MOVIE TICKET BOOKING SYSTEM

A PROJECT REPORT

Submitted by

SUREKA V	**2303811710422162**
TEJES SHREE J	**2303811710422167**
VARUNIKA S	**2303811710422175**
YAZHINI P	**2303811710422185**

in partial fulfillment of the requirements for the award degree of

Bachelor in Engineering

CSB1303 – OBJECT ORIENTED ANALYSIS AND DESIGN

in

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

K. RAMAKRISHNAN COLLEGE OF TECHNOLOGY (AUTONOMOUS)
SAMAYAPURAM - 621112

DECEMBER - 2025

K. RAMAKRISHNAN COLLEGE OF TECHNOLOGY

(AUTONOMOUS)

SAMAYAPURAM - 621112

BONAFIDE CERTIFICATE

The work embodied in the present project report entitled " **MOVIE TICKET BOOKING SYSTEM**" has been carried out by the students **SUREKA V, TEJES SHREE J, VARUNIKA S, YAZHINI P**. The work reported here in is original and we declare that the project is their own work, except where specifically acknowledged, and has not been copied from other sources or been previously submitted for assessment.

Date of Viva Voce: ……

Mrs.V. KALPANA M.E., (Ph.D.,)

SUPERVISOR

Assistant Professor

Department of CSE

K. Ramakrishnan College of Technology (Autonomous)

Samayapuram – 621 112.

Mr. R. RAJAVARMAN M.E., (Ph.D.,)

HEAD OF THE DEPARTMENT

Assistant Professor (Sr. Grade)

Department of CSE

K. Ramakrishnan College of Technology (Autonomous)

Samayapuram – 621 112.

INTERNAL EXAMINER

EXTERNAL EXAMNIER

ABSTRACT

The Movie Ticket Booking System is a modern web-based application designed to make discovering movies, selecting showtimes, and booking tickets simple and user-friendly. It allows customers to browse currently running and upcoming movies, view detailed information such as cast, genre, duration, language, and trailers. Users can choose their preferred movie, showtime, and seats through an interactive layout showing real-time availability. The system offers options like different screen types, show categories, and seating preferences to enhance the booking experience. A secure online payment gateway supports multiple payment modes, and upon successful payment, users receive an instant e-ticket with complete booking details.

KEYWORDS

Movie Ticket Booking System, web-based application, online ticket reservation, seat booking and automated seat tracking, showtime management, secure online payments, QR code payment, admin dashboard and booking management, payment success and retry modules, customer reviews and feedback, and user-friendly interface.

ACKNOWLEDGEMENT

We thank our **DR. N. VASUDEVAN**, Principal, for his valuable suggestions and support during the course of my research work.

We thank our **MR. R. RAJAVARMAN**, Head of the Department, Assistant Professor (Sr. Grade), Department of Computer Science and Engineering, for his valuable suggestions and support during the course of my research work.

We wish to record my deep sense of gratitude and profound thanks to my Guide **Mrs. V. KALPANA,** Assistant Professor , Department of Computer Science and Engineering**,** for her keen interest, inspiring guidance, constant encouragement with my work during all stages, to bring this thesis into fruition.

We are extremely indebted to our project coordinator **Mrs. V. KALPANA,** Assistant Professor, Department of Computer Science and Engineering, for her valuable suggestions and support during the course of my research work.

We also thank the faculty and non-teaching staff members of the Department of Computer Science And Engineering, K.Ramakrishnan College Of Technology, Samayapuram, for their valuable support throughout the course of my research work.

Finally, we thank our parents, friends and our well wishes for their kind support.

SIGNATURE

_____

_____

_____

_____

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ABBREVIATIONS

API	-	Application Programming Interface
DB	-	Database
UPI	-	Unified Payment Interface
GUI	-	Graphical User Interface
SQL	-	Structured Query Language
UX	-	User Experience

CHAPTER 1

INTRODUCTION

1.1 INTRODUCTION ABOUT DOMAIN

The domain of this project lies in the entertainment and cinema management sector, focusing specifically on online movie ticket reservation systems. With the rapid advancement of digital technology, theatres and multiplexes are increasingly adopting web-based platforms to enhance customer convenience and streamline their operations. Online ticket booking systems have become essential tools that allow users to view movies, check show timings, select seats, and book tickets instantly, reducing the need for physical queues and manual booking processes. This domain integrates concepts of web development, database management, user interface design, and automated service workflows. It aims to improve customer experience by offering quick, easy, and personalized booking options while supporting theatre staff with real-time seat availability, automated booking updates, and efficient schedule management.

1.2 PROBLEM DESCRIPTION

The Movie Ticket Booking System is a web application developed using Python Flask and MySQL that allows users to conveniently book movie tickets online without visiting the theatre in person. Customers can browse currently running movies, view showtimes, check theatre locations, watch trailers, and select their preferred date, time, and seat category (Normal/ Premium/ VIP). The system allows users to register, log in, and select seats through an interactive seating layout that displays real-time seat availability. After seat selection, customers can make secure simulated online payments and instantly download or view their digital tickets. Additionally, users have the option to cancel bookings within the allowed timeframe and give feedback or ratings based on their movie experience. This platform ensures a smooth, user-friendly, and automated movie ticket reservation process for all customers.

1.3 OBJECTIVE OF THE PROJECT

The main objective of the Movie Ticket Booking System is to create an efficient, convenient, and user-friendly platform that allows users to book movie tickets online without waiting in long queues or visiting the theatre physically. The project aims to simplify the ticket reservation process by enabling customers to browse movies, view showtimes, select theatres, choose preferred seats through an interactive layout, and complete payments securely. It also seeks to enhance the reliability of bookings through features such as digital tickets, booking confirmation alerts, and easy cancellation options. Additionally, the system provides users with access to their booking history and the ability to rate movies or share feedback after watching. For theatre administrators, the project aims to offer a centralized dashboard to manage movies, update show schedules, monitor seat availability, and oversee user bookings.

1.4 SCOPE OF THE PROJECT

The scope of the Movie Ticket Booking System covers both user-focused and administrative functionalities. It allows customers to register, log in, and conveniently book movie tickets online by selecting their preferred movie, theatre, showtime, seat category, and specific seat(s) through an interactive seating layout. The system also includes additional features such as secure online payment simulation, booking cancellation, viewing booking history, and submitting feedback after watching a movie. Users can also view trailers, movie details, and seat availability before confirming their booking. On the administrative side, the project provides a dedicated dashboard where theatre staff or admins can manage movies, update show schedules, monitor bookings, manage seat availability, and review customer feedback for operational improvement. Administrators can also add or remove movies, update pricing, and check overall booking reports.

CHAPTER 2

SYSTEM REQUIREMENT SPECIFICATION (SRS)

2.1 FUNCTIONAL REQUIREMENTS

2.1.1 User Management

The system allows users to register using their basic details and log in securely with their credentials. Two roles are supported: Customer and Admin. Customers can browse movies and book tickets, while Admins manage all system operations. A secure logout option is provided to safely end the user session.

2.1.2 Movie & Show Selection

Customers can explore currently available movies, view detailed information, watch trailers, and select theatres, showtimes, and preferred formats (2D/3D/IMAX). The system displays available seats through an interactive seating layout and prevents double booking of seats.

2.1.3 Ticket Booking

Customers can select specific seats from the seat map and proceed to book tickets by confirming the showtime, seat category (Normal/Premium/VIP), and ticket quantity. The system verifies seat availability in real time before booking.

2.1.4 Payment Processing

Customers complete their ticket booking through a secure simulated online payment gateway. The system verifies the payment details and processes the transaction instantly. Once the payment succeeds, the booking status becomes Confirmed, and a digital ticket is automatically generated. This ensures smooth, fast, and reliable payment handling for all users.

2.1.5 Booking History

Users can view all their past and upcoming reservations. Each booking entry shows details like date, time, table number, occasion, and payment status, helping users track their reservation history.

2.1.6 Admin Module

The Admin Module handles the authentication process for administrators. Admins can log in using their username and password, and the system verifies their credentials securely. Only after successful login can the admin access the internal management features. This module ensures that only authorized administrators can enter the admin area.

2.1.7 Admin Dashboard Module

The Admin Dashboard Module provides the control panel where the admin manages all theatre operations. From this dashboard, admins can add or update movies, set show timings, check seat availability, view or manage bookings, handle cancellations, and monitor customer feedback. It also includes a complaint management section where admins can view complaints, update their status, edit entries, or delete resolved tickets. This module helps the admin maintain smooth system operations.

2.1.8 Security

The system ensures strong security by storing all user passwords using hashing techniques and protecting sessions to prevent unauthorized access. Only authenticated users are allowed to view or manage bookings or access admin features, ensuring controlled system usage. Additionally, sensitive operations such as payments and admin actions include extra validation measures to maintain data integrity and safeguard user information.

2.2 NON-FUNCTIONAL REQUIREMENTS

2.2.1 Performance Requirements

The system should load pages quickly and respond to user actions within a few seconds. Movie-related operations such as loading showtimes, displaying seat availability, or processing payments must be executed efficiently to avoid delays. The system should support multiple users booking tickets simultaneously without performance degradation.

2.2.2 Reliability Requirements

The system must operate consistently without crashes and ensure that booking information, seat selections, and payment details are safely stored. It should handle unexpected issues such as server downtime or failed transactions gracefully by showing meaningful error messages and preserving user data without loss.

2.2.3 Usability Requirements

The interface should be simple, visually appealing, and easy for all users to navigate, even for those with limited technical knowledge. Options such as movie selection, seat layout, and payment forms must be clear and intuitive.

2.2.4 Security Requirements

User information, including login credentials and payment data, must be encrypted and protected against unauthorized access. Only authenticated customers and admins should access their respective dashboards. Secure session management must be implemented.

2.2.5 Scalability Requirements

The system must be built to support future expansion, such as adding more theatres, new screens, additional showtimes, or advanced features like loyalty points. It should handle increasing user traffic and a larger number of bookings without compromising performance.

2.2.6 Availability Requirements

The ticket booking system should remain accessible at all times, especially during peak movie release periods and weekend showtimes. Any scheduled maintenance should be minimal and communicated in advance so that users are not inconvenienced.

2.2.7 Maintainability Requirements

The system should be designed with clean, modular, and well-structured code to allow developers to make updates easily, fix bugs, or extend functionality in the future. Proper documentation must be maintained to support long-term system maintenance and upgrades.

2.3 HARDWARE REQUIREMENTS

2.3.1 Server-Side Hardware Requirements

The Movie Ticket Booking System requires a server machine with at least 8 GB RAM, a quad-core processor, and 50 GB of available disk space to efficiently run the backend services, database, and web server. A reliable network interface card is essential to handle multiple simultaneous user requests, especially during peak showtime bookings. A server machine with at least 8 GB RAM, a quad-core processor, and 50 GB of available disk space is recommended to ensure smooth execution of backend services, real-time seat availability updates, and secure transaction handling.

2.3.2 Client-Side Hardware Requirements

End users (both customers and admins) can access the system from any device with basic hardware specifications such as 2 GB RAM, a dual-core processor, and at least 1 GB of free storage. Since the system runs through standard web browsers, it is compatible with laptops, desktops, tablets, and smartphones as long as they have a stable internet connection. For administrators managing showtimes, bookings, and feedback, a slightly enhanced configuration such as 4 GB RAM and a larger display screen can improve usability, especially when handling multiple dashboard operations.

2.3.3 Network Requirements

A consistent internet connection with a minimum of 5 Mbps bandwidth is required for customers to browse movies, view seat layouts, and book tickets smoothly. The server environment requires a high-speed, reliable network to ensure fast response times, uninterrupted ticket processing, and smooth handling of large traffic during movie releases and peak booking hours. On the server side, a high-speed, dedicated network environment is required to handle large volumes of concurrent user requests, especially during popular movie releases, festival weekends, and advance booking windows. The server should support high throughput and low latency to facilitate quick data retrieval from the database, secure payment gateway communication, and instant generation of booking confirmations

2.4 SOFTWARE REQUIREMENTS

The Movie Ticket Booking System needs a web server like Apache or Flask to run the website. It uses a database such as MySQL to store movies, users, and booking information. The frontend is created using HTML, CSS, and JavaScript for easy interaction. The backend requires PHP or Python to process bookings and connect with the database. A modern web browser like Chrome or Firefox is needed to access the system.

