RESTAURANT MANAGEMENT

A MINI-PROJECT BY:

NITHIESH S	**230701215**
PRADEEP KUMAR S	**230701230**
PRAJAN RAJ R	**230701232**

in partial fulfillment of the award of the degree

OF

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE AND ENGINEERING

RAJALAKSHMI ENGINEERING COLLEGE, CHENNAI

An Autonomous Institute

CHENNAI

NOVEMBER 2024

BONAFIDE CERTIFICATE

Certified that this project **"MOVIE TICKET BOOKING SYSTEM"** is the bonafide work of **"NITHIESH S , PRADEEP KUMAR S , PRAJAN RAJ R"** who carried out the project work under my supervision.

Submitted for the practical examination held on _____

SIGNATURE

Ms. ASWANA LAL
Asst. Professor
Computer Science and Engineering,
Rajalakshmi Engineering College
(Autonomous),
Thandalam,Chennai-602105

INTERNAL EXAMINER **EXTERNAL EXAMINER**

ABSTRACT

The **Movie Ticket Booking System** is an innovative software solution designed to simplify and automate the movie ticket reservation process. Built using **Java, JSP, and MySQL**, this system caters to both cinema administrators and customers by streamlining movie management and seat booking.

Key Features:

- **User Authentication**: Secure login and registration for customers and administrators.
- **Movie Browsing & Booking**: Users can explore available movies, select showtimes, and reserve seats in real-time.
- **Admin Management Dashboard**: Allows administrators to manage movie schedules, update showtimes, and view booking reports.
- **Database Integration**: A robust MySQL backend ensures secure storage and efficient management of user data, movie details, and seat reservations.
- **Architecture**: Implements a three-tier MVC model, enhancing scalability and maintainability.

This project delivers a user-friendly platform that reduces operational complexities, enhances customer satisfaction, and improves overall efficiency for movie theaters. With potential for further development, such as real payment gateway integration and mobile application support, this system offers a scalable and future-proof solution for cinema operations.

TABLE OF CONTENTS

Movie Ticket Booking System

1. INTRODUCTION

1.1 Introduction

The Movie Ticket Booking System is a web-based application designed to automate the process of reserving movie tickets. This system allows users to browse movies, select showtimes, book seats, and manage bookings online. It also includes an admin interface for managing movies and reservations efficiently.

1.2 Objectives

- Simplify the movie ticket booking process for users.
- Provide secure and user-friendly interfaces for customers and administrators.
- Maintain a reliable and scalable database for managing movie schedules, reservations, and user information.
- Reduce the manual effort involved in managing ticket reservations.

1.3 Modules

1. User Module: Enables user registration, login, and booking management.
2. Admin Module: Allows administrators to add, edit, and delete movies, showtimes, and seats.
3. Booking Module: Facilitates seat selection, ticket booking, and confirmation.
4. Payment Module: Integrates a dummy payment gateway for simulated transactions.

2. SURVEY OF TECHNOLOGIES

2.1 Software Description

The Movie Ticket Booking System is developed using the following technologies:

- Java: Backend application logic.
- JSP (Java Server Pages): Frontend dynamic content rendering.
- MySQL: Relational database management for data storage and retrieval.
- Apache Tomcat: Web server for deploying and running the application.

2.2 Languages

2.2.1 SQL

SQL (Structured Query Language) is used to define and manipulate the data stored in the MySQL database. It handles operations like inserting, updating, and retrieving movie schedules, user information, and booking records.

2.2.2 Python

Python is employed optionally for auxiliary tasks such as testing, scripting, or generating analytics reports based on booking data.

3. REQUIREMENTS AND ANALYSIS

3.1 Requirement Specification

Functional Requirements

- Users must be able to register and log in.
- Users should browse movies, select showtimes, and reserve tickets.
- Admins should manage movies, schedules, and user reservations.

Non-Functional Requirements

- The system should ensure data security and prevent unauthorized access.
- It must handle multiple concurrent users efficiently.
- The system should be scalable to accommodate future enhancements.

3.2 Hardware and Software Requirements

Hardware Requirements

- Processor: Intel i3 or above.
- RAM: Minimum 4GB.
- Storage: 10GB free disk space.

