**Project Problem Statement:   
Movie Booking Portal**

**1. Background**

In today’s digital era, online ticket booking systems have become the backbone of the entertainment industry. People prefer booking movie tickets online instead of standing in long queues. However, many booking platforms suffer from issues such as limited search/filtering options, lack of real-time seat availability, and poor testing reliability.

To overcome these challenges, the **Movie Booking Portal** project aims to design and develop a **robust, scalable, and testable system** using **Core Java (backend logic), PostgreSQL (database management), and Selenium (automated testing)**.

Hence, the need is to **develop a Movie Booking Portal** that will allow users to:

* Browse available movies by city, theatre, genre, or showtime
* View real-time seat availability
* Select and book seats securely
* Make payments (dummy payment gateway integration)
* Receive booking confirmation

At the same time, the system should allow **Admin users** to:

* Add/Edit/Delete movies, theatres, and show timings
* Monitor bookings and generate reports

**2. Objectives**

The **main objectives** of the Movie Booking Portal are:

1. Implement **Core Java** for backend logic (OOP concepts, exception handling, JDBC).
2. Store and manage persistent data (movies, users, bookings, payments) in **PostgreSQL**.
3. Ensure **seat availability and booking concurrency handling**.
4. Provide **role-based access** (User vs. Admin).
5. Automate functional and regression testing using **Selenium**.
6. Maintain modularity, scalability, and reliability of the system.

**3. System Requirements**

**Functional Requirements**

* **User Module**
  + User Registration & Login
  + Browse/Search Movies (by city, genre, language, theatre)
  + Select Show & Seats
  + Book Tickets (dummy payment integration)
  + View Booking History
* **Admin Module**
  + Admin Login
  + Add/Update/Delete Movie Details
  + Add/Update/Delete Theatre & Show Timings
  + View Total Bookings & Reports
* **Booking Module**
  + Real-time seat selection & availability check
  + Prevent double-booking with concurrency handling
  + Booking confirmation generation (Ticket ID, Seat No., Show Time)
* **Testing Module (Selenium)**
  + Automated test cases for **login, movie search, seat booking, and admin functionalities**
  + Regression testing for every update
  + Cross-browser compatibility testing

**4. Technology Stack**

* **Frontend (basic UI):** HTML (optional for GUI)
* **Backend:** Core Java (Collections, JDBC, Multithreading, Exception Handling)
* **Database:** PostgreSQL (Movies, Theatres, Users, Bookings, Payments tables)
* **Testing:** Selenium WebDriver (UI and functional testing)

**5. Expected Deliverables**

* Fully functional **Movie Booking Portal**
* Well-structured **database schema** in PostgreSQL
* **Core Java modules** (User, Admin, Booking, Payment simulation)
* Automated **Selenium test suite**
* Documentation: SRS, System Design, Test Cases, and User Guide

### ****Tables****

#### **1. Users Table**

CREATE TABLE users (

user\_id SERIAL PRIMARY KEY,

username VARCHAR(50) UNIQUE NOT NULL,

password VARCHAR(100) NOT NULL,

email VARCHAR(100) UNIQUE NOT NULL,

phone VARCHAR(15),

role VARCHAR(10) CHECK (role IN ('USER', 'ADMIN')) DEFAULT 'USER'

);

#### **2. Movies Table**

CREATE TABLE movies (

movie\_id SERIAL PRIMARY KEY,

title VARCHAR(100) NOT NULL,

genre VARCHAR(50),

language VARCHAR(30),

duration INT, -- in minutes

rating DECIMAL(2,1)

);

#### **3. Theatres Table**

CREATE TABLE theatres (

theatre\_id SERIAL PRIMARY KEY,

name VARCHAR(100) NOT NULL,

city VARCHAR(50) NOT NULL,

total\_seats INT NOT NULL

);

#### **4. Shows Table**

CREATE TABLE shows (

show\_id SERIAL PRIMARY KEY,

movie\_id INT REFERENCES movies(movie\_id) ON DELETE CASCADE,

theatre\_id INT REFERENCES theatres(theatre\_id) ON DELETE CASCADE,

show\_time TIMESTAMP NOT NULL,

available\_seats INT

);

#### **5. Bookings Table**

CREATE TABLE bookings (

booking\_id SERIAL PRIMARY KEY,

user\_id INT REFERENCES users(user\_id) ON DELETE CASCADE,

show\_id INT REFERENCES shows(show\_id) ON DELETE CASCADE,

seats\_booked INT NOT NULL,

booking\_time TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

);