Project Requirement Document (PRD)

Online Movie Ticket Booking System

Document Version: 1.0

Date: March 31, 2025

Prepared by: Praniga S, Sreejitha S, Archana K

Organization: [TechM/Knowledge Institute of technology]

---

1. Introduction

1.1 Purpose

The purpose of this document is to outline the functional and non-functional requirements for the "Online Movie Ticket Booking" system. This web-based application aims to automate the movie ticket booking process, replacing manual methods with a convenient, efficient, and user-friendly platform. The system will provide real-time movie listings, showtimes, and seat selection capabilities for customers while offering administrative tools for managing movies, theatres, and user interactions.

1.2 Scope

The "Online Movie Ticket Booking" system will enable:

- Customers to search, view, and book movie tickets online.
- Administrators manage movies, showtimes, and customer data.
- Guest users to browse movie listings without registration. The system will be developed using Spring Boot for the backend, Spring MVC and Thymeleaf for the frontend, MySQL Workbench for database management, and Postman for API testing.

1.3 Objectives

- Provide a seamless ticket booking experience with real-time updates.
- Support multiple user roles: Administrator, Customer, and Guest.
- Ensure secure authentication, payment processing, and data management.

- Deliver a scalable and maintainable solution using modern web technologies.

---

2. System Overview

2.1 Problem Definition

The manual process of booking movie tickets is time-consuming and prone to errors. The "Online Movie Ticket Booking" system addresses this by offering an automated platform where users can view movie schedules, select seats, and complete transactions online, reducing dependency on physical counters and improving operational efficiency.

2.2 User Roles

1. **Administrator**: Manages movies, showtimes, theatres, and customer interactions.
2. **Customer**: Registered users who can book, view, and cancel tickets.
3. **Guest**: Unregistered users who can search and view movie listings.

---

3. Functional Requirements

3.1 Modules and Features

3.1.1 Administrator Module

- **Login**
    - Secure authentication for administrators.
    - Function: login()
- **Movie Management**
    - Add new movies with details (title, language, genre, release date, etc.).
        - Function: addMovie()
    - Update existing movie details for specific theatres.
        - Function: updateMovie()
    - Delete movies from the system.
        - Function: deleteMovie()
- **Profile Management**
    - Register administrators.
        - Function: registration()
    - Update personal details.
        - Function: update()
- **Report Management**
    - Retrieve a list of currently running movies.

        - Function: movieDetails()
    - Obtain a list of registered customers.
        - Function: customerDetails()

3.1.2 Customer Module

- **Login**
    - Secure authentication for registered customers.
        - Function: login()
- **Profile Module**
    - Register as a customer for ticket booking.
        - Function: registration()
    - View personal details.
        - Function: view()
    - Modify personal details.
        - Function: update()
- **Booking Module**
    - Book movie tickets online.
        - Function: bookTickets()
    - View the list of available movies.
        - Function: viewMovieList()
    - Search movies by title, language, genre, theatre, or city.
        - Function: searchMovie()
    - Cancel booked tickets.
        - Function: cancelTickets()
- **Payment Module**
    - Facilitate payment via credit/debit cards.
        - Function: creditDebitCardPayment()
    - Allow cash payment at the theatre.
        - Function: cashPayment()
    - Support the UPI payment option.
        - Function: upiPayment()

3.1.3 Guest Module

- **Search and View**
    - Search for movies by title, language, genre, theatre, or city.
        - Function: searchMovie()
    - View the list of available movies.
        - Function: viewMovieList()

3.2 Core Features

1. Display cities with selected theatres.
2. Manage multiple halls per theatre, with one movie show per hall at a time.
3. Organize multiple shows for each movie.

4. Provide details on the theatre and showtime for selected movies.
5. Enable seat selection with a visual seating arrangement (available/booked seats).
6. Support multiple payment options: credit/debit cards, cash, UPI.

---

4. Non-Functional Requirements

4.1 Performance

- The system should handle up to 1,000 concurrent users without significant latency.
- Page load times should not exceed 3 seconds under normal conditions.

4.2 Security

- Secure user authentication using encrypted passwords (e.g., BCrypt).
- HTTPS for all transactions and sensitive data transfers.
- Role-based access control for administrators and customers.

4.3 Usability

- Intuitive and responsive UI compatible with desktops and mobile devices.
- Clear error messages and user guidance.

4.4 Scalability

- The system should support the addition of new theatres, movies, and users without requiring major refactoring.

4.5 Reliability

- 99.9% uptime with minimal downtime for maintenance.

---

5. Technology Stack

5.1 Backend

- **Framework**: Spring Boot
- **Purpose**: RESTful API development, business logic, and database integration.

5.2 Frontend

- **Framework**: Spring MVC with Thymeleaf

- **Purpose**: Server-side rendering of dynamic web pages.

5.3 Database

- **Tool**: MySQL Workbench
- **Purpose**: Schema design, data storage, and management.

5.4 Testing

- **Tool**: Postman
- **Purpose**: API testing and validation.

5.5 Development Environment

- **IDE**: Eclipse
- **Version Control**: Git (recommended)

---

6. Development Roadmap

6.1 Steps to Proceed

1. **Project Setup**
    - Install Eclipse, MySQL Workbench, and Postman.
    - Create a new Spring Boot project in Eclipse using Spring Initializr.
    - Configure dependencies: Spring Web, Spring Data JPA, Thymeleaf, MySQL Driver.
2. **Database Design**
    - Design tables in MySQL Workbench:
        - users (for admins and customers: id, username, password, role, etc.).
        - movies (id, title, language, genre, release_date, etc.).
        - theatres (id, name, city, etc.).
        - halls (id, theatre_id, hall_number, etc.).
        - shows (id, movie_id, hall_id, showtime, etc.).
        - bookings (id, customer_id, show_id, seat_numbers, payment_status, etc.).
    - Export the schema and connect it to Spring Boot via application.properties.
3. **Backend Development**
    - Create entities (e.g., User, Movie, Theatre) with JPA annotations.
    - Build repositories using Spring Data JPA.
    - Develop REST controllers for admin, customer, and guest functionalities.
    - Implement authentication using Spring Security.
4. **Frontend Development**
    - Design Thymeleaf templates for:
        - Login/registration pages.
        - Movie listing and search pages.

        - Seat selection interface.
        - Payment confirmation page.
    - Integrate with Spring MVC controllers.
5. **API Testing**
    - Use Postman to test endpoints (e.g., /api/movies, /api/bookings).
    - Validate responses for all CRUD operations and authentication.
6. **Integration and Deployment**
    - Test the full application locally.
    - Deploy to a local server (e.g., Tomcat embedded in Spring Boot).
    - (Optional) Deploy to a cloud platform like Heroku or AWS.

6.2 Milestones

- **Week 1**: Project setup and database design.
- **Week 2**:Frontend development (Thymeleaf templates).
- **Week 3**: Backend development (APIs and logic).
- **Week 4**: Integration and testing with Postman.
- **Week 4**: Bug fixing, final testing, and deployment preparation.

---

7. Assumptions and Constraints

7.1 Assumptions

- Users have basic internet access and familiarity with web applications.
- Payment gateways (e.g., for UPI, cards) are simulated for this project.

7.2 Constraints

- Limited to a single currency for transactions.
- Initial deployment is local; cloud hosting requires additional configuration.

---

8. Deliverables

- Source code hosted on a Git repository.
- MySQL database schema and sample data.
- Postman collection for API testing.
- Deployed application (local server).
- Project documentation (this PRD and a user guide).