2.5 USER CHARACTERISTICS

Users of the Movie Ticket Booking System are expected to have basic computer or mobile device knowledge. They should know how to operate a web browser and navigate simple online interfaces. Customer users do not require any technical expertise; they should be able to browse movies, select showtimes, and book tickets easily without training. Admin users, however, need slightly higher technical skills to manage movies, show schedules, seat availability, and bookings. All users should be comfortable using to interact with the system effectively.

2.6 CONSTRAINTS

The system has a few limitations. It requires a stable internet connection to load movies, showtimes, and real-time seat availability accurately. It must follow privacy and security rules to protect user data and payment information, and it can only run smoothly on supported and updated web browsers. The system also depends on the server's capacity; if too many users try to book tickets at the same time especially during new movie releases it may experience slowdowns. All bookings, cancellations, and payments must follow the theatre's policies and rules. These constraints ensure that the system remains secure, accurate, and reliable for all users.

CHAPTER 3

ANALYSIS AND DESIGN

3.1 USE CASE DIAGRAM

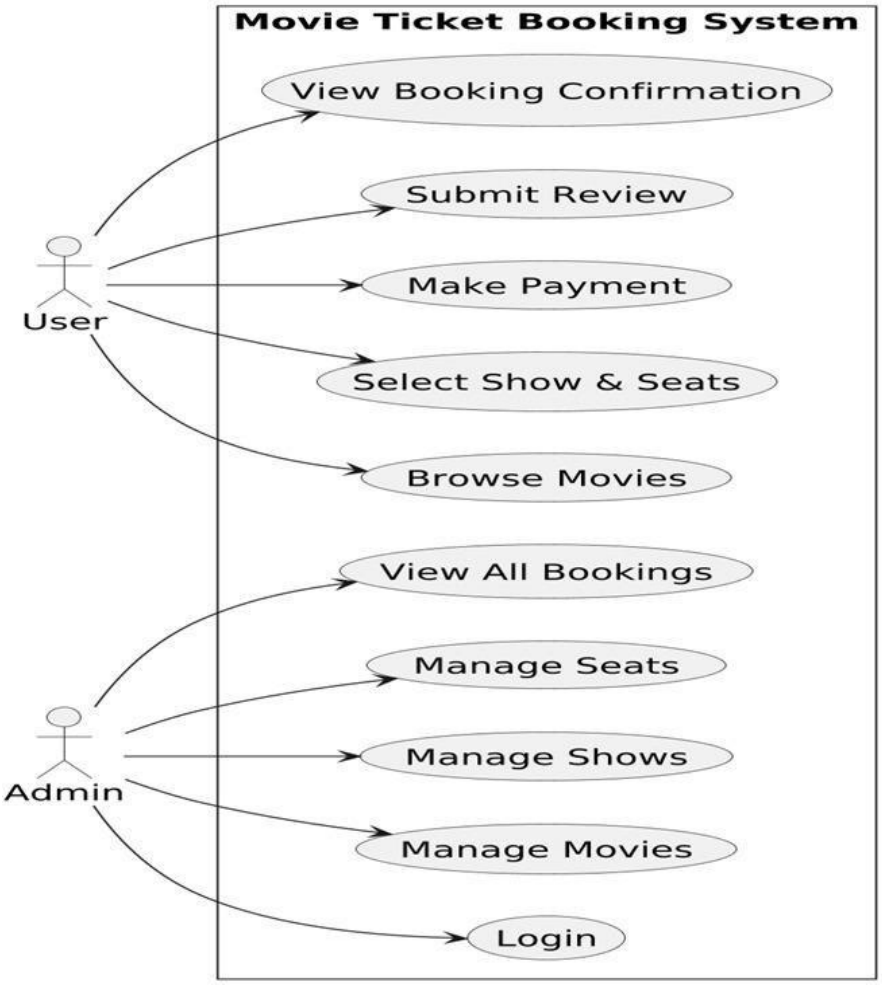

Figure 3.1 Use Case Diagram

3.1.1 USE CASE DIAGRAM DESCRIPTION

This diagram shows the Movie Ticket Booking System with two main actors: the User and the Admin. The user can browse movies, select shows and seats, make payments, view booking confirmations, and submit reviews. The admin manages movies, shows, and seat arrangements, and can also view all bookings. Each oval represents a use case, meaning an action performed in the system. Overall, the diagram explains how both user and admin interact with the system to complete their tasks.

3.2 CLASS DIAGRAM

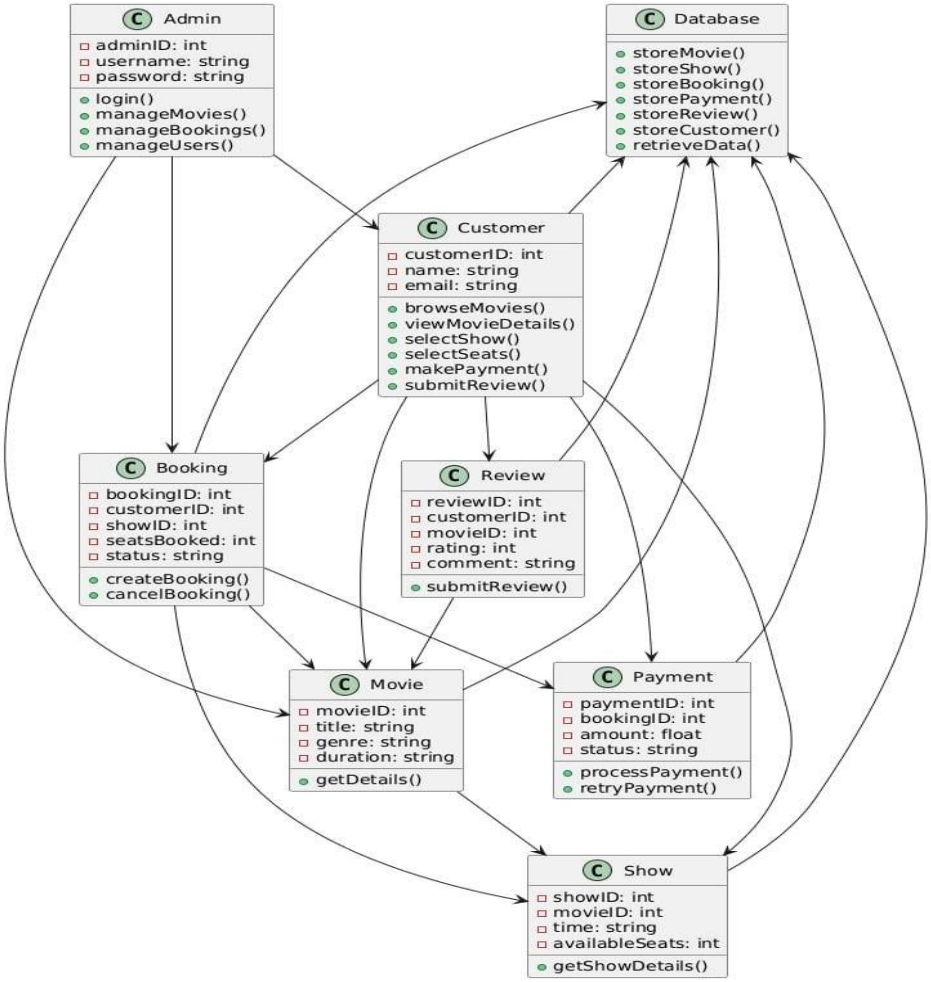

Figure 3.2 Class Diagram

3.2.1 CLASS DIAGRAM DESCRIPTION

The class diagram shows the main parts of a Movie Ticket Booking System and how they are connected. It includes classes like User, Admin, Booking, Show, Movie, Seat, and Payment, each with their own attributes and functions. The User can book tickets and make payments, while the Admin manages movies, shows, and seats. Bookings are linked to users, shows, payments, and selected seats. different classes work together to complete the movie ticket booking process.

3.3 ACTIVITY DIAGRAM

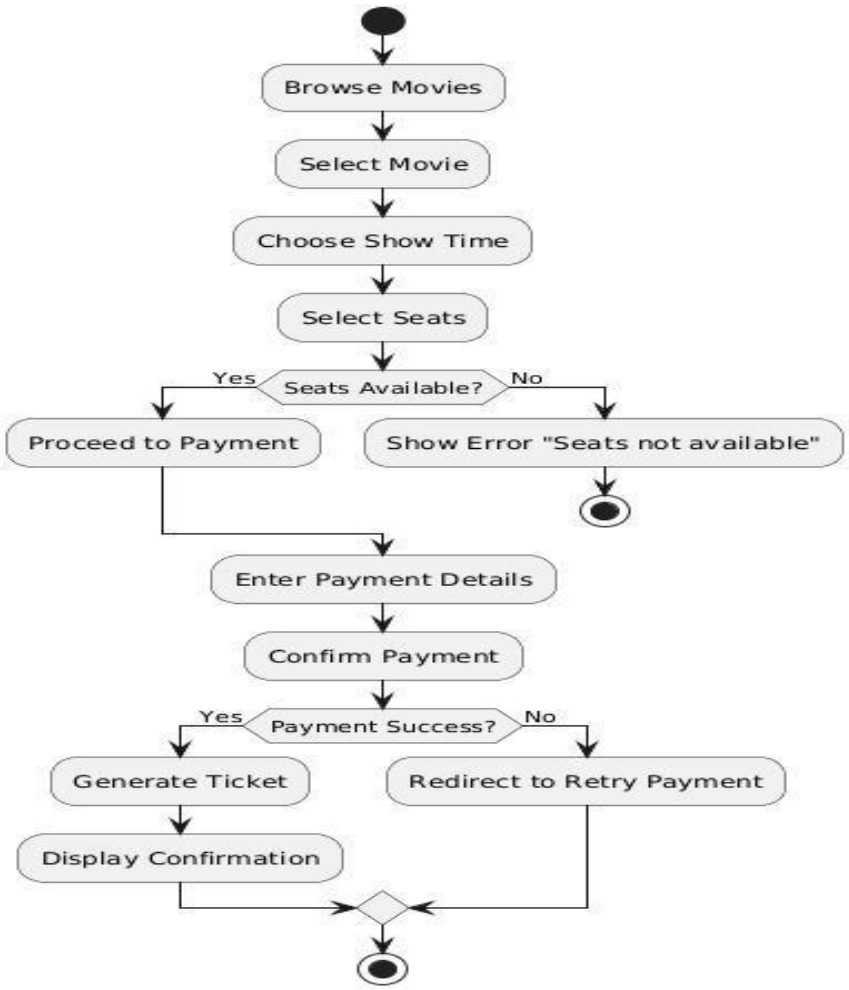

Figure 3.3 Activity Diagram

3.3.1 ACTIVITY DIAGRAM DESCRIPTION

The activity diagram shows the step-by-step process of booking a movie ticket. It starts with browsing movies, selecting a movie, choosing a show time, and picking the seats. If seats are available, the user proceeds to payment; otherwise, an error is shown. After entering and confirming the payment details, the system checks if the payment is successful. If successful, a ticket is generated and the confirmation is displayed; if not, the user is redirected to retry the payment.

