5.5
Frontend	React 18 + TailwindCSS
Base de Datos	MySQL 8.0
IDE	IntelliJ IDEA (Backend) / VS Code (Frontend)
Pruebas	Postman, JUnit, MockMvc
Control de Versiones	GitHub
Diseño UI	Figma
Gestión BD	PhpMyAdmin
Documentación API	Swagger / OpenAPI

Base de Datos Relacional

El modelo relacional fue diseñado en base a los requerimientos del sistema y documentado en la **Fase I**.

Entidades principales

- **USUARIOS** → roles: Administrador, Cliente, Agente.
- **AEROLÍNEAS, AEROPUERTOS, RUTAS** → configuración de vuelos.
- **AVIONES, CLASES, ASIENTOS_AVION** → estructura operativa.
- **VUELOS, OPERACION_VUELO, TARIFAS, RESERVAS** → núcleo de negocio.
- **PAGOS, CANCELACIONES, RECLAMOS** → procesos de soporte.

Seguridad y Autenticación

El sistema implementa **Spring Security con JWT**:

- Los usuarios deben autenticarse mediante /auth/login.
- Los tokens se incluyen en el encabezado Authorization: Bearer <token>.
- Los roles determinan acceso a endpoints (admin / user).

✈️ Documentación Endpoints - Aerovía API v1

Base URL: /api/v1
Seguridad: Todos los endpoints (excepto /auth) requieren autenticación vía Token JWT (bearerAuth).

🔑 Autenticación (/auth)

Endpoints para registro e inicio de sesión.

Método	Endpoint	Descripción	Payload Requerido	Seguridad
POST	/login	Autentica un usuario (correo/contraseña) y devuelve JWT.	LoginRequestDto (correo, password)	Pública
POST	/register	Registra un nuevo usuario con rol CLIENTE.	RegisterRequestDto (nombre, correo, password)	Pública
GET	/me	Devuelve la información del usuario autenticado.	N/A	isAuthenticated()

Payloads Detallados:

- **POST /login** (LoginRequestDto)
  ```
  {
    "correo": "user@example.com", // Requerido, formato email
    "password": "your_password"   // Requerido
  }
  ```

- **POST /register** (RegisterRequestDto)
  ```
  {
    "nombre": "Nombre Apellido", // Requerido, min 3, max 150 chars
    "correo": "newuser@example.com", // Requerido, formato email
    "password": "password123"      // Requerido, min 8 chars
  }
  ```

🏢 Aerolíneas (/aerolineas)

Gestión del catálogo de aerolíneas.

- **Rol Requerido:** ADMIN (excepto consultas GET)

Método	Endpoint	Descripción	Payload Requerido	Seguridad
POST	/	Crear nueva aerolínea.	CreateAerolineaDto (nombre, codigoIata)	ADMIN
GET	/	Listar todas las aerolíneas.	N/A	isAuthenticated()
GET	/{id}	Consultar aerolínea por ID.	N/A	isAuthenticated()
PUT	/{id}	Actualizar aerolínea por ID.	CreateAerolineaDto (nombre, codigoIata)	ADMIN
DELETE	/{id}	Eliminar aerolínea por ID.	N/A	ADMIN

Payloads Detallados:

- **POST / y PUT /{id}** (CreateAerolineaDto)
  {
  "nombre": "Nombre Aerolínea", // Requerido, max 150 chars
  "codigoIata": "AA" // Requerido, 2-3 letras/números mayúsculas (ej. "AA", "AV", "IB", "UX", "AMX")
  }

🌍 Aeropuertos (/aeropuertos)

Gestión del catálogo de aeropuertos.