Software Requirements

- Operating System: Windows/Linux/Mac.
- Java Development Kit (JDK): Version 8 or above.
- Apache Tomcat ServerVersion 9.
- MySQL Database: Version 5.7 or above.
- IDE: Eclipse or IntelliJ IDEA.

4. PROGRAM CODE

4.1 MAIN CODE:

```
package com.raman;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.SQLException;
import java.util.Scanner;
import static com.raman.UserLogin.*;

public class TicketBookingSystem {

    public static void LogIn() {

        Scanner scanner = new Scanner(System.in);
        UserLogin ul=new UserLogin(); //object of userlogin
        AdminControl ac = new AdminControl();  //object of admincontrol
        UserControl uc = new UserControl();    //object of usercontrol

        validatingCredentials();
         var name = username;

        if(ul.role.equalsIgnoreCase("admin")){
          System.out.println("What would you like to do Admin ?");
          System.out.println("1. Add Theatre\n2. Edit Theater\n3. Remove Theater\n4. Add
Movie\n5. Edit Movie\n6. Remove Movie");
          int choice = scanner.nextInt();
           scanner.nextLine(); // Consume newline character

           switch (choice)
           {
              case 1:
                 ac.addTheater();
                 break;
              case 2:
                 ac.editTheater();
                 break;
              case 3:
                 ac.removeTheater();
                 break;
```

```
            case 4:
                ac.addMovie();
                break;
            case 5:
                ac.editMovie();
                break;
            case 6:
                ac.removeMovie();
                break;
            default:
                System.out.println("Invalid choice. Please try again.");}

        }else if (ul.role.equalsIgnoreCase("user")) {
            System.out.println("Welcome to Movieplex!!!   " + name);
            uc.displayMovieChart();
        }else {
            System.out.println("Invalid Credentials!!! Try Again.....");
        }

    }
    public static class UserRegistration {
        private JDBC jdbc = new JDBC();
        private Connection connection = jdbc.establishConnection();
        private Scanner scanner = new Scanner(System.in);

        public void createNewUser() {
            System.out.println("Enter Username:");
            String username = scanner.nextLine();

            System.out.println("Enter Password:");
            String password = scanner.nextLine();

            System.out.println("Enter Role (Admin/User):");
            String role = scanner.nextLine();

            System.out.println("Enter Mail ID:");
            String mailId = scanner.nextLine();

            // SQL query to insert a new user with Mail_Id
            String query = "INSERT INTO UserCredentials (Username, Password, Role, Mail_Id)
VALUES (?, ?, ?, ?)";
```

```
    try (PreparedStatement statement = connection.prepareStatement(query)) {
      statement.setString(1, username);
      statement.setString(2, password);
      statement.setString(3, role);
      statement.setString(4, mailId);

      int rowsAffected = statement.executeUpdate();
      if (rowsAffected > 0) {
        System.out.println("User registered successfully.");
      } else {
        System.out.println("Failed to register user.");
      }
    } catch (SQLException e) {
      System.err.println("Error while registering user: " + e.getMessage());
    }
  }
}

public static void main(String[] args) {
   Scanner scanner = new Scanner(System.in);
   UserRegistration userRegistration = new UserRegistration();
   UserLogin userLogin = new UserLogin();

   while (true) {
      System.out.println("\n1. Register New User");
      System.out.println("2. Login");
      System.out.println("3. Exit");
      System.out.print("Choose an option: ");
      int choice = scanner.nextInt();
      scanner.nextLine(); // Consume newline
      switch (choice) {
         case 1:
            userRegistration.createNewUser();
            break;
         case 2:
            LogIn();
            break;
         case 3:
            System.out.println("Exiting...");
            System.exit(0);
```

```
            default:
                System.out.println("Invalid choice. Try again.");
        }
    }
  }
}
```