3.4 SEQUENCE DIAGRAM

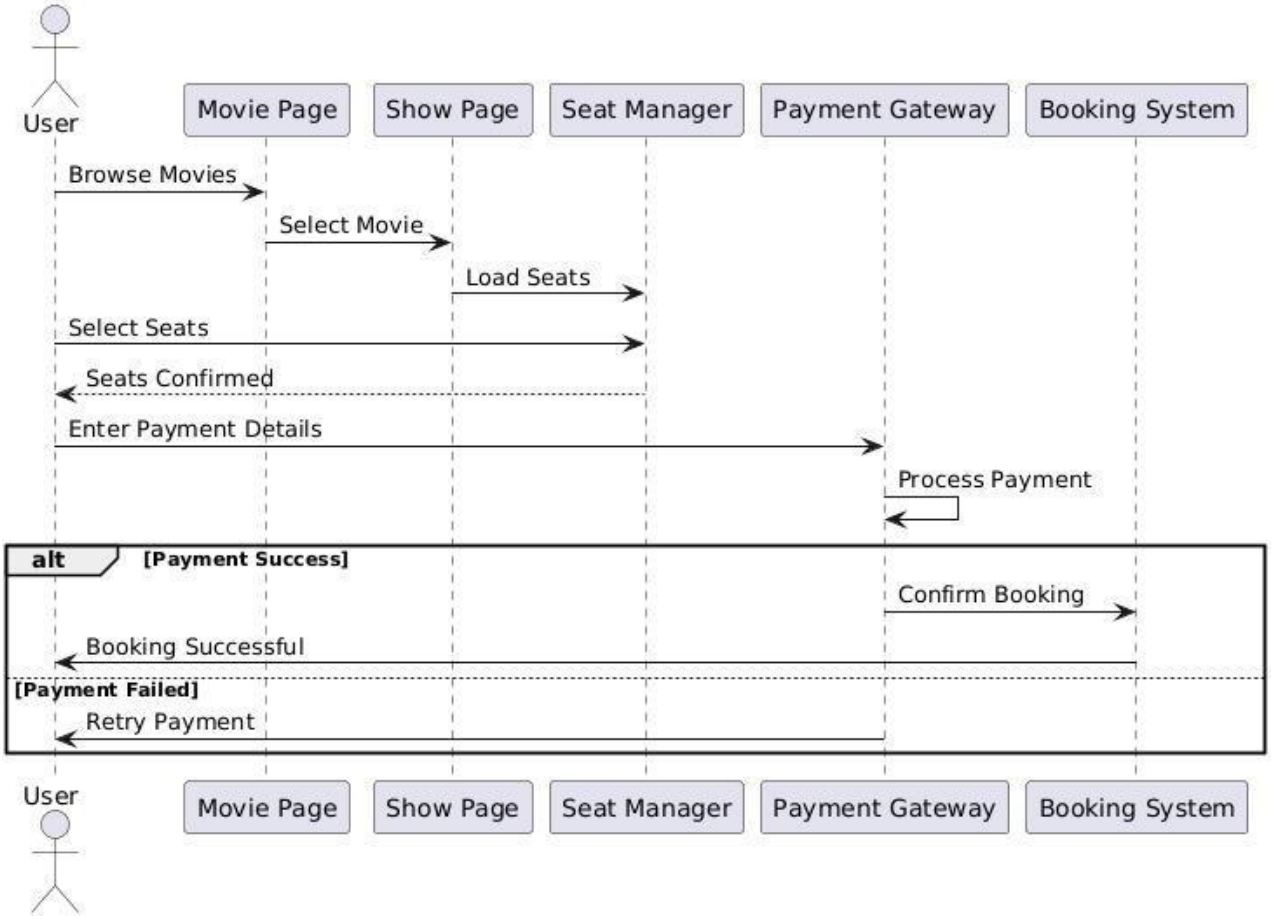

Figure 3.4 Sequence Diagram

3.4.1 SEQUENCE DIAGRAM DESCRIPTION

The sequence diagram shows how a user interacts with different parts of the system to book a movie ticket. It begins with the user browsing movies, selecting a movie, and choosing seats, while the Show Page and Seat Manager load and confirm the seat details. The user then enters payment information, which is processed by the Payment Gateway. If the payment succeeds, the Booking System confirms the booking and sends a success message to the user. If the payment fails, the system prompts the user to retry the payment, showing the alternate flow.

3.5 STATE MACHINE DIAGRAM

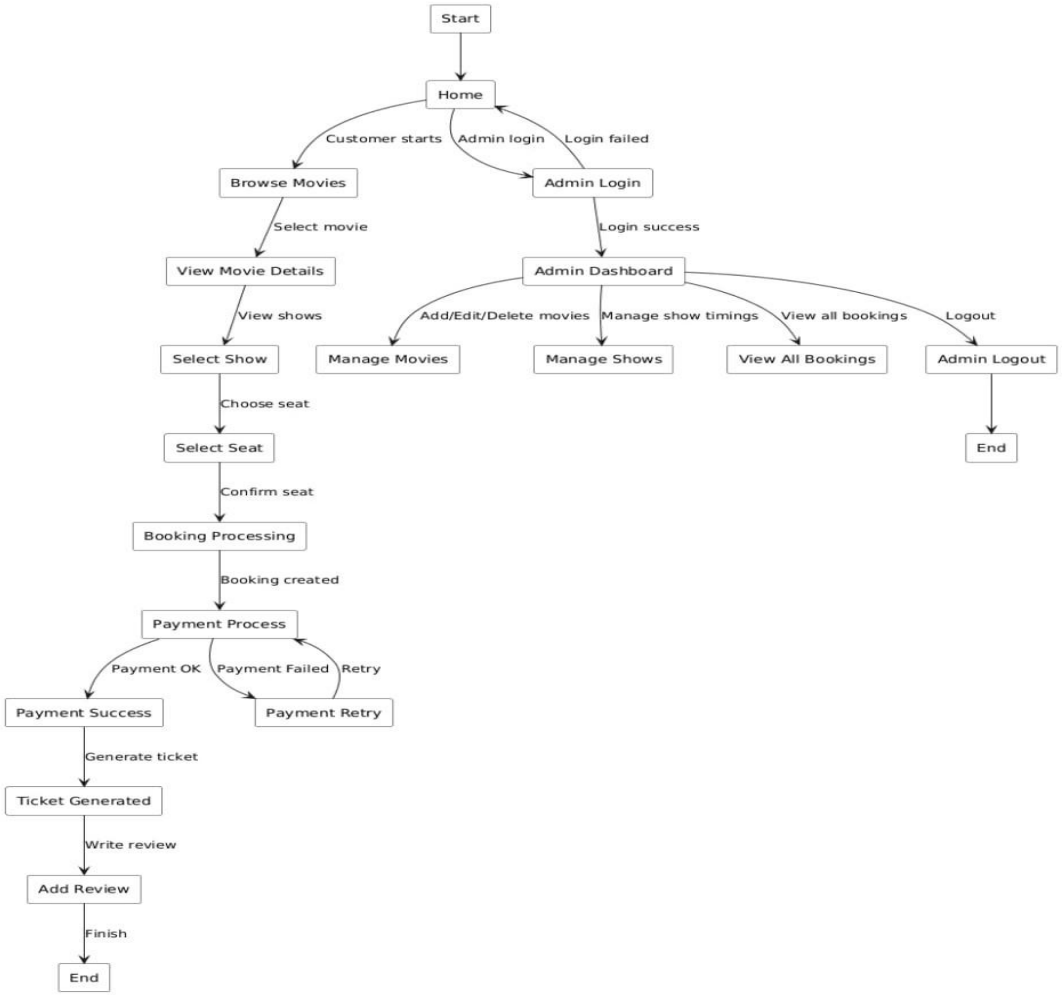

Figure 3.5 State Machine Diagram

3.5.1 STATE MACHINE DIAGRAM DESCRIPTION

The state machine diagram shows how customers and admins move through different steps in the Movie Ticket Booking System. It begins at the home state, where the customer starts browsing movies while the admin attempts to log in. The customer then views movie details, selects a show and seat, and begins the booking and payment process. After payment success, the system generates the ticket and allows the customer to add a review before ending. For admins, a successful login leads to the dashboard, where they can manage movies, shows, and bookings. Finally, the admin logs out, completing their flow within the system.

3.6 COMPONENT DIAGRAM

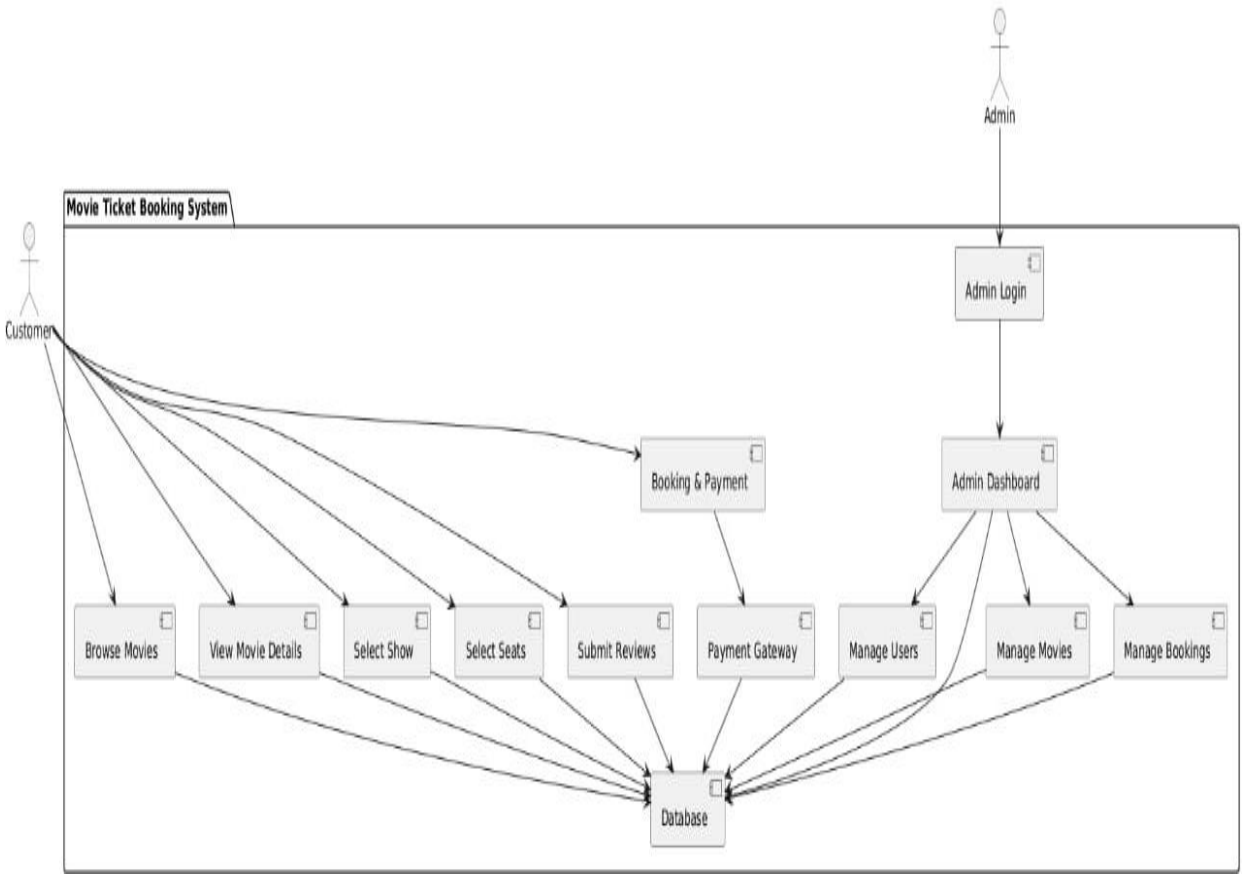

Figure 3.6 Component Diagram

3.6.1 COMPONENT DIAGRAM DESCRIPTION

The component diagram shows how the Movie Ticket Booking System is divided into Customer, Admin, and Shared Service modules. The Customer Module handles browsing movies, viewing details, selecting shows and seats, booking tickets, and submitting reviews. The Admin Module allows administrators to log in, access the dashboard, and manage movies, users, and bookings. Shared Services such as the Payment Gateway and Database support both customer and admin operations. Overall, the diagram highlights how different components interact to deliver smooth booking and management functions.