- **Rol Requerido:** ADMIN (excepto consultas GET)

Método	Endpoint	Descripción	Payload Requerido	Seguridad
POST	/	Crear nuevo aeropuerto.	CreateAeropuertoDto (codigoIata, nombre, ciudad, pais)	ADMIN
GET	/	Listar todos los aeropuertos.	N/A	isAuthenticated()
GET	/{id}	Consultar aeropuerto por ID.	N/A	isAuthenticated()
PUT	/{id}	Actualizar aeropuerto por ID.	CreateAeropuertoDto (codigoIata, nombre, ciudad, pais)	ADMIN
DELETE	/{id}	Eliminar aeropuerto por ID.	N/A	ADMIN

Payloads Detallados:

- **POST / y PUT /{id}** (CreateAeropuertoDto)
  {
  "codigoIata": "SAL", // Requerido, 3 letras mayúsculas
  "nombre": "Aeropuerto Internacional Monseñor Romero", // Requerido, max 150 chars
  "ciudad": "San Salvador", // Requerido, max 120 chars
  "pais": "El Salvador" // Requerido, max 120 chars
  }

✈️ Aviones (/aviones)

Gestión del inventario de aeronaves.

- **Rol Requerido:** ADMIN (excepto consultas GET)

Método	Endpoint	Descripción	Payload Requerido	Seguridad
POST	/	Crear nuevo avión.	CreateAvionDto (matricula, modelo, capacidadTotal)	ADMIN
GET	/	Listar todos los aviones.	N/A	isAuthenticated()
GET	/{id}	Consultar avión por ID.	N/A	isAuthenticated()
PUT	/{id}	Actualizar avión por ID.	CreateAvionDto (matricula, modelo, capacidadTotal)	ADMIN
DELETE	/{id}	Eliminar avión por ID.	N/A	ADMIN

Payloads Detallados:

- **POST / y PUT /{id}** (CreateAvionDto)
  ```
  {
    "matricula": "EC-MIA",     // Requerido, max 20 chars
    "modelo": "Airbus A350",   // Requerido, max 80 chars
    "capacidadTotal": 350      // Requerido, min 1
  }
  ```

💺 Asientos de Avión (/asientos-avion)

Configuración de la distribución de asientos por avión.

- **Rol Requerido:** ADMIN (excepto consultas GET)

Método	Endpoint	Descripción	Payload Requerido	Seguridad
POST	/	Crear un asiento para un avión.	CreateAsientoAvionDto (avionId, codigoAsiento, claseId)	ADMIN
GET	/avion/{avionId}	Listar asientos de un avión.	N/A	isAuthenticated()
GET	/{id}	Consultar asiento por ID.	N/A	isAuthenticated()
PUT	/{id}	Actualizar asiento por ID.	CreateAsientoAvionDto (avionId, codigoAsiento, claseId)	ADMIN
DELETE	/{id}	Eliminar asiento por ID.	N/A	ADMIN

Payloads Detallados:

- **POST / y PUT /{id}** (CreateAsientoAvionDto)
  ```
  {
    "avionId": 1,          // Requerido, ID del avión al que pertenece
    "codigoAsiento": "12A", // Requerido, max 10 chars (ej. "1A", "23C", "BUSINESS1")
    "claseId": 2           // Requerido, ID de la Clase (Economy, Business, etc.)
  }
  ```

🏷️ Clases (/clases)

Catálogo de clases de servicio (Economy, Business, etc.).

- **Rol Requerido:** ADMIN (excepto consultas GET)

Método	Endpoint	Descripción	Payload Requerido	Seguridad
POST	/	Crear nueva clase.	CreateClaseDto (nombre, descripcion?)	ADMIN
GET	/	Listar todas las clases.	N/A	isAuthenticated()
GET	/{id}	Consultar clase por ID.	N/A	isAuthenticated()
PUT	/{id}	Actualizar clase por ID.	CreateClaseDto (nombre, descripcion?)	ADMIN
DELETE	/{id}	Eliminar clase por ID.	N/A	ADMIN

Payloads Detallados:

- **POST / y PUT /{id}** (CreateClaseDto)
  ```
  {
    "nombre": "Economy Plus", // Requerido, max 50 chars
    "descripcion": "Asientos con espacio extra para piernas" // Opcional, max 200 chars
  }
  ```

🗺️ Rutas (/rutas)

Catálogo de rutas entre aeropuertos.