4.2.LOGIN:

```
package com.raman;

import java.sql.*;
import java.util.Scanner;

import static com.raman.MakePayment.scanner;

public class UserLogin  {

   static JDBC jdbc = new JDBC();
   static Connection connection = jdbc.establishConnection();
   public static String role="";
   public static String username;

   Scanner sc = new Scanner(System.in);

   public static void validatingCredentials() {
      System.out.println("Enter Username: ");
      username = scanner.nextLine();

      System.out.println("Enter password: ");
      String password = scanner.nextLine();

      // Query to retrieve user credentials
       String query = "SELECT Role FROM UserCredentials WHERE Username = ? AND
Password = ?";
       try {
       PreparedStatement statement = connection.prepareStatement(query);
       statement.setString(1, username);
       statement.setString(2, password);

       ResultSet resultSet = statement.executeQuery();

       if (resultSet.next()) {
```

```
            // User exists, save the role
            role = resultSet.getString("Role");
            System.out.println("User authenticated successfully. Role: " + role);

        } else {
            // User doesn't exist
            System.out.println("Warning.");
        }

        // Close resources
        resultSet.close();
        statement.close();
        connection.close();
    } catch (SQLException e) {
        System.err.println("Error validating user credentials: " + e.getMessage());
    }

    }
}
```