3.7 DEPLOYMENT DIAGRAM

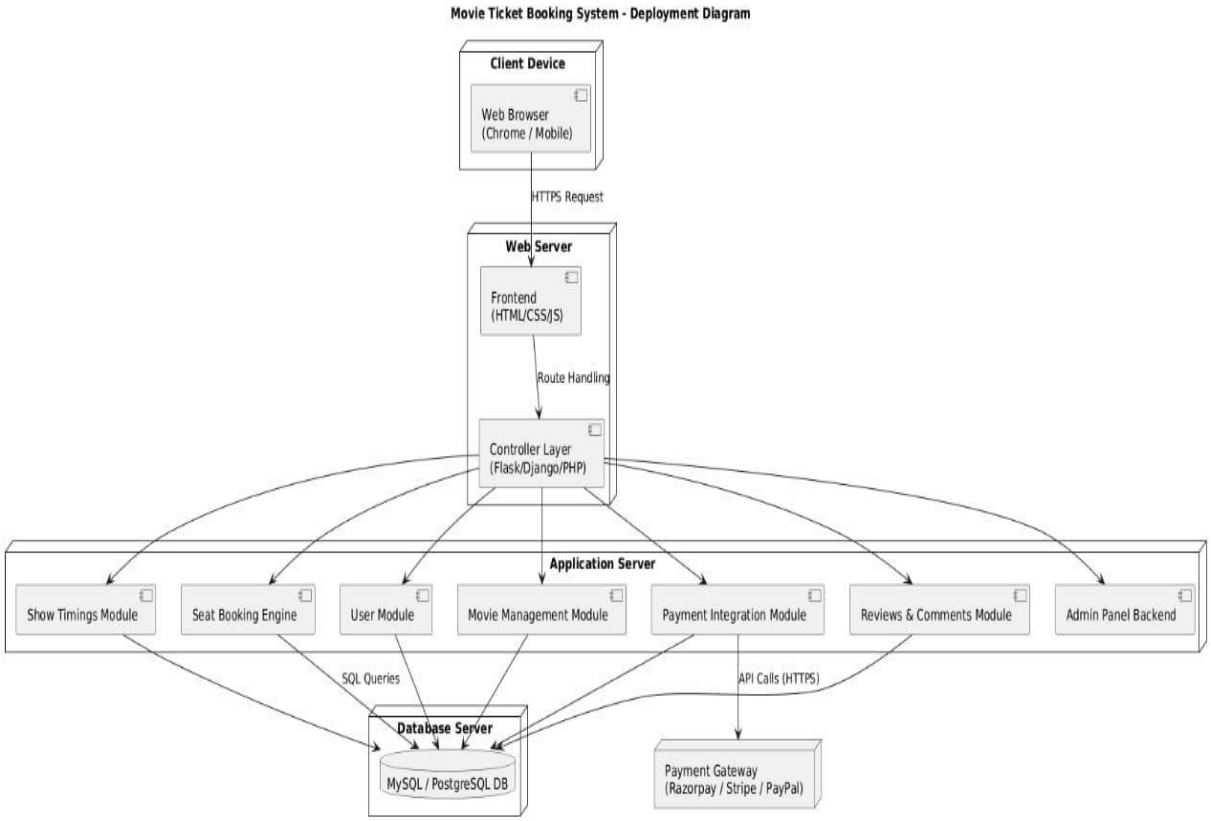

Figure 3.7 Deployment Diagram

3.7.1 DEPLOYMENT DIAGRAM DESCRIPTION

The deployment diagram shows how the Movie Ticket Booking System is installed and runs across different devices and servers. The user accesses the system through a web browser on a client device, which sends requests to the web server. The web server handles the frontend and routes requests to the application server, where modules like seat booking, movie management, payments, and reviews operate. These modules access the database server to store or retrieve information, while payment requests are sent to an external payment gateway. Overall, the diagram explains how all hardware and software components work together to deliver the complete booking system.

3.8 PACKAGE DIAGRAM

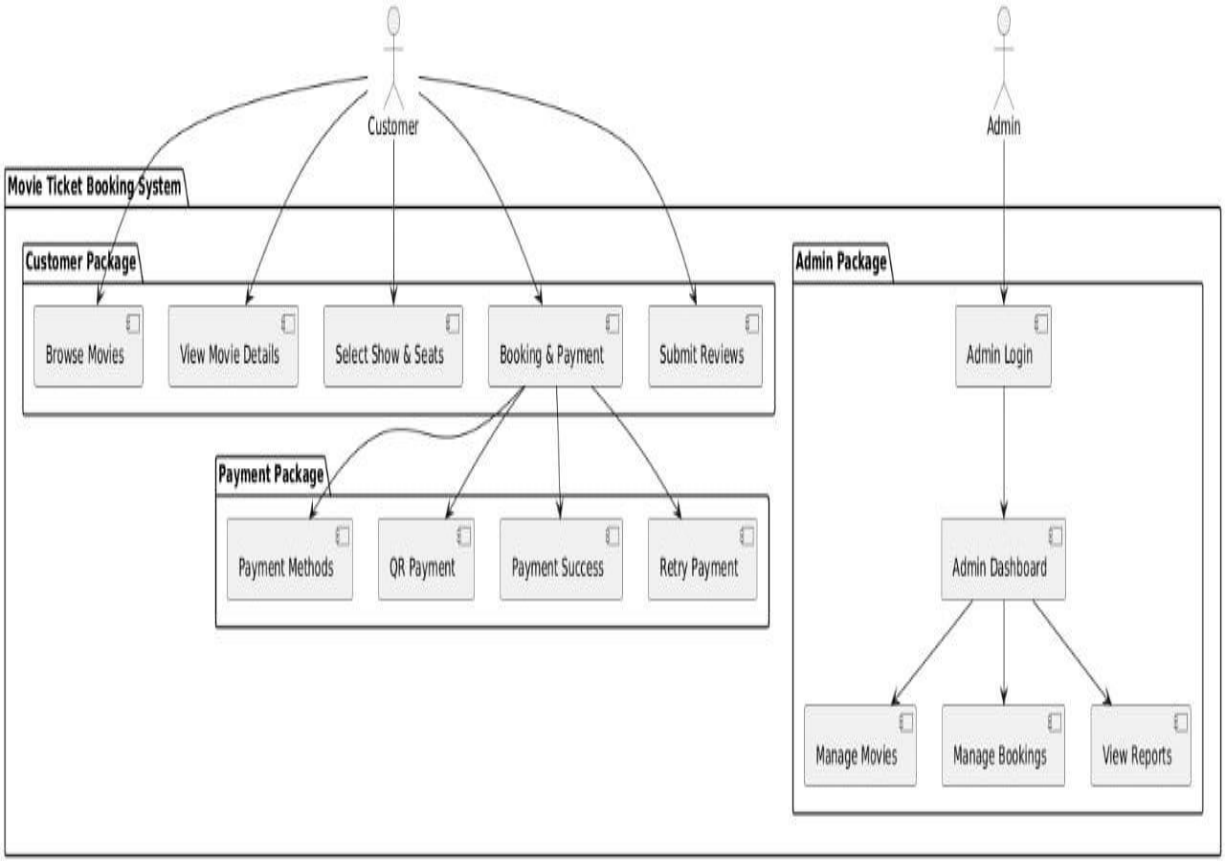

Figure 3.8 Package Diagram

3.8.1 PACKAGE DIAGRAM DESCRIPTION

The package diagram represents the structure of a Movie Ticket Booking System divided into Customer, Payment, and Admin packages. The Customer package handles browsing movies, viewing details, selecting shows and seats, booking tickets, and submitting reviews. The Payment package supports different payment methods, QR payments, payment success, and retry options. The Admin package includes login access, a dashboard, and tools to manage movies, bookings, and reports. Overall, the diagram shows how different functional modules interact to provide a smooth movie booking experience.

3.9 COLLABORATION DIAGRAM

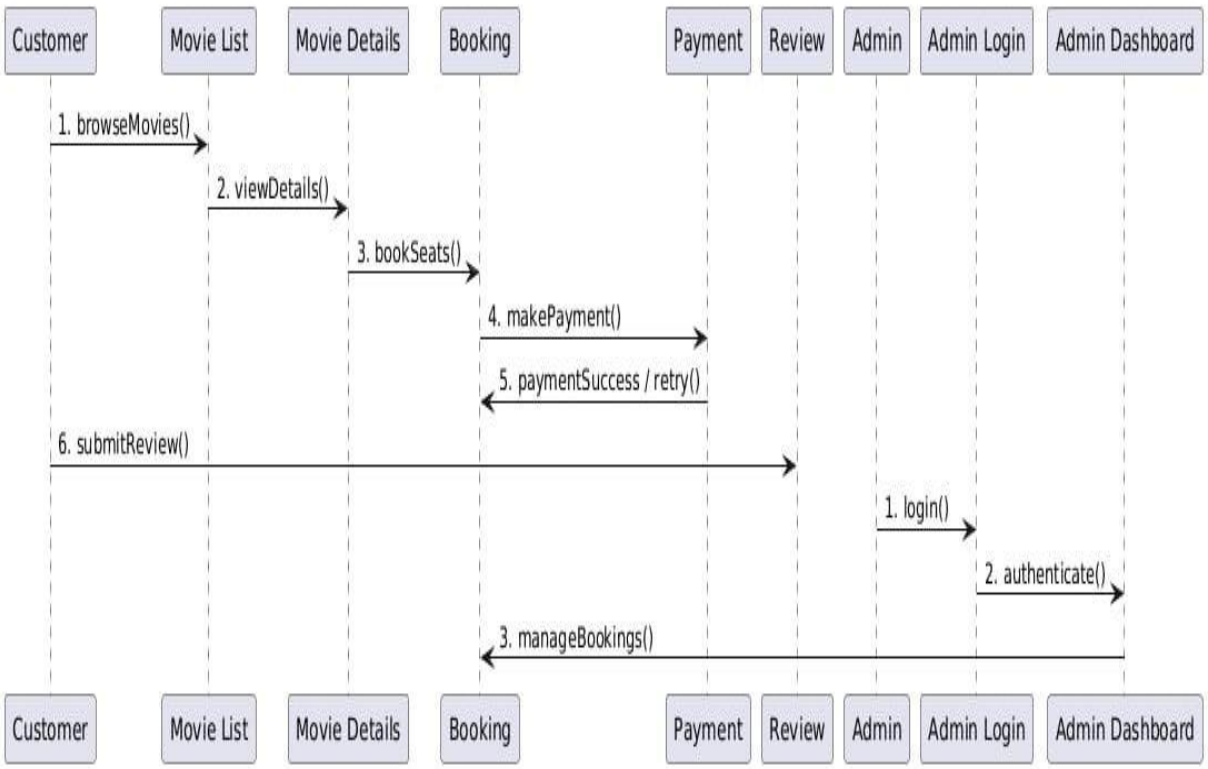

Figure 3.9 Collaboration Diagram

3.9.1 COLLABORATION DIAGRAM DESCRIPTION

The collaboration diagram illustrates how different objects in the Movie Ticket Booking System communicate with each other to complete ticket booking and management tasks. It shows customer interactions such as browsing movies, viewing details, booking seats, making payments, and submitting reviews without requiring login. The diagram also represents administrator interactions, where the admin logs in, accesses the dashboard, and manages movies and bookings. Numbered messages indicate the order of communication between objects, emphasizing object relationships and coordination within the system.

3.9 DESIGN PATTERNS USED

The Movie Ticket Booking System incorporates a combination of GRASP (General Responsibility Assignment Software Patterns) and GoF (Gang of Four) design patterns to ensure a modular, maintainable, and scalable architecture. GRASP principles such as Controller, Creator, and Information Expert are applied to assign responsibilities efficiently—where dedicated controllers manage booking requests, payment processing, and seat allocation, while expert classes handle movie data, schedules, and user information. The Low Coupling and High Cohesion patterns help in organizing system components so each module performs a specific task without unnecessary dependencies, enhancing flexibility for future upgrades. In addition, GoF design patterns further strengthen the system design. The Singleton pattern is used for managing shared resources like the database connection and configuration settings, ensuring consistency throughout the application. The Factory Method pattern helps create objects such as user sessions, payment handlers, and ticket instances without tightly coupling them to specific classes. The Observer pattern is applied in scenarios like real-time seat updates, where changes in seat availability must be instantly reflected across user interfaces. The MVC (Model–View–Controller) architectural pattern, commonly implemented in web applications, separates data processing, business logic, and user interface modules, making the system easier to test, develop, and maintain.

CHAPTER 4

IMPLEMENTATION

4.1 MODULE DESCRIPTION

4.1.1 User Module

The User Module is responsible for managing all customer-related interactions within the Movie Ticket Booking System. It allows users to browse available movies, view show timings, read descriptions, watch trailers, and check ticket prices. Users can select seats, proceed with booking, make payments, and receive digital tickets instantly. Additionally, customers can submit reviews and star ratings for movies they have watched.

4.1.2 Booking & Seating Module

The Booking & Seating Module manages the complete seat selection and ticket booking workflow. Users can select their preferred seats from an interactive seat layout, which updates in real time to reflect availability. The module prevents double booking by immediately marking seats as booked once the reservation is confirmed.