- **Rol Requerido:** ADMIN (excepto consultas GET)

Método	Endpoint	Descripción	Payload Requerido	Seguridad
POST	/	Crear nueva ruta.	CreateRutaDto (origenId, destinoId, distanciaKm)	ADMIN
GET	/	Listar todas las rutas.	N/A	isAuthenticated()
GET	/{id}	Consultar ruta por ID.	N/A	isAuthenticated()
PUT	/{id}	Actualizar ruta por ID.	UpdateRutaDto (origenId?, destinoId?, distanciaKm?)	ADMIN
DELETE	/{id}	Eliminar ruta por ID.	N/A	ADMIN

Payloads Detallados:

- **POST /** (CreateRutaDto)
  ```
  {
    "origenId": 1,     // Requerido, ID del Aeropuerto origen
    "destinoId": 2,    // Requerido, ID del Aeropuerto destino (diferente de origen)
    "distanciaKm": 3500 // Requerido, min 1
  }
  ```

- **PUT /{id}** (UpdateRutaDto) - *Campos opcionales*
  ```
  {
    "origenId": 3,     // Opcional, nuevo ID de Aeropuerto origen
    "destinoId": 4,    // Opcional, nuevo ID de Aeropuerto destino
    "distanciaKm": 3650 // Opcional, nueva distancia
  }
  ```

👨‍✈️ Tripulantes (/tripulantes)

Gestión del personal de vuelo.

- **Rol Requerido:** ADMIN (excepto GET que permiten AGENTE)

Método	Endpoint	Descripción	Payload Requerido	Seguridad
POST	/	Crear nuevo tripulante.	CreateTripulanteDto (nombre, tipo)	ADMIN
GET	/	Listar todos los tripulantes.	N/A	ADMIN, AGENTE
GET	/{id}	Consultar tripulante por ID.	N/A	ADMIN, AGENTE
PUT	/{id}	Actualizar tripulante por ID.	CreateTripulanteDto (nombre, tipo)	ADMIN
DELETE	/{id}	Eliminar tripulante por ID.	N/A	ADMIN

Payloads Detallados:

- **POST / y PUT /{id}** (CreateTripulanteDto)
  {
  "nombre": "Carlos Fuentes", // Requerido, max 150 chars
  "tipo": "PILOTO" // Requerido, max 50 chars (ej. "PILOTO", "COPILOTO", "SOBRECARGO")
  }

💵 Tarifas Base (/tarifas)

Definición de códigos de tarifa asociados a clases.

- **Rol Requerido:** ADMIN (excepto consultas GET)

Método	Endpoint	Descripción	Payload Requerido	Seguridad
POST	/	Crear nueva tarifa base.	CreateTarifaDto (codigo, claseId, reembolsable)	ADMIN
GET	/	Listar todas las tarifas base.	N/A	isAuthenticated()
GET	/{id}	Consultar tarifa base por ID.	N/A	isAuthenticated()
PUT	/{id}	Actualizar tarifa base por ID.	CreateTarifaDto (codigo, claseId, reembolsable)	ADMIN
DELETE	/{id}	Eliminar tarifa base por ID.	N/A	ADMIN

Payloads Detallados:

- **POST / y PUT /{id}** (CreateTarifaDto)

```
{
  "codigo": "YFLEX",       // Requerido, max 20 chars (código único)
  "claseId": 1,            // Requerido, ID de la Clase asociada
  "reembolsable": true     // Requerido, boolean
}
```

✈️ Vuelos Base (/vuelos)

Catálogo de vuelos (número de vuelo, aerolínea, ruta).

