CASE STUDY 2

Online Movie Ticket Booking REST API – Development & Testing

Problem Statement

A cinema chain wants to build a RESTful backend service for an **Online Movie Ticket Booking System**.
The system should allow users to view movies, check show timings, and book tickets.
All APIs must be **manually tested** and **automated**.

Objectives

Students should be able to:

- Understand REST API architecture
- Build REST APIs using Flask
- Design CRUD-based endpoints
- Test APIs using Postman
- Automate API tests using `requests` & `pytest`
- Apply unit testing and TDD concepts

Functional Requirements

1. REST API Design

Follow REST principles:

- Stateless communication
- Resource-based URLs
- JSON request/response
- Proper HTTP status codes

---

2. API Endpoints to Implement (Flask)

HTTP Verb	Endpoint	Description
GET	/api/movies	Retrieve all movies
GET	/api/movies/	Get movie by ID
POST	/api/movies	Add a new movie
PUT	/api/movies/	Update movie details
DELETE	/api/movies/	Delete a movie
POST	/api/bookings	Book movie tickets

---

3. Sample Movie Data (JSON)

```
{
  "id": 101,
  "movie_name": "Interstellar",
  "language": "English",
  "duration": "2h 49m",
  "price": 250
}
```

Testing Requirements

□ Manual API Testing (Postman)

Students must:

- Perform GET, POST, PUT, DELETE requests
- Use headers: `Content-Type: application/json`
- Validate:
    - Status codes (200, 201, 400, 404)
    - Response payload
    - Booking failure scenarios

□ Automated API Testing (Python)

Use:

- `requests` library
- `pytest`

Test Scenarios:

- Fetch all movies
- Add a new movie

- Book tickets
- Validate JSON responses
- Assert HTTP status codes

Automation Framework Concepts Applied

- Pytest test discovery
- Fixtures for setup/teardown
- Parameterized tests
- Assertions & exception handling
- Command-line execution
- HTML test reports