4.1.3 Payment & Review Module

The Payment & Review Module combines secure payment processing with feedback collection. It provides multiple payment options including card payment, QR payment, and payment retry for failed transactions. Once the user completes the payment successfully, the module updates the booking status to “Paid” and generates a digital ticket or confirmation page.

4.1.4 Admin Module

The Admin Module provides complete management control for system administrators. Through a secure login page, admins can access the admin dashboard to manage movies, monitor bookings, update pricing, and view reviews left by customers.

Admins can also add new movies, edit existing movie details, upload posters, include trailers, or remove outdated films.

4.1.5 Admin Dashboard Module

The Movie Management Module handles all functionalities related to storing and updating movie information. Admins can add movies with details such as title, show time, price, description, image URL, trailer link, and total seats The module also allows modification of movie details and deletion of movies no longer in theatres.

4.2 TECHNOLOGY DESCRIPTION

The Movie Ticket Booking System is developed using modern web technologies to ensure smooth performance, secure transactions, and an easy-to-use interface. The backend is built using Python Flask, which handles server-side processing, routing, and communication between the database and the user interface. MySQL is used as the database to store all essential information such as movies, showtimes, seat availability, user details, and booking records. The frontend is designed using HTML, CSS, and JavaScript to create a responsive and interactive user experience on both desktop and mobile devices. CSS ensures a clean and visually appealing layout, while JavaScript enhances interactivity, such as updating seat selection in real time. The system runs on a web server capable of handling multiple users simultaneously, and secure hashing techniques are used to protect passwords and user data. Together, these technologies provide a reliable, efficient, and user-friendly online ticket booking platform.

CHAPTER 5

TESTING

5.1 TESTING STRATEGY

The Movie Ticket Booking System follows a complete testing strategy to ensure that the application is accurate, reliable, secure, and performs well under all conditions. The testing process begins with Unit Testing, where each small module such as user login, movie listing, seat selection, and payment processing is tested individually to confirm that it works correctly. After that, Integration Testing is performed to check whether different modules interact smoothly, such as the connection between the booking system, database, and payment gateway. System Testing is then conducted to verify the entire application as a whole and ensure that all features meet the system requirements. User Acceptance Testing (UAT) involves real users testing the system to confirm that it is user-friendly, stable, and ready for deployment. Performance Testing evaluates how the system performs under heavy usage, including high-traffic booking situations, measuring speed, responsiveness, and stability. Finally, Security Testing checks for vulnerabilities like SQL injection, cross-site scripting, and unauthorized access, ensuring that all user data, payment details, and login credentials remain fully protected.

5.2 SAMPLE TEST CASES

Test Case 1: User Login

Objective: To verify that a registered user can successfully log in.

Input: Valid email (user@gmail.com) and correct password.

Expected Output: User is redirected to the home/dashboard page.

Status: Pass / Fail

Test Case 2: Movie Search

Objective: To confirm that the system displays correct movie results when a user searches.

Input: Search keyword “Leo”.

Expected Output: The system displays all movies matching the keyword “Leo”.

Status: Pass / Fail

Test Case 3: Seat Selection

Objective: To check if the user can select only available seats.

Input: User clicks on an available seat (e.g., A5).

Expected Output: Seat A5 becomes highlighted/selected and added to the booking cart.

Status: Pass / Fail

Test Case 4: Payment Processing

Objective: To verify successful payment through the integrated payment gateway.

Input: Valid UPI/Card details with sufficient balance.

Expected Output: Payment is successful and an e-ticket is generated.

Status: Pass / Fail

Test Case 5: Ticket Cancellation

Objective: To ensure users can cancel a booked ticket within the allowed time.

Input: User selects an existing booking and clicks “Cancel Ticket.”

Expected Output: Ticket is canceled and refund policy details are displayed.

Status: Pass / Fail

Test Case 6: Feedback Submission

Objective: To verify that users can submit ratings and comments for a movie.

Input: Rating = 4 stars, Comment = “Good movie!”.

Expected Output: Feedback is submitted successfully and stored in the database.

Status: Pass / Fail

5.3 TEST RESULTS

The Movie Ticket Booking System was tested thoroughly across all major modules to ensure smooth and error-free functioning. In the User Module, features such as movie browsing, seat selection, booking, payment processing, ticket generation, and review submission worked correctly on all devices. The Admin Module also performed successfully, with admin login, movie management, booking monitoring, and price updates operating without issues. All administrative actions adding, editing, or deleting movies were reflected instantly on the user side. Similarly, the Movie Management Module allowed movie details, posters, trailer links, and showtimes to be updated accurately, with every change stored properly in the database and displayed immediately in the UI. The Booking and Seating Module also behaved as expected, allowing users to select only available seats, preventing double bookings, and confirming bookings with real-time seat updates.

The Payment and Review Module functioned reliably, with all payment methods including card and QR payments being processed correctly, while failed transactions redirected users properly to the retry option. Feedback submission also worked smoothly, with user ratings and comments stored accurately and shown correctly in the review section. Overall testing confirms that all modules of the Movie Ticket Booking System are stable, consistent, and functioning exactly as intended, ensuring a reliable experience for both users and administrators.

CHAPTER 6

CONCLUSION AND FUTURE ENHANCEMENT

6.1 CONCLUSION

The Movie Ticket Booking System has been thoroughly designed, implemented, and tested to ensure a smooth and user-friendly experience for both customers and administrators. All core modules including user management, movie management, seat booking, payment processing, and review handling performed successfully during testing, demonstrating reliability and consistency across different devices and usage conditions. The system effectively streamlines the entire ticket booking workflow by offering real-time seat updates, secure online payments, and instant digital ticket generation. Administrative functionalities also operate efficiently, allowing authorized staff to manage movies, monitor bookings, update prices, and review user feedback with ease. Overall, the project achieves its objective of providing a modern, efficient, and fully functional movie ticket reservation platform that enhances convenience for users while supporting theaters with organized management and improved operational control.

6.2 FUTURE ENHANCEMENT

The Movie Ticket Booking System can be further improved with advanced features to enhance both user convenience and administrative efficiency. Future upgrades may include incorporating AI-based movie recommendations tailored to user preferences and browsing history. Adding SMS/WhatsApp ticket delivery and reminder notifications can make the booking experience more seamless. Integration of loyalty programs, reward points, and discount coupons would help increase user engagement. Multi-language support can make the system accessible to a wider audience. Additionally, implementing real-time analytics for admins such as daily booking reports, peak-hour trends, and customer behaviour insights can improve strategic decision-making. A mobile application version may also be developed in the future to provide faster access and offline ticket storage.

APPENDIX A

(SOURCE CODE)

SQL QUERY

```
CREATE DATABASE IF NOT EXISTS movie_booking;

USE movie_booking;

CREATE TABLE IF NOT EXISTS admin (

    id INT AUTO_INCREMENT PRIMARY KEY,

    username VARCHAR(50) NOT NULL,

    password VARCHAR(50) NOT NULL

);

INSERT INTO admin (username, password)

SELECT ‘admin’, ‘admin123’

FROM DUAL

WHERE NOT EXISTS (

    SELECT 1 FROM admin WHERE username = ‘admin’

);

CREATE TABLE IF NOT EXISTS movies (

    id INT AUTO_INCREMENT PRIMARY KEY,

    title VARCHAR(100) NOT NULL,

    show_time VARCHAR(50) NOT NULL,

    price DECIMAL(10,2) NOT NULL,
```

```
    image_url VARCHAR(255),
    description TEXT,
    total_seats INT DEFAULT 100,
    trailer_url VARCHAR(255)
);
CREATE TABLE IF NOT EXISTS bookings (
    id INT AUTO_INCREMENT PRIMARY KEY,
    customer_name VARCHAR(100) NOT NULL,
    email VARCHAR(100) NOT NULL,
    seat_no VARCHAR(20) NOT NULL,
    movie_id INT NOT NULL,
    show_time VARCHAR(50) NOT NULL,
    price DECIMAL(10,2) NOT NULL,
    payment_status VARCHAR(20) DEFAULT 'Pending',
    FOREIGN KEY (movie_id)
        REFERENCES movies(id)
        ON DELETE CASCADE
);
CREATE TABLE IF NOT EXISTS reviews (
    id INT AUTO_INCREMENT PRIMARY KEY,
    movie_id INT NOT NULL,
    email VARCHAR(120),
```

```
    rating INT,
    comment TEXT,
    created_at DATETIME DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (movie_id) REFERENCES movies(id) ON DELETE CASCADE
);
INSERT INTO movies (title, show_time, price, image_url, description, trailer_url)
SELECT 'Nanban', '12:00 PM', 200.00, NULL,
'Three friends reunite and rediscover life's true meaning through friendship, dreams, and compassion.',
'https://www.youtube.com/embed/aIYNflpgHkI?si=bJMsZDg9-Q5H6jlr'
FROM DUAL
WHERE NOT EXISTS (SELECT 1 FROM movies WHERE title = 'Nanban');
INSERT INTO movies (title, show_time, price, image_url, description, trailer_url)
SELECT 'Dude', '2:30 PM', 220.00, NULL,
'Friendship, comedy, drama.',
'https://www.youtube.com/embed/PnL8C13b9Cc?si=uQ58ssTGGlv9yxoi'
FROM DUAL
WHERE NOT EXISTS (SELECT 1 FROM movies WHERE title = 'Dude');
INSERT INTO movies (title, show_time, price, image_url, description, trailer_url)
SELECT 'Sillunu oru kadhal', '3:00 PM', 250.00, NULL,
'Love, sacrifice, heartbreak, destiny.',
'https://www.youtube.com/embed/182B1Roehus?si=R6ZbgzpKRoPlvMig'
```

```
FROM DUAL
WHERE NOT EXISTS (SELECT 1 FROM movies WHERE title = 'Sillunu oru kadhal');
INSERT INTO movies (title, show_time, price, image_url, description, trailer_url)
SELECT 'VIP', '4:30 PM', 270.00, NULL,
'A bold engineer fights challenges in career and family while proving his worth against powerful enemies.',
'https://www.youtube.com/embed/fZOwwAzI9jM?si=R2KOsGlsJ6k4nhTV'
FROM DUAL
WHERE NOT EXISTS (SELECT 1 FROM movies WHERE title = 'VIP');
INSERT INTO movies (title, show_time, price, image_url, description, trailer_url)
SELECT 'Gilli', '5:00 PM', 300.00, NULL,
'A fearless kabaddi player rescues a girl from a deadly gangster and fights to protect her.',
'https://www.youtube.com/embed/EtJXEmW_XNM?si=1Lp42eO24hIFCTbY'
FROM DUAL
WHERE NOT EXISTS (SELECT 1 FROM movies WHERE title = 'Gilli');
INSERT INTO movies (title, show_time, price, image_url, description, trailer_url)
SELECT 'Ethirneechal', '6:30 PM', 280.00, NULL,
'An underdog transforms his life through determination and becomes an inspiring athlete.',
'https://www.youtube.com/embed/JhaTrO-Anpg?si=YG1fGhl9L9OXRJz2'
FROM DUAL
```

```
WHERE NOT EXISTS (SELECT 1 FROM movies WHERE title = ‘Ethirneechal’);

INSERT INTO movies (title, show_time, price, image_url, description, trailer_url)

SELECT ‘Remo’, ‘7:00 PM’, 260.00, NULL,

‘A lovestruck man disguises himself as a nurse to win the heart of the woman he loves.’,

‘https://www.youtube.com/embed/GEB4qrrWIgs?si=PlY-q9Pqej4Gaz4f’

FROM DUAL

WHERE NOT EXISTS (SELECT 1 FROM movies WHERE title = ‘Remo’);

INSERT INTO movies (title, show_time, price, image_url, description, trailer_url)

INSERT INTO movies (title, show_time, price, image_url, description, trailer_url)
```