- **Rol Requerido:** ADMIN (excepto consultas GET)

Método	Endpoint	Descripción	Payload Requerido	Seguridad
POST	/	Crear un vuelo base.	CreateVueloDto (numeroVuelo, aerolineaId, rutaId, duracionMin)	ADMIN
GET	/	Listar todos los vuelos base.	N/A	isAuthenticated()
GET	/{id}	Consultar vuelo base por ID.	N/A	isAuthenticated()
DELETE	/{id}	Eliminar vuelo base por ID.	N/A	ADMIN

Payloads Detallados:

- **POST /** (CreateVueloDto)

```
{
  "numeroVuelo": "IB3456", // Requerido, formato AA1234
  "aerolineaId": 1,       // Requerido, ID de la Aerolínea
  "rutaId": 2,            // Requerido, ID de la Ruta
  "duracionMin": 480      // Requerido, min 1 (duración en minutos)
}
```

🗓️ Operaciones de Vuelo (/operaciones-vuelo)

Programación y consulta de instancias específicas de vuelos.

- **Rol Requerido:** ADMIN, AGENTE (excepto GET y /buscar)

Método	Endpoint	Descripción	Payload Requerido	Seguridad
POST	/	Programar nueva operación de vuelo.	CreateOperacionVueloDto (vueloId, avionId, fechaSalida, fechaLlegada, estado)	ADMIN, AGENTE
GET	/	Listar todas las operaciones.	N/A	isAuthenticated()
GET	/{id}	Consultar operación por ID.	N/A	isAuthenticated()
DELETE	/{id}	Eliminar operación por ID.	N/A	ADMIN, AGENTE
GET	/buscar	Buscar vuelos disponibles (params).	N/A - Usa Query Params (BusquedaVueloParamsDto)	isAuthenticated()

Payloads Detallados:

- **POST /** (CreateOperacionVueloDto)
  {
  "vueloId": 1, // Requerido, ID del Vuelo base
  "avionId": 2, // Requerido, ID del Avión asignado
  "fechaSalida": "2025-12-20T08:00:00", // Requerido, fecha/hora futura (ISO Format)
  "fechaLlegada": "2025-12-20T16:30:00", // Requerido, fecha/hora futura (posterior a salida)
  "estado": "PROGRAMADO" // Requerido, Enum: PROGRAMADO, EN_VUELO, ARRIBADO, CANCELADO
  }

- **GET /buscar** (Query Params - BusquedaVueloParamsDto)
  - origenId (Long, Requerido)
  - destinoId (Long, Requerido)
  - fechaSalida (LocalDate YYYY-MM-DD, Requerido, futura o presente)

🏷️ Tarifas por Operación (/tarifas-operacion)

Asignación de precios y cupos a tarifas para operaciones específicas.

- **Rol Requerido:** ADMIN, AGENTE (excepto GET)

Método	Endpoint	Descripción	Payload Requerido	Seguridad
POST	/	Asignar tarifa a operación.	CreateTarifaOperacionDto (operacionId, tarifaId, precio, asientosDisponibles)	ADMIN, AGENTE
GET	/operacion/{opId}	Listar tarifas de una operación.	N/A	isAuthenticated()
GET	/{id}	Consultar tarifa asignada por ID.	N/A	isAuthenticated()
PUT	/{id}	Actualizar precio/cupo.	CreateTarifaOperacionDto (operacionId,	ADMIN, AGENTE

			tarifaId, precio, asientosDisponibles)	
DELETE	/{id}	Eliminar/desasignar tarifa.	N/A	ADMIN, AGENTE

Payloads Detallados:

- **POST / y PUT /{id}** (CreateTarifaOperacionDto)
  ```
  {
    "operacionId": 1,          // Requerido, ID de OperacionVuelo
    "tarifaId": 2,             // Requerido, ID de Tarifa base
    "precio": 150.75,          // Requerido, número positivo (formato decimal)
    "asientosDisponibles": 100  // Requerido, número entero >= 0
  }
  ```

👨‍✈️ Tripulación por Operación (/operaciones-tripulacion)

Asignación de tripulantes a operaciones de vuelo.