4.3.USER CONTROL:

```
package com.raman;
import java.util.*;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

public class UserControl{

    JDBC jdbc = new JDBC();
    Connection connection=jdbc.establishConnection();

    public static int movieId=0;
    public static  int theaterId= 0;
    public static int numberOfSeats =0;
    public static int totalPrice = 0;
    public static String uniqueId=null;
```

```
    public static String showDate=null;
    public static String showTime=null;
    public static String emailId=null;
    public static String movieName=null;
    public static String genre=null;
    public static String theaterName=null;
    public static int finalPrice = 0;
    public static String showTiming=null;

    public void displayMovieChart() {

        try {
            UserControl ua = new UserControl();
            Scanner sc = new Scanner(System.in);

            // Execute SQL query to retrieve data from MovieDetails and
TheaterDetails tables
            Statement statement = connection.createStatement();
            String query = "SELECT MovieDetails.MovieId, MovieName, Genre,
Duration_in_min, TicketPrice, MovieDetails.TheaterId, TheaterName, Location,
ShowDate1,ShowDate2,ShowDate3 FROM MovieDetails INNER JOIN
TheaterDetails ON MovieDetails.TheaterId = TheaterDetails.TheaterId INNER
JOIN ShowDetails ON MovieDetails.MovieId = ShowDetails.MovieId;";
            ResultSet resultSet = statement.executeQuery(query);

            // Display combined table

            System.out.println("+-----------+------------------------------+---------------
------+--------------+---------+-------------+------------------------------+-----------
--------+-------------+-----------------+-----------------+");
            System.out.println("| Movie ID  |          Movie Name          |       Genre
|   Duration    |  Price   | Theater ID  |       Theater Name           |    Location       |
Show Date 1  |   Show Date 2    |    Show Date 3   |");
            System.out.println("+-----------+------------------------------+---------------
------+--------------+---------+-------------+------------------------------+-----------
--------+-------------+-----------------+-----------------+");
```

```
        while (resultSet.next()) {
            int movieId = resultSet.getInt("MovieId");
            String movieName = resultSet.getString("MovieName");
            String genre = resultSet.getString("Genre");
            int duration = resultSet.getInt("Duration_in_min");
            int ticketPrice = resultSet.getInt("TicketPrice");
            int theaterId = resultSet.getInt("TheaterId");
            String theaterName = resultSet.getString("TheaterName");
            String location = resultSet.getString("Location");
            String showDate1 = resultSet.getString("ShowDate1");
            String showDate2 = resultSet.getString("ShowDate2");
            String showDate3 = resultSet.getString("ShowDate3");

            System.out.printf("| %-9s | %-30s | %-20s | %-13s | %-8s | %-12s | %-
30s | %-18s | %-12s | %-16s |",
                    movieId, movieName, genre, duration, ticketPrice, theaterId,
theaterName, location, showDate1, showDate2);
            System.out.println("    "+showDate3);
        }

        System.out.println();
        System.out.println("PLease select the following available actions.");
        boolean isValidInput = false;

        while (!isValidInput) {
            // Display menu options
            System.out.println("1. Book A ticket");
            System.out.println("2. Exit");
            System.out.println();
            System.out.print("Enter your choice: ");

            // Read user input
            int choice = sc.nextInt();
            sc.nextLine();
```

```
            // Process user input
            switch (choice) {
                case 1:
                    // Perform action for option 1
                    isValidInput = true;
                    ua.selectionProcess();
                    break;
                case 2:
                    // Perform action for option 2
                    isValidInput = true;
                    System.out.println("Please visit us again.");
                    break;
                default:
                    // Invalid input, prompt user to enter again
                    System.out.println("Invalid input. Please enter a valid option.");
                    break;
            }
        }

        sc.close();


        // Close connection
        resultSet.close();
        statement.close();
        connection.close();
    } catch (SQLException e) {
        e.printStackTrace();
    }

}
//method to book the ticket
public void selectionProcess() throws SQLException{

   Scanner sc = new Scanner(System.in);
   Countdown cd = new Countdown();
```

```
        UserControl userControl = new UserControl();

        System.out.println("Please Be patient");

         int userId = -1; // Default value if user is not found
         PreparedStatement statement = null;
         ResultSet resultSet = null;

         try {
            String sql = "SELECT UserId FROM UserCredentials WHERE
Username = ?";
            statement = connection.prepareStatement(sql);
            statement.setString(1, new UserLogin().username);
            resultSet = statement.executeQuery();

            if (resultSet.next()) {
               userId = resultSet.getInt("UserId");
            }
            }catch(SQLException e) {
               e.printStackTrace();
            }

         System.out.println("User Id is: "+ userId);
         System.out.println("Press any button to proceed");
         sc.nextLine();
         System.out.println("Enter the name for billing purpose: ");
         String username = sc.nextLine();

         System.out.println("Enter movieId: ");
         movieId = sc.nextInt();

         System.out.println("Enter theaterId: ");
         theaterId = sc.nextInt();

      // Fetch movie details based on MovieId
         String movieDetailsQuery = "SELECT MovieName, Duration_in_min,
```

```
Genre, TicketPrice FROM MovieDetails WHERE MovieId = ?";
        PreparedStatement movieDetailsStatement =
connection.prepareStatement(movieDetailsQuery);
        movieDetailsStatement.setInt(1, movieId);
        ResultSet movieDetailsResult = movieDetailsStatement.executeQuery();

        movieName = null;
        int duration = 0;
        genre = null;
        int ticketPrice = 0;

        if (movieDetailsResult.next()) {
            movieName = movieDetailsResult.getString("MovieName");
            duration = movieDetailsResult.getInt("Duration_in_min");
            genre = movieDetailsResult.getString("Genre");
            ticketPrice = movieDetailsResult.getInt("TicketPrice");
        } else {
            System.out.println("Movie with ID " + movieId + " not found.");
            return;
        }

    // Fetch theater details based on TheaterId
        String theaterDetailsQuery = "SELECT TheaterName, Location FROM
TheaterDetails WHERE TheaterId = ?";
        PreparedStatement theaterDetailsStatement =
connection.prepareStatement(theaterDetailsQuery);
        theaterDetailsStatement.setInt(1, theaterId);
        ResultSet theaterDetailsResult = theaterDetailsStatement.executeQuery();