PYTHON FLASK

```
import os

from flask import Flask, render_template, request, redirect, url_for, flash, session, send_from_directory

import pymysql

from werkzeug.utils import secure_filename

UPLOAD_FOLDER = 'static/images'

ALLOWED_EXTENSIONS = {'png', 'jpg', 'jpeg', 'gif'}

app = Flask(__name__)

app.secret_key = "moviesecret"

app.config['UPLOAD_FOLDER'] = UPLOAD_FOLDER

os.makedirs(app.config['UPLOAD_FOLDER'], exist_ok=True)

conn = pymysql.connect(
```

```
    host="localhost",
    user="root",
    password="Teja@1234!",
    database="movie_booking",
    cursorclass=pymysql.cursors.DictCursor,
    autocommit=False
)
cursor = conn.cursor()
def allowed_file(filename):
    return '.' in filename and filename.rsplit('.', 1)[1].lower() in
ALLOWED_EXTENSIONS
def _ids_placeholders(ids):
    return ",".join(["%s"] * len(ids)) if ids else ""
def youtube_embed(url):
    if 'watch?v=' in url:
        return url.replace('watch?v=', 'embed/')
    return url
def get_movie_ratings(movie_id):
    cursor.execute("""
        SELECT AVG(rating) AS avg_rating, COUNT(*) AS cnt
        FROM reviews
        WHERE movie_id=%s
```

```
    """, (movie_id,))
    row = cursor.fetchone()
    if not row:
        return 0.0, 0
    avg = float(row['avg_rating']) if row['avg_rating'] is not None else 0.0
    cnt = int(row['cnt']) if row['cnt'] is not None else 0
    return avg, cnt
@app.route('/')
def index():
    return render_template('index.html')
@app.route('/movies')
def movies():
    cursor.execute("SELECT * FROM movies ORDER BY id DESC")
    movies = cursor.fetchall()
 for m in movies:
        avg, cnt = get_movie_ratings(m['id'])
        m['avg_rating'] = float(round(avg, 1)) if cnt > 0 else 0.0
        m['rating_count'] = int(cnt)
 return render_template('movies.html', movies=movies)
@app.route('/book/<int:movie_id>', methods=['GET', 'POST'])
def book_movie(movie_id):
    except:
```

```
        conn.rollback()
        flash("Failed to submit feedback.")
    return redirect(url_for('review'))
@app.route('/comments/<int:movie_id>')
def comments(movie_id):
    # Fetch movie details
    cursor.execute("SELECT id, title, image_url, show_time, price FROM movies
WHERE id=%s", (movie_id,))
    movie = cursor.fetchone()
    if not movie:
        flash("Movie not found.")
        return redirect(url_for('movies'))
    cursor.execute("""
        SELECT email, rating, comment, created_at
        FROM reviews
        WHERE movie_id=%s
        ORDER BY created_at DESC
    """, (movie_id,))
    reviews = cursor.fetchall()
    # Get average rating & count
    avg, cnt = get_movie_ratings(movie_id)
    return render_template(
```

```
        'comments.html',
        movie=movie,
        reviews=reviews,
        avg_rating=round(avg, 1),
        rating_count=cnt
    )
if __name__ == '__main__':
    app.run(debug=True)
```

INDEX.HTML

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>Movie Ticket Booking System</title>
    <link rel="stylesheet" href="{{ url_for('static', filename='style.css') }}">
    <style>
        /* background + layout kept as before */
        body {
            margin: 0;
            padding: 0;
            background: url('{{ url_for("static", filename="images/bg1.jpg") }}') no-repeat center center fixed;
```

```
    background-size: cover;
    font-family: Arial, sans-serif;
}
.header-bar {
    background: rgba(0,0,0,0.75);
    color: #ffeb00;
    padding: 15px;
    text-align: center;
    font-size: 32px;
    font-weight: bold;
    letter-spacing: 1px;
    box-shadow: 0 3px 8px rgba(0,0,0,0.4);
}
.content-box {
    background: rgba(255,255,255,0);
    width: 70%;
    margin: 40px auto;
    padding: 20px;
    text-align: center;
}
.room-img {
    width: 80%;
```

```
    border-radius: 12px;
    box-shadow: 0 4px 12px rgba(0,0,0,0.5);
}
.buttons {
    margin-top: 20px;
}
.buttons a {
    text-decoration: none;
    padding: 12px 25px;
    margin: 10px;
    border-radius: 10px;
    font-size: 18px;
    display: inline-block;
    color: white;
    font-weight: bold;
}
.blue-btn  { background: #007bff; }
.green-btn { background: #28a745; }
.orange-btn { background: #ff8800; }
.gray-btn { background: #444; }
.buttons a:hover { opacity: 0.9; transform: scale(1.03); }
```

```
.footer { text-align: center; margin-top: 40px; color: white; font-size: 18px; text-shadow: 2px 2px 4px black; padding-bottom: 25px; }
</style>
</head>
<body>
<div class="header-bar">🎬 Movie Ticket Booking System</div>
<div class="content-box">
<img src="{{ url_for('static', filename='images/theater.jpg') }}" class="room-img">
<div class="buttons">
<a href="{{ url_for('movies') }}" class="blue-btn">Movie List</a>
<a href="{{ url_for('payment') }}" class="green-btn">Pay (If booked)</a>
<a href="{{ url_for('payment') }}" class="orange-btn">Retry Payment</a>
<a href="{{ url_for('admin') }}" class="gray-btn">Admin</a>
<!-- NEW -->
<a href="{{ url_for('review') }}" class="blue-btn">Review & Feedback</a>
</div>
</div>
<div class="footer">
<p>📍 Chennai, India | 📞 +91-6384994375</p>
<p>✉️ support@movietickets.com</p>
</div>
```

```
</body>

</html>
```

TICKET.HTML

```
<!DOCTYPE html>

<html>

<head>

   <meta charset="UTF-8">

   <title>Movie Ticket</title>

   <style>

      body {

         margin: 0;

         padding: 0;

         font-family: 'Arial', sans-serif

  background: #101010;

         display: flex;

         justify-content: center;

         align-items: center;

         height: 100vh;

      }

   </style>

</head>
```

```
<body>
<div class="ticket-container">
  <div class="ticket-header">
    Movie Ticket
  </div>
  <div class="ticket-body">
 <div class="ticket-row">
        <div class="label">Movie</div>
        <div class="value">{{ ticket.title }}</div>
      </div>
      <div class="ticket-row">
        <div class="label">Customer</div>
        <div class="value">{{ ticket.customer_name }}</div>
      </div>
      <div class="ticket-row">0.
    <div class="label">Show Time</div>
        <div class="value">{{ ticket.show_time }}</div>
      </div>
      <div class="ticket-row">
        <div class="label">Seat No</div>
        <div class="value">{{ ticket.seat_no }}</div>
```

```
        </div>
        <div class="ticket-row">
            <div class="label">Price</div>
            <div class="value">₹{{ ticket.price }}</div>
        </div>
        <div class="line"></div>
        <div class="qr-box">
            <img src="https://api.qrserver.com/v1/create-qr-
code/?size=150x150&data=TICKET{{ ticket.id }}" alt="QR Code">
        </div>a
            Scan this QR at entry gate
        </div>
        <a class="home-link" href="{{ url_for('movies') }}">Back to Movies</a>
</div>
</div>
</body>
</html>
```

ADMIN.HTML

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
```

```
    <title>Admin Login</title>
    <link rel="stylesheet" href="{{ url_for('static', filename='style.css') }}">
    <style>
        body {
            margin: 0;
            padding: 0;
            height: 100vh;
            background: url('{{ url_for("static", filename="images/bg.jpg") }}') no-repeat
center center fixed;
            background-size: cover;
            font-family: Arial, sans-serif;
        }
        .center-box {
            width: 350px;
            background: rgba(0, 0, 0, 0.65);
            padding: 25px;
            border-radius: 12px;
            color: white;
            text-align: center;
            margin: 120px auto;
            box-shadow: 0 0 10px black;
        }
```

```
    .center-box input {
        width: 90%;
        padding: 10px;
        margin: 8px 0;
        border-radius: 8px;
        border: none;
    }
    .center-box button {
        width: 95%;
        padding: 10px;
        margin-top: 10px;
        border-radius: 8px;
        border: none;
        background: #28a745;
        color: white;
        font-size: 16px;
        cursor: pointer;
    }
    .center-box button:hover {
        background: #218838;
    }
</style>
```

```
</head>
<body>
    <div class="center-box">
        <h2>Admin Login</h2>
        <form method="POST">
            <input type="text" name="username" placeholder="Username" required><br>
            <input type="password" name="password" placeholder="Password"
required><br>
            <button type="submit">Login</button>
        </form>
    </div>
</body>
</html>
```

APPENDIX B

(SCREENSHOTS)

