LAB 6 REPORT: API GATEWAY PATTERN

Project Name: Movie Ticket Booking System

Member Name: Nguyễn Đình Quyền - 23010485

Trần Hữu Kiên - 23010258

1. OBJECTIVES

The goal of this lab is to implement an **API Gateway** as the single entry point for the microservices ecosystem. The Gateway is responsible for:

- **Routing:** Forwarding client requests to the appropriate backend service (Movie Service).
- **Security:** Enforcing a centralized authentication check before requests reach the backend.
- **Abstraction:** Hiding the internal architecture (ports and IPs) from the client.

2. PROJECT STRUCTURE

We added a new component, `api_gateway`, which sits in front of the `movie_service`.

- **Gateway:** Runs on Port **5000**.
- **Movie Service:** Runs on Port **5001** (Backend).

[INSERT SCREENSHOT OF YOUR FOLDER STRUCTURE SHOWING BOTH FOLDERS] *(Chụp ảnh cây thư mục có cả `api_gateway` và `movie_service`)*

3. IMPLEMENTATION DETAILS

3.1. Technology Stack

- **Runtime:** Node.js
- **Libraries:** `express` (Server), `axios` (HTTP Client for forwarding requests).

3.2. Source Code (`gateway.js`)

The following code implements the Reverse Proxy logic. It intercepts requests to `/api/movies`, validates the security token, and forwards valid requests to the internal Movie Service.

JavaScript

```
const express = require('express');
const axios = require('axios');
const app = express();
```

```
const PORT = 5000; // Gateway Port
const MOVIE_SERVICE_URL = 'http://localhost:5001'; // Internal Backend URL

app.use(express.json());

// 1. SECURITY CHECK STUB
// Centralized authentication logic
const validateToken = (req) => {
    const authHeader = req.headers['authorization'];
    if (!authHeader) return false;
    const token = authHeader.split(' ')[1];
    // Simple whitelist check for demonstration
    return token === 'valid-user-token' || token === 'valid-admin-token';
};

// 2. ROUTING & REVERSE PROXY
app.all('/api/movies*', async (req, res) => {
    // Security Guard: Block invalid requests here
    if (!validateToken(req)) {
        return res.status(401).json({ error: "Unauthorized access" });
    }

    // Forwarding to Backend
    const targetUrl = `${MOVIE_SERVICE_URL}${req.originalUrl}`;

    try {
        const response = await axios({
            method: req.method,
            url: targetUrl,
            data: req.body
        });
        res.status(response.status).json(response.data);
    } catch (error) {
        // Handle Service Downtime
        res.status(503).json({ error: "Movie Service Unavailable" });
    }
});

app.listen(PORT, () => {
    console.log(`API Gateway listening on port ${PORT}`);
});
```

---

4. CONCLUSION

We have successfully implemented the **API Gateway Pattern**. The Gateway now acts as a protective shield and a unified interface for the Movie Ticket Booking System, satisfying the security and routing requirements of Lab 6.