        theaterName = null;
        String theaterLocation = null;

        if (theaterDetailsResult.next()) {
            theaterName = theaterDetailsResult.getString("TheaterName");
            theaterLocation = theaterDetailsResult.getString("Location");
        } else {
```

```
            System.out.println("Theater with ID " + theaterId + " not found.");
            return;
        }

        userControl.validateDate();

        System.out.println("Select the Show Id as per the corresponding Timing\n\n1. Morning(09:00 AM)\n2. Afternoon(01:00 PM\n3. Evening(05:00 PM\n4. Night(09:00 PM");

        while (true) {
            showTime = sc.nextLine();

            // Check if the input is valid (1, 2, 3, or 4)
            if (showTime.matches("[1-4]")) {
                // Valid input
                break;
            } else {
                System.out.println("Please enter 1, 2, 3, or 4.");
            }
        }

        showTiming = "";

        int showTimeValue = Integer.parseInt(showTime);

        switch(showTimeValue) {
        case 1: showTiming="09:00 AM";
            break;
        case 2: showTiming="01:00 PM";
          break;
        case 3: showTiming="05:00 PM";
          break;
        case 4: showTiming="09:00 PM";
          break;
```

```
        }

        System.out.println("Enter numberOfSeats: ");
        numberOfSeats = sc.nextInt();

        sc.nextLine();
        System.out.println("Enter Your E-mail Id: ");
        emailId = sc.nextLine();

        System.out.println("");
        new UserControl().validateAndStoreUniqueId();

        totalPrice=calculateTotalPrice(movieId, theaterId, numberOfSeats);
        finalPrice=totalPrice;

        if(totalPrice==0) {
           System.exit(showTimeValue);
        }


        // Insert booking details into the database
        String insertQuery = "INSERT INTO TransactionalDetails (UserId,
Username, MovieId, MovieName, TheaterId, TheaterName, Location,
NumberOfSeats, Price_before_coupon, Email_id, ShowDate, ShowTime,
UniqueId) "
        + "VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)";

        PreparedStatement preparedStatement =
connection.prepareStatement(insertQuery);
        preparedStatement.setLong(1, userId);
        preparedStatement.setString(2, username);
        preparedStatement.setInt(3, movieId);
        preparedStatement.setString(4, movieName);
        preparedStatement.setInt(5, theaterId);
        preparedStatement.setString(6, theaterName);
        preparedStatement.setString(7, theaterLocation);
```

```
        preparedStatement.setInt(8, numberOfSeats);
        preparedStatement.setInt(9, totalPrice);
        preparedStatement.setString(10, emailId);
        preparedStatement.setString(11, showDate);
        preparedStatement.setString(12, showTiming);
        preparedStatement.setString(13, uniqueId);
        preparedStatement.executeUpdate();

        cd.timeCheck();

        // Close resources
        preparedStatement.close();
        connection.close();
    }

    public void validateDate() {

        Scanner scanner = new Scanner(System.in);
        System.out.println("Please Enter the date(YY-MM-DD) on which you
would like to watch the movie : \n\nNote* Kindly refer the above table to see the
movie availibility");
        showDate = scanner.nextLine();
        boolean isValidDate = false;

        while (!isValidDate) {

          // Check if showDate is empty
           if (showDate.isEmpty()) {
              System.out.println("Date cannot be empty. Please enter again:");
              showDate = scanner.nextLine(); // Read the next line again
              continue; // Skip further processing in this iteration
           }

           // SQL query to check if the provided date exists for the current movie
and theater
           String query = "SELECT COUNT(*) AS CountExists " +
```

```
                    "FROM ShowDetails sd " +
                    "JOIN MovieDetails md ON sd.MovieId = md.MovieId " +
                    "WHERE (md.MovieId = ?) " +
                    "AND (sd.ShowDate1 = ? OR sd.ShowDate2 = ? OR
sd.ShowDate3 = ?)";

          try (PreparedStatement preparedStatement =
connection.prepareStatement(query)) {
              preparedStatement.setInt(1, movieId);
              preparedStatement.setString(2, showDate);
              preparedStatement.setString(3, showDate);
              preparedStatement.setString(4, showDate);

              ResultSet resultSet = preparedStatement.executeQuery();

              // If the count is greater than 0, the date is valid
              if (resultSet.next() && resultSet.getInt("CountExists") > 0) {
                  System.out.println("Date is valid.");
                  isValidDate = true;
              } else {
                  System.out.println("Date is not valid. Please enter again:");

              showDate = scanner.nextLine();
              }
          } catch (SQLException e) {
           // TODO Auto-generated catch block
           e.printStackTrace();
         }
       }
   }

   public  int calculateTotalPrice(int movieId, int theaterId, int numberOfSeats)
throws SQLException {
       int ticketPrice = 0;

       // Fetch ticket price based on MovieId and TheaterId
```

```
        String ticketPriceQuery = "SELECT TicketPrice FROM MovieDetails
WHERE MovieId = ? AND TheaterId = ?";
        PreparedStatement ticketPriceStatement =
connection.prepareStatement(ticketPriceQuery);
        ticketPriceStatement.setInt(1, movieId);
        ticketPriceStatement.setInt(2, theaterId);
        ResultSet ticketPriceResult = ticketPriceStatement.executeQuery();
        if (ticketPriceResult.next()) {
            ticketPrice = ticketPriceResult.getInt("TicketPrice");
        } else {
            System.out.println("Ticket price not found for the given movie and
theater combination.");
            return 0;
        }
        return numberOfSeats * ticketPrice;
    }

    //method for the functions related to unique id

 // Method to validate and store unique ID
    public void validateAndStoreUniqueId() {
        Scanner scanner = new Scanner(System.in);

        boolean isValidId = false;

        while (!isValidId) {
            System.out.println("Please enter a uniqueId of your choice: ");
            uniqueId = scanner.nextLine();

            // Check if the uniqueId is already present in the database
            if (isUniqueIdExists(uniqueId)) {
                System.out.println("This ID is not unique. Please try another one.");
            } else {
                isValidId = true;
            }
        }
```

```
    }

 // Method to check if the uniqueId already exists in the database
    private boolean isUniqueIdExists(String uniqueId) {
        boolean exists = false;
        try {
            PreparedStatement preparedStatement =
connection.prepareStatement("SELECT COUNT(*) FROM TransactionalDetails
WHERE UniqueId = ?");
            preparedStatement.setString(1, uniqueId);
            ResultSet resultSet = preparedStatement.executeQuery();
            if (resultSet.next()) {
                int count = resultSet.getInt(1);
                exists = count > 0;
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return exists;
    }

}
```

