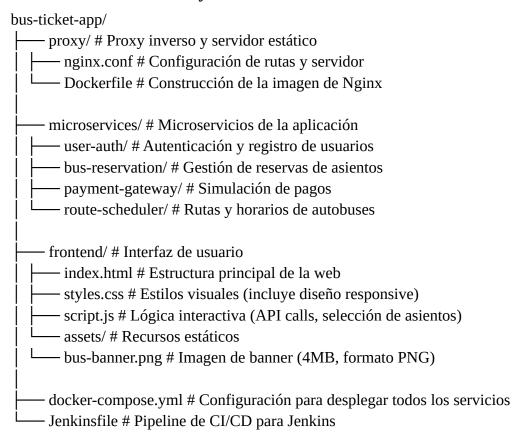
Estructura de Ficheros y su Función



2. Funcionalidad del Proxy (Nginx)

- Enrutamiento de APIs:
 - /api/auth/* → Microservicio user-auth (puerto 5001).
 - /api/reservation/* → Microservicio bus-reservation (puerto 5002).
 - /api/payment/* → Microservicio payment-gateway (puerto 5003).
 - /api/routes/* → Microservicio route-scheduler (puerto 5004).
- **Servidor Estático:** Sirve los archivos HTML/CSS/JS desde frontend/.
- **MIME Types:** Asegura que los archivos CSS/JS se sirvan con encabezados correctos.

3. Funcionalidad de Microservicios y APIs

user-auth (puerto 5001)

- Registro de Usuario:
 - Endpoint: POST /api/auth/register
 - Parámetros: username, password, email.
 - Respuesta: {"message": "User registered successfully"}.
- Login:
 - Endpoint: POST /api/auth/login
 - Parámetros: username, password.
 - Respuesta: {"token": "JWT"} (si es válido).

bus-reservation (puerto 5002)

- Consultar Disponibilidad:
 - Endpoint: GET /api/reservation/availability/{bus_id}/{date}
 - Respuesta: {"available_seats": [1, 2, ..., 40]}.
- Reservar Asiento:
 - Endpoint: POST /api/reservation/reserve
 - Parámetros: bus_id, seat_number, date.
 - Respuesta: {"message": "Seat reserved successfully"}.

payment-gateway (puerto 5003)

- Procesar Pago:
 - Endpoint: POST /api/payment/pay
 - Parámetros: amount, card_number, expiry_date, cvv.
 - Respuesta: {"message": "Payment successful", "transaction_id": "12345"}.

route-scheduler (puerto 5004)

- Listar Rutas:
 - Endpoint: GET /api/routes/routes

• Respuesta: Array de rutas con id, origin, destination.

• Horarios por Ruta:

• Endpoint: GET /api/routes/schedules/{route_id}

• Respuesta: {"schedules": ["08:00", "22:00"]}.

4. Comandos curl para Pruebas

```
# Registro de usuario
curl -X POST http://localhost:8080/api/auth/register \
-H "Content-Type: application/json" \
-d'{"username": "user1", "password": "pass123", "email": "user1@example.com"}'
# Login
curl -X POST http://localhost:8080/api/auth/login \
-H "Content-Type: application/json" \
-d'{"username": "user1", "password": "pass123"}'
# Listar rutas
curl -H "Authorization: Bearer <token>" http://localhost:8080/api/routes/routes
# Ver disponibilidad de asientos (bus_id=1, fecha=2023-10-05)
curl -H "Authorization: Bearer <token>" http://localhost:8080/api/reservation/availability/1/2023-
10-05
# Reservar asiento (bus_id=1, seat=5, fecha=2023-10-05)
curl -X POST http://localhost:8080/api/reservation/reserve \
-H "Content-Type: application/json" \
-H "Authorization: Bearer <token>" \
-d'{"bus_id": 1, "seat_number": 5, "date": "2023-10-05"}'
# Procesar pago
curl -X POST http://localhost:8080/api/payment/pay \
-H "Content-Type: application/json" \
-H "Authorization: Bearer <token>" \
-d'{"amount": 43, "card_number": "41111111111111", "expiry_date": "12/25", "cvv": "123"}'
```

5. Despliegue con Docker Compose

```
# 1. Construir imágenes
docker compose build# 2. Levantar servicios
docker compose up -d# 3. Verificar contenedores
docker ps
```

docker compose down

6. Jenkinsfile (Pipeline Declarativo)

```
pipeline {
  agent any
  stages {
    stage('Build') {
       steps {
         sh 'docker compose build'
     }
    stage('Test') {
       steps {
         sh 'docker compose run --rm user-auth pytest'
         sh 'docker compose run --rm bus-reservation pytest'
         sh 'docker compose run --rm payment-gateway pytest'
         sh 'docker compose run --rm route-scheduler pytest'
       }
     }
     stage('Deploy') {
       steps {
         sh 'docker compose up -d'
     }
     stage('Cleanup') {
       steps {
         sh 'docker system prune -f'
     }
  }
```