1. HOME PAGE

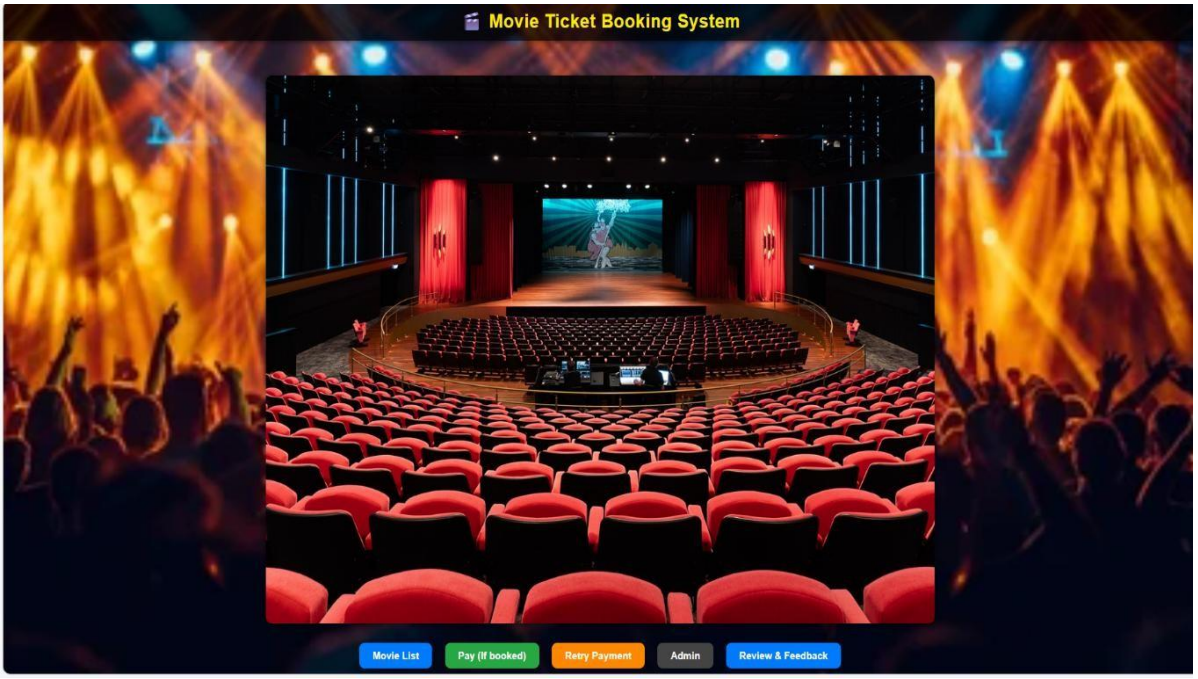

Figure B.1 HOME PAGE

2. MOVIE LIST

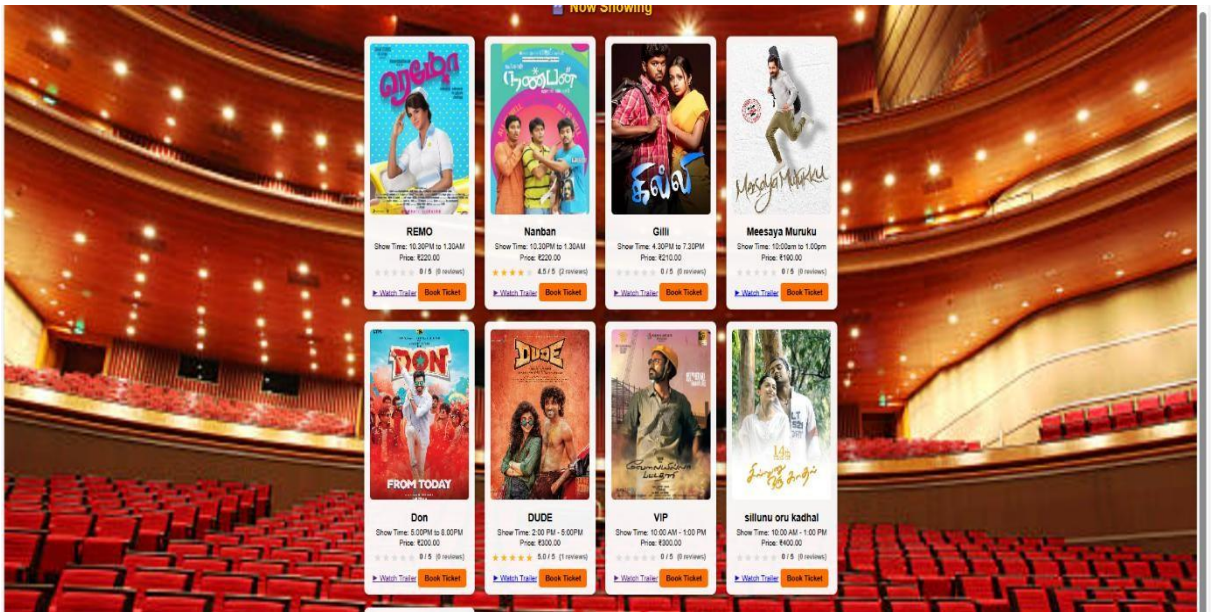

Figure B.2 MOVIE LIST

3. BOOK TICKET

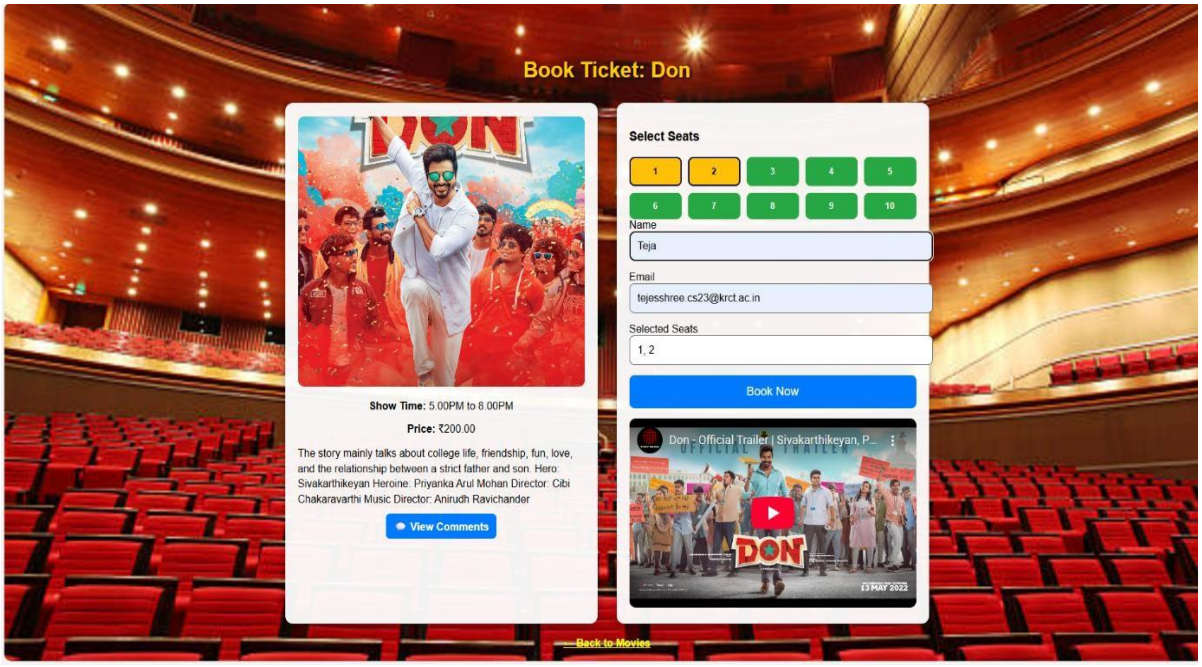

Figure B.3 BOOK TICKET

4. PAYMENT METHOD

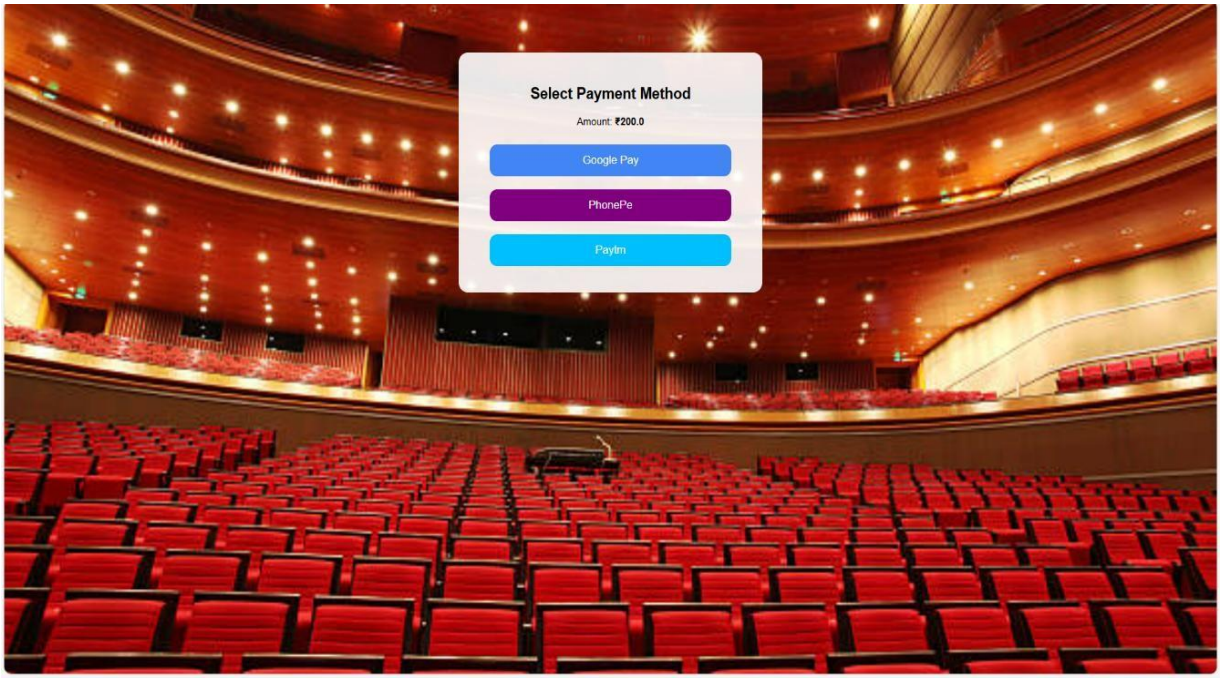

Figure B.4.1 PAYMENT METHOD

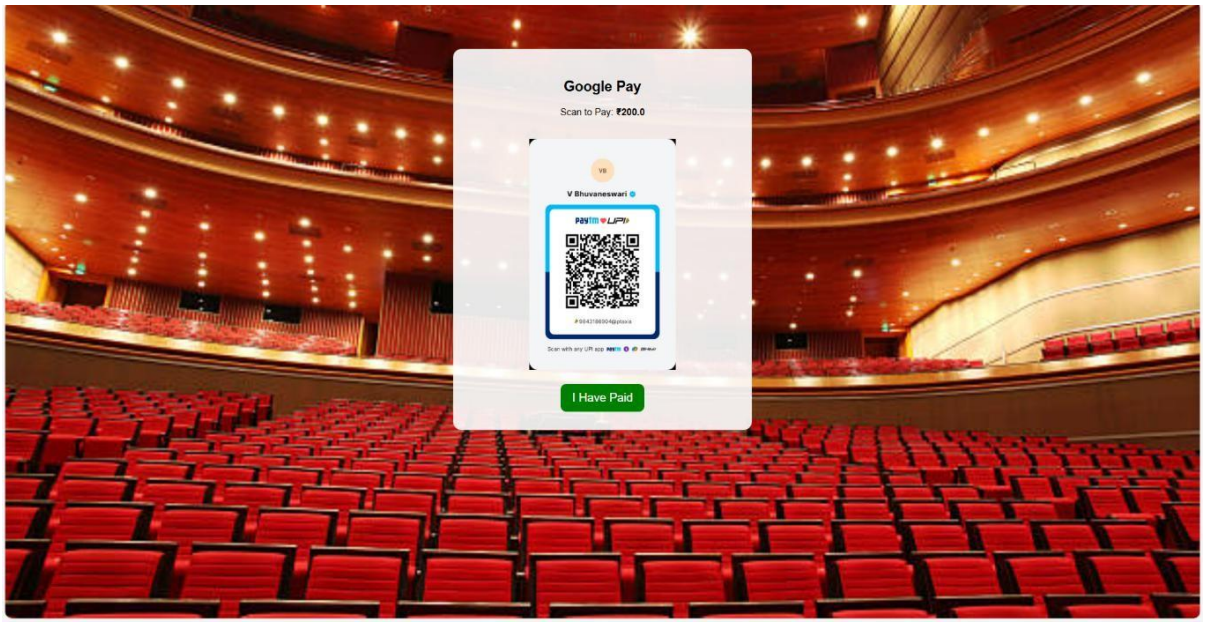

Figure B.4.2 QRCODE

5. PAYMENT SUCCESS

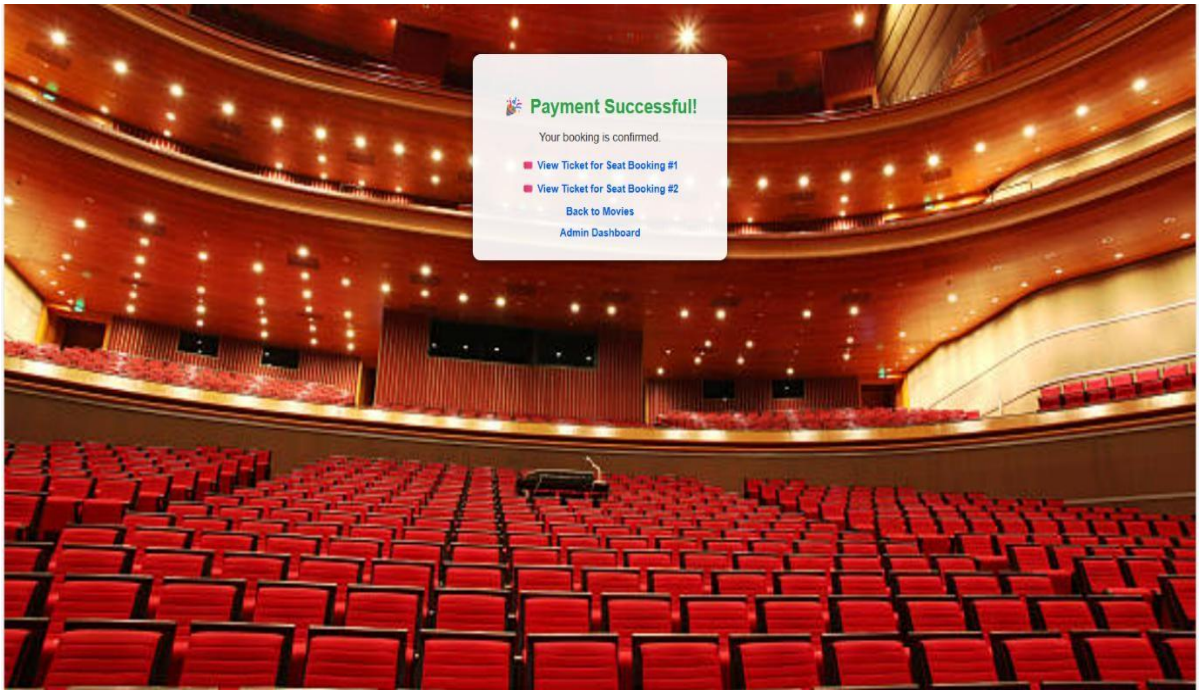

Figure B.5 PAYMENT SUCCESS

6. VIEW TICKET

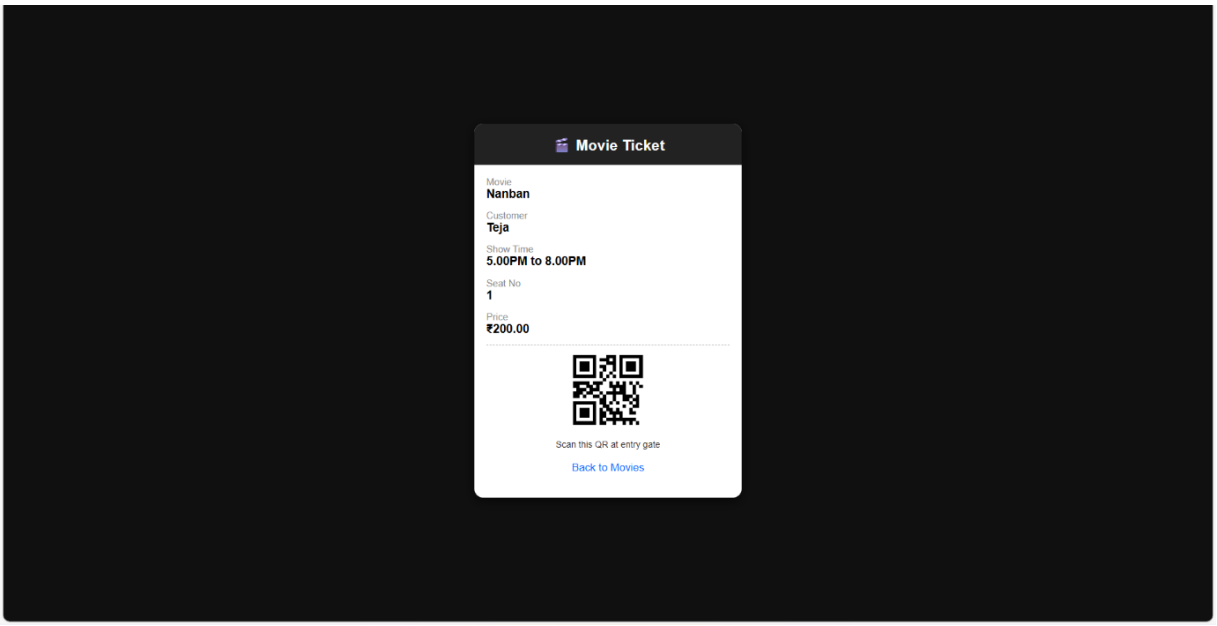

Figure B.6 VIEW TICKET

7. ADMIN LOGIN

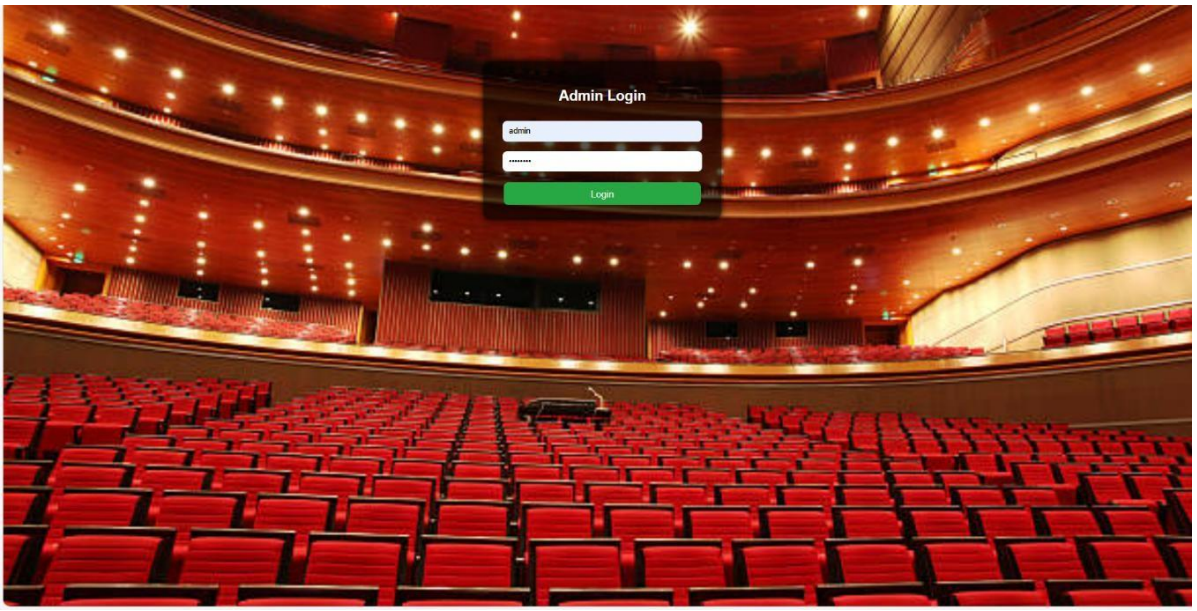

Figure B.7 ADMIN LOGIN

8. ADMIN DASHBOARD

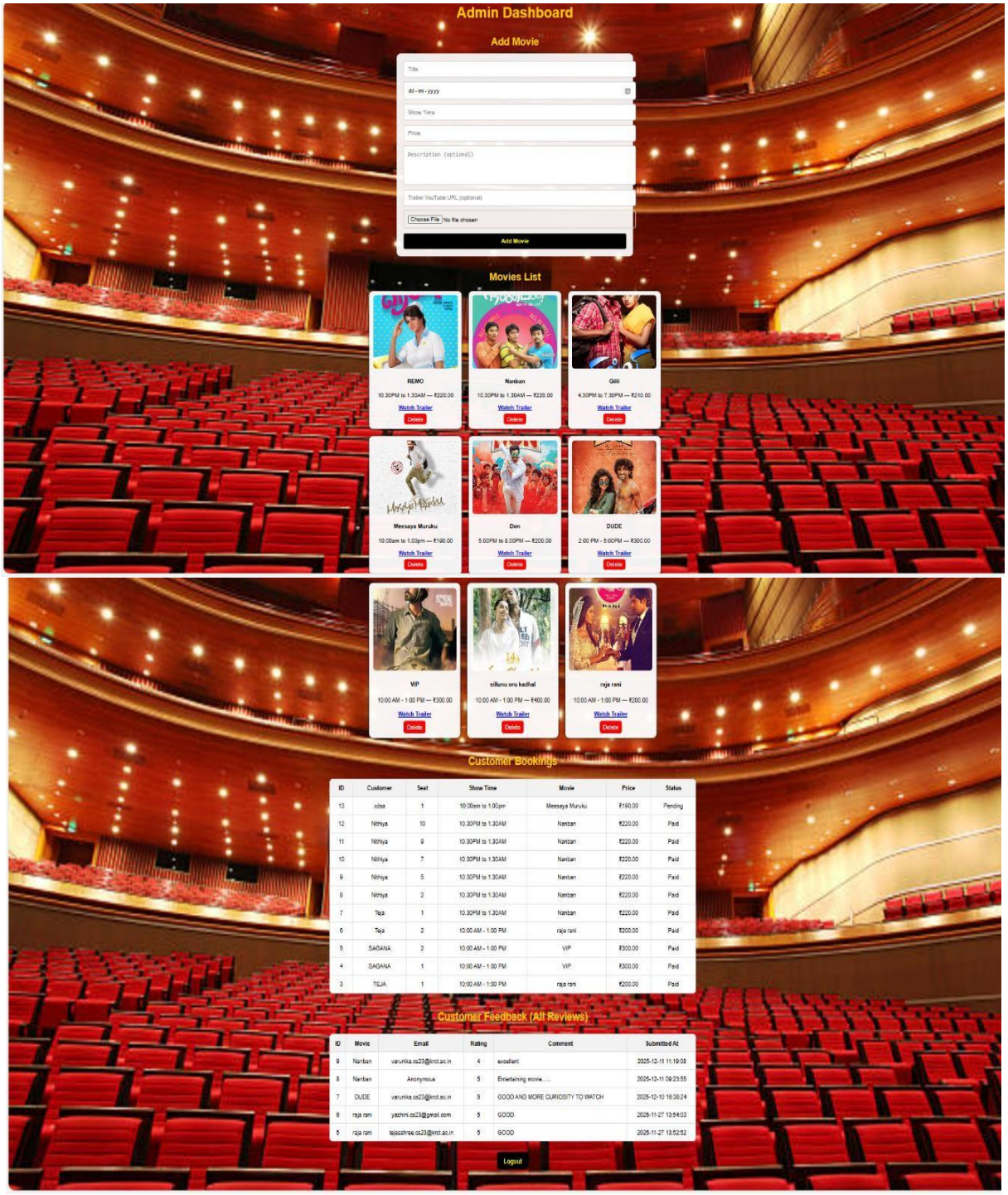

Figure B.8 ADMIN DASHBOARD