- **Rol Requerido:** ADMIN, AGENTE

Método	**Endpoint**	**Descripción**	**Payload Requerido**	**Seguridad**
POST	/	Asignar tripulante a operación.	AsignarTripulanteDto (operacionId, tripulanteId, rolEnVuelo)	ADMIN, AGENTE
DELETE	/{id}	Desasignar tripulante.	N/A	ADMIN, AGENTE
GET	/operacion/{opId}	Listar tripulación asignada.	N/A	ADMIN, AGENTE

Payloads Detallados:

- **POST /** (AsignarTripulanteDto)
  {
  "operacionId": 1, // Requerido, ID de OperacionVuelo
  "tripulanteId": 5, // Requerido, ID del Tripulante
  "rolEnVuelo": "PILOTO" // Requerido, max 50 chars (ej. "PILOTO", "SOBRECARGO PRINCIPAL")
  }

🧳 Reservas (/reservas)

Gestión de reservas de vuelos.

Método	Endpoint	Descripción	Payload Requerido	Seguridad
POST	/	Crear nueva reserva.	CreateReservaDto (operacionVueloId, items[])	isAuthenticated()
GET	/	Listar reservas (ADMIN).	N/A - Usa Query Params	ADMIN
PATCH	/{codigoReserva}	Modificar reserva (asientos, etc.).	UpdateReservaDto (cambios[])	isAuthenticated()
GET	/{codigoReserva}	Consultar detalle por código.	N/A	isAuthenticated()
GET	/mis-reservas	Listar reservas del cliente autenticado.	N/A	CLIENTE

Payloads Detallados:

- **POST /** (CreateReservaDto)

```
{
  "operacionVueloId": 1, // Requerido, ID de la OperacionVuelo
  "items": [            // Requerido, al menos 1 item
    {
      "pasajeroId": 10,          // Requerido
      "asientoAvionId": 25,      // Requerido
      "tarifaOperacionId": 3     // Requerido
    }
    // ... más items si es reserva múltiple
  ]
}
```

- **PATCH /{codigoReserva}** (UpdateReservaDto)

```
{
  "cambios": [ // Requerido, al menos 1 cambio
    {
      "reservaAsientoId": 12,     // Requerido, ID del asiento reservado a modificar
      "pasajeroId": 11,          // Opcional, nuevo pasajero
      "asientoAvionId": 28,      // Opcional, nuevo asiento
      "tarifaOperacionId": 4     // Opcional, nueva tarifa
    }
    // ... más cambios
  ]
}
```

- **GET /** (Admin Query Params Opcionales): page, size, sort, direction, estado, fechaDesde, fechaHasta, aerolineaId, origenId, destinoId.

💳 Pagos (/pagos)

Registro y consulta de pagos asociados a reservas.

- **Rol Requerido:** ADMIN, AGENTE (excepto GET / que es solo ADMIN)

Método	Endpoint	Descripción	Payload Requerido	Seguridad
POST	/	Registrar pago.	CreatePagoDto (reservaId, fechaPago, metodoPago, monto)	ADMIN, AGENTE

GET	/	Listar pagos (ADMIN).	N/A - Usa Query Params	ADMIN
GET	/reserva/{reservaId}	Listar pagos de una reserva.	N/A	ADMIN, AGENTE
GET	/{id}	Consultar pago por ID.	N/A	ADMIN, AGENTE

Payloads Detallados:

- **POST /** (CreatePagoDto)
  {
  "reservaId": 1, // Requerido
  "fechaPago": "2025-10-23T10:15:30", // Requerido, fecha/hora (ISO Format)
  "metodoPago": "Tarjeta VISA", // Requerido, max 40 chars
  "monto": 150.75 // Requerido, número positivo (decimal)
  }

- **GET /** (Admin Query Params Opcionales): page, size, sort, direction, reservaId, codigoReserva, fechaDesde, fechaHasta.

⚠️ Cancelaciones (/cancelaciones)

Gestión del proceso de cancelación de reservas.

