LAB 5 REPORT: IMPLEMENTING THE MOVIE MICROSERVICE

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 extract the **Movie Management** functionality from the existing monolithic architecture and implement it as a standalone **Microservice**. This service complies with the following architectural principles:

- **Service Independence:** The service runs on a dedicated port (**5001**), completely decoupled from the main Booking System (which runs on port 3000).
- **Data Ownership:** The service manages its own dedicated database (**SQLite**), ensuring strict data isolation and eliminating shared data files.
- **Service Contract:** It exposes standardized RESTful APIs for creating and retrieving movie data.

2. PROJECT STRUCTURE

To achieve isolation, the new service is placed in a separate directory named `movie_service`.

- **Root Directory:** `movie_service/`
- **Entry Point:** `server.js` (Encapsulates Routes, Controller, and Repository logic)
- **Database:** `movies.db` (Auto-generated SQLite file for this service only)
- **Dependencies:** `express`, `sqlite3`

[INSERT SCREENSHOT OF YOUR `movie_service` FOLDER HERE] *(Chụp ảnh cây thư mục `movie_service` mới tạo)*

3. IMPLEMENTATION DETAILS

3.1. Database Schema (SQLite)

We migrated the data model from the shared JSON file to a relational SQLite table. The schema supports the required movie attributes based on the original business logic:

- `id`: INTEGER PRIMARY KEY (Auto-increment)

- `title`: TEXT
- `poster`: TEXT
- `duration`: TEXT
- `price`: INTEGER

3.2. Source Code (`server.js`)

The following code implements the microservice. It consolidates the logic from the previous `movieController.js` and `movieRepo.js` into a single cohesive unit, replacing file I/O with SQL queries.

JavaScript

```
// File: movie_service/server.js
const express = require('express');
const sqlite3 = require('sqlite3').verbose();
const app = express();
const PORT = 5001; // Dedicated port for isolation (Lab 5 Requirement)

// Middleware to parse JSON bodies
app.use(express.json());

// 1. DATA OWNERSHIP: Connect to independent database
// This replaces the shared 'movies.json' file access
const db = new sqlite3.Database('./movies.db', (err) => {
    if (err) console.error("DB Error:", err.message);
    else console.log("Connected to the independent 'movies.db' database.");
});

// 2. SCHEMA SETUP & SEEDING (Data Migration)
db.serialize(() => {
    // Create table if not exists
    db.run(`CREATE TABLE IF NOT EXISTS movies (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        title TEXT,
        poster TEXT,
        duration TEXT,
        price INTEGER
    )`);

    // Seed sample data if table is empty
    db.get("SELECT count(*) as count FROM movies", (err, row) => {
        if (row.count === 0) {
            const stmt = db.prepare("INSERT INTO movies (title, poster,
duration, price) VALUES (?, ?, ?, ?)");
            stmt.run("Mai", "mai.jpg", "131 mins", 120000);
            stmt.run("Dao, Pho va Piano", "dao.jpg", "100 mins", 100000);
            stmt.run("Dune: Part Two", "dune2.jpg", "166 mins", 150000);
            stmt.finalize();
            console.log("Database initialized with sample data.");
        }
    });
});

// 3. SERVICE API (REST Contract)
```

```
// GET /api/movies - List all movies
app.get('/api/movies', (req, res) => {
    db.all("SELECT * FROM movies", [], (err, rows) => {
        if (err) return res.status(500).json({ error: err.message });
        res.json(rows);
    });
});

// GET /api/movies/:id - Get movie details
app.get('/api/movies/:id', (req, res) => {
    const id = req.params.id;
    db.get("SELECT * FROM movies WHERE id = ?", [id], (err, row) => {
        if (err) return res.status(500).json({ error: err.message });
        if (!row) return res.status(404).json({ error: "Movie not found" });
        res.json(row);
    });
});

// POST /api/movies - Add new movie
app.post('/api/movies', (req, res) => {
    const { title, poster, duration, price } = req.body;

    // Validation
    if (!title || !price) {
        return res.status(400).json({ error: "Title and Price are required"
});
    }

    const sql = "INSERT INTO movies (title, poster, duration, price) VALUES
(?, ?, ?, ?)";
    db.run(sql, [title, poster, duration, price], function(err) {
        if (err) return res.status(500).json({ error: err.message });
        res.json({
            message: "Movie added successfully!",
            movie: { id: this.lastID, title, poster, duration, price }
        });
    });
});

// 4. START SERVER
app.listen(PORT, () => {
    console.log(`Movie Microservice is running on http://localhost:${PORT}`);
});
```

4. CONCLUSION

We have successfully transitioned the **Movie Management** logic from the monolithic codebase into a standalone microservice. It owns its data, runs on its own process, and communicates via a standardized REST API, fully satisfying the requirements of Lab 5.