9. VIEW COMMENTS

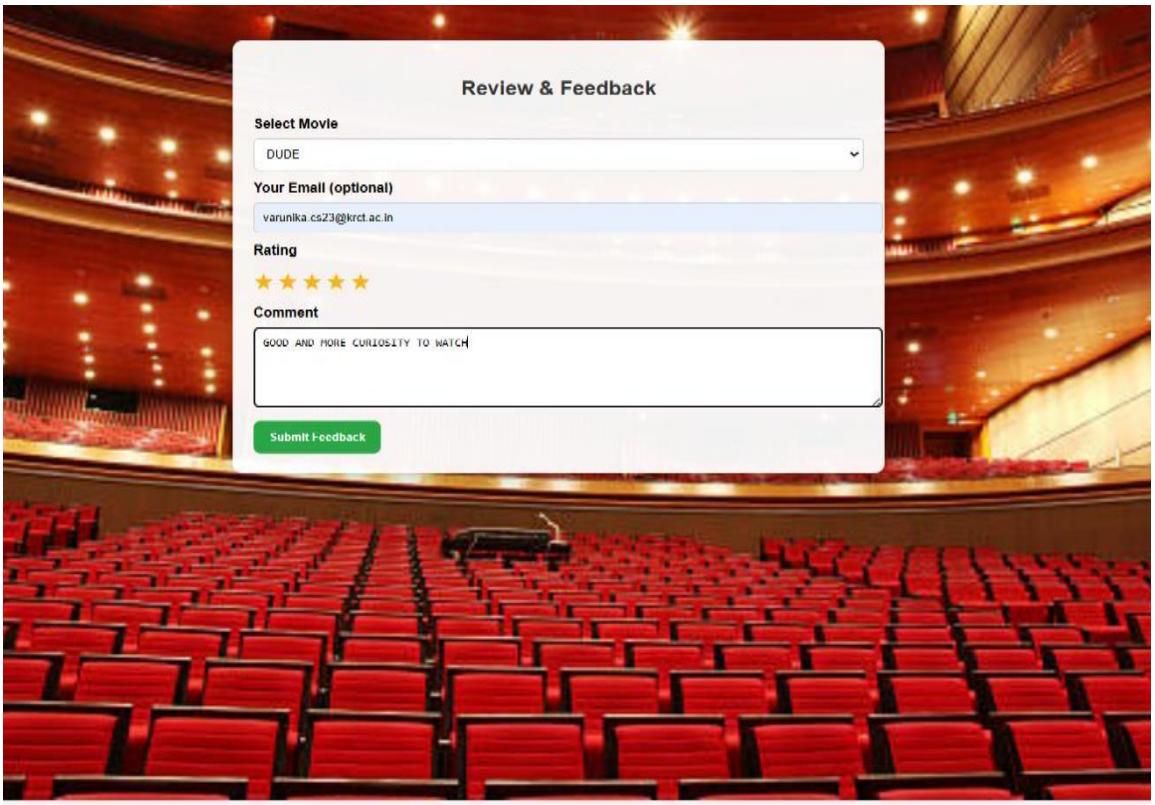

Figure B.9 VIEW COMMENTS

REFERENCES

REFERENCE BOOKS

1. Mark Lutz (2013), Learning Python, O'Reilly Media, 5th Edition, pp. 1–1600.
2. Miguel Grinberg (2018), Flask Web Development, O'Reilly Media, 2nd Edition, pp. 1– 250.
3. Robin Nixon (2018), Learning PHP, MySQL & JavaScript, O'Reilly Media, 5th Edition, pp. 1–800.
4. Wesley J. Chun (2010), Core Python Programming, Prentice Hall, 2nd Edition, pp. 1– 1120.

REFERENCE WEBSITES

1. Flask Documentation: https://flask.palletsprojects.com/
2. MDN Web Docs (HTML, CSS, JS): https://developer.mozilla.org/
3. MySQL Official Documentation: https://dev.mysql.com/doc/
4. W3Schools HTML/CSS Tutorials: https://www.w3schools.com/