Método	Endpoint	Descripción	Payload Requerido	Seguridad
POST	/	Cancelar una reserva.	CreateCancelacionDto (reservaId)	isAuthenticated()
GET	/reserva/{reservaId}	Obtener cancelación por reserva ID.	N/A	isAuthenticated()
GET	/{id}	Consultar cancelación por ID.	N/A	ADMIN, AGENTE
GET	/	Listar todas las cancelaciones.	N/A	ADMIN, AGENTE

Payloads Detallados:

- **POST /** (CreateCancelacionDto)
  ```
  {
    "reservaId": 1 // Requerido, ID de la Reserva a cancelar
  }
  ```

👤 Pasajeros (/pasajeros)

Catálogo de pasajeros registrados.

- **Rol Requerido:** ADMIN, AGENTE (excepto DELETE que es solo ADMIN)

Método	Endpoint	Descripción	Payload Requerido	Seguridad
POST	/	Crear nuevo pasajero.	CreatePasajeroDto (nombreCompleto, fechaNacimiento, nroPasaporte?)	ADMIN, AGENTE
GET	/	Listar todos los pasajeros.	N/A	ADMIN, AGENTE
GET	/{id}	Consultar pasajero por ID.	N/A	ADMIN, AGENTE
PUT	/{id}	Actualizar pasajero por ID.	CreatePasajeroDto (nombreCompleto, fechaNacimiento, nroPasaporte?)	ADMIN, AGENTE
DELETE	/{id}	Eliminar pasajero por ID.	N/A	ADMIN

Payloads Detallados:

- **POST / y PUT /{id}** (CreatePasajeroDto)
  ```
  {
    "nombreCompleto": "Juan Perez Garcia", // Requerido, max 150 chars
    "fechaNacimiento": "1990-05-15",     // Requerido, fecha pasada (YYYY-MM-DD)
    "nroPasaporte": "A12345678"          // Opcional, max 30 chars
  }
  ```

📣 Reclamos (/reclamos)

Gestión de reclamos de pasajeros sobre reservas.

Método	Endpoint	Descripción	Payload Requerido	Seguridad
POST	/	Crear reclamo.	CreateReclamoDto (reservaId, pasajeroId, descripcion)	isAuthenticated()
GET	/reserva/{reservaId}	Listar reclamos de una reserva.	N/A	isAuthenticated()
GET	/pasajero/{pasajeroId}	Listar reclamos de un pasajero.	N/A	isAuthenticated()
GET	/{id}	Consultar reclamo por ID.	N/A	isAuthenticated()
PATCH	/{id}/estado	Actualizar estado del reclamo.	UpdateReclamoEstadoDto (estado)	ADMIN, AGENTE

Payloads Detallados:

- **POST /** (CreateReclamoDto)
  {
  "reservaId": 1, // Requerido
  "pasajeroId": 10, // Requerido
  "descripcion": "El vuelo se retrasó 3 horas sin aviso." // Requerido, max 500 chars
  }

- **PATCH /{id}/estado** (UpdateReclamoEstadoDto)
  {
  "estado": "EN_PROCESO" // Requerido, Enum: ABIERTO, EN_PROCESO, CERRADO
  }

👥 Usuarios (/usuarios) - Administración

Gestión de usuarios del sistema (roles, estado).

- **Rol Requerido:** ADMIN

Método	Endpoint	Descripción	Payload Requerido	Seguridad
POST	/	Crear usuario (ADMIN/AGENTE).	CreateUserAdminDto (nombre, correo, password, rol, activo)	ADMIN
GET	/	Listar usuarios (filtros).	N/A - Usa Query Params	ADMIN
GET	/{id}	Consultar usuario por ID.	N/A	ADMIN
PUT	/{id}	Actualizar usuario por ID.	UpdateUserAdminDto (nombre?, correo?, password?, rol?)	ADMIN
PATCH	/{id}/activar	Activar	N/A	ADMIN

		usuario.		
PATCH	/{id}/desactivar	Desactivar usuario.	N/A	ADMIN
DELETE	/{id}	Eliminar usuario por ID.	N/A	ADMIN

Payloads Detallados:

- **POST /** (CreateUserAdminDto)
  ```
  {
    "nombre": "Agente Ventas",
    "correo": "agente@aerovia.test",
    "password": "Password123!", // Min 8 chars
    "rol": "AGENTE",          // Enum: ADMIN, AGENTE, CLIENTE
    "activo": true            // Boolean
  }
  ```

- **PUT /{id}** (UpdateUserAdminDto) - *Campos opcionales*
  ```
  {
    "nombre": "Agente Ventas Actualizado",
    "correo": "agente.ventas@aerovia.test",
    "password": "NewSecurePassword!", // Solo si se cambia
    "rol": "AGENTE"
  }
  ```

- **GET /** (Admin Query Params Opcionales): page, size, sort, direction, rol, activo.

📊 Estadísticas (/estadisticas)

Indicadores operativos del sistema.

- **Rol Requerido:** ADMIN, AGENTE

Método	**Endpoint**	**Descripción**	**Payload Requerido**	**Seguridad**
GET	/	Obtener estadísticas básicas del sistema.	N/A	ADMIN, AGENTE

Los endpoints están documentados con **Swagger UI**, accesible desde:
👉 http://localhost:8080/swagger-ui.html

Dependencias y Versiones (pom.xml)

- spring-boot-starter-data-jpa
- spring-boot-starter-security
- spring-boot-starter-web
- jjwt-api / jjwt-impl / jjwt-jackson
- mysql-connector-j
- springdoc-openapi-ui
- lombok
- spring-boot-starter-test

Validaciones del Servidor

- Validación de campos vacíos (@NotNull, @NotBlank)
- Tipos de dato correctos (regex, fechas, enteros)
- Restricciones lógicas:
    - Fechas de vuelo ≥ fecha actual
    - Edad > 0
    - Códigos IATA de 2–3 caracteres

Usuarios de Prueba

Rol	Usuario	Contraseña
Administrador	admin@aerovia.test	123
Cliente	cliente@aerovia.test	123

Ejemplo de Código

Endpoint REST – Spring Boot

```
@RestController
@RequestMapping("/api/vuelos")
public class VueloController {

    @Autowired
    private VueloService vueloService;

    @GetMapping("/{id}")
    public ResponseEntity<Vuelo> getVueloById(@PathVariable Long id) {
        Vuelo vuelo = vueloService.findById(id);
        return ResponseEntity.ok(vuelo);
    }

    @PostMapping
    public ResponseEntity<Vuelo> createVuelo(@RequestBody Vuelo vuelo) {
        Vuelo nuevoVuelo = vueloService.save(vuelo);
        return ResponseEntity.status(HttpStatus.CREATED).body(nuevoVuelo);
    }
}
```

Prueba unitaria (JUnit)

```
@SpringBootTest
class VueloServiceTest {
   @Autowired
   private VueloService vueloService;

   @Test
   void testCalcularDuracionVuelo() {
      int duracion = vueloService.calcularDuracion(14, 16);
      assertEquals(120, duracion);
   }
}
```

Repositorios y Recursos

- **Backend:** https://github.com/pgap22/dwf-catedra
- **Frontend:** https://github.com/pgap22/dwf-frontend-catedra
- **Modelo ER y prototipos:**. https://viewer.diagrams.net/?tags=%7B%7D&lightbox=1&highlight=0000ff&edit=_blank&layers=1&nav=1&title=DWF-T.drawio&dark=auto#Uhttps%3A%2F%2Fdrive.google.com%2Fuc%3Fid%3D1hrYJrleNusqMvMgfG37ufbN0AEH5yvWA%26export%3Ddownload

Conclusiones

El proyecto **Aerovía API** demuestra la aplicación práctica de arquitecturas modernas en el desarrollo de sistemas distribuidos, garantizando seguridad, escalabilidad y mantenimiento.

La integración entre **Spring Boot** y **React** permitió una experiencia fluida para el usuario final y una estructura modular para el equipo de desarrollo.

La documentación completa asegura la continuidad y sostenibilidad del sistema en futuras iteraciones.