4.4.ADMIN CONTROL:

```
package com.raman;

import java.sql.*;
import java.util.Scanner;

public class AdminControl {

    JDBC jdbc = new JDBC();
    Connection connection = jdbc.establishConnection();

    // Method to add a new theater
    public void addTheater() {
        try {
```

```
            Scanner scanner = new Scanner(System.in);
            System.out.println("Enter theater name:");
            String theaterName = scanner.nextLine();
            System.out.println("Enter theater location:");
            String location = scanner.nextLine();

            // Prepare SQL statement
            String sql = "INSERT INTO TheaterDetails (TheaterName, Location)
VALUES (?, ?)";
            PreparedStatement statement = connection.prepareStatement(sql);
            statement.setString(1, theaterName);
            statement.setString(2, location);

            // Execute SQL statement
            int rowsAffected = statement.executeUpdate();
            if (rowsAffected > 0) {
                System.out.println("Theater added successfully.");
            } else {
                System.out.println("Failed to add theater.");
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

    // Method to edit theater details
    public void editTheater() {
        try {
            Scanner scanner = new Scanner(System.in);
            System.out.println("Enter theater ID to edit:");
            int theaterId = scanner.nextInt();
            scanner.nextLine(); // Consume newline character

            System.out.println("Enter new theater name:");
            String theaterName = scanner.nextLine();
            System.out.println("Enter new theater location:");
```

```
            String theaterLocation = scanner.nextLine();

            // Prepare SQL statement
            String sql = "UPDATE TheaterDetails SET TheaterName = ?, Location =
? WHERE TheaterId = ?";
            PreparedStatement statement = connection.prepareStatement(sql);
            statement.setString(1, theaterName);
            statement.setString(2, theaterLocation);
            statement.setInt(3, theaterId);

            // Execute SQL statement
            int rowsAffected = statement.executeUpdate();
            if (rowsAffected > 0) {
                System.out.println("Theater updated successfully.");
            } else {
                System.out.println("Failed to update theater.");
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

    // Method to remove a theater
    public void removeTheater() {
        try {
            Scanner scanner = new Scanner(System.in);
            System.out.println("Enter theater ID to remove:");
            int theaterId = scanner.nextInt();

            // Prepare SQL statement
            String sql = "DELETE FROM TheaterDetails WHERE TheaterId = ?";
            PreparedStatement statement = connection.prepareStatement(sql);
            statement.setInt(1, theaterId);

            // Execute SQL statement
            int rowsAffected = statement.executeUpdate();
```

```
            if (rowsAffected > 0) {
                System.out.println("Theater removed successfully.");
            } else {
                System.out.println("Failed to remove theater.");
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

    // Method to add a new movie
    public void addMovie() {
        try {
            Scanner scanner = new Scanner(System.in);
            System.out.println("Enter movie name:");
            String movieName = scanner.nextLine();
            System.out.println("Enter movie duration:");
            int duration = scanner.nextInt();
            scanner.nextLine(); // Consume newline character
            System.out.println("Enter TheaterId:");
            int TheaterId = scanner.nextInt();
            scanner.nextLine();

            // Prepare SQL statement
            String sql = "INSERT INTO MovieDetails (MovieName,
Duration_in_min, TheaterId) VALUES (?, ?, ?)";
            PreparedStatement statement = connection.prepareStatement(sql);
            statement.setString(1, movieName);
            statement.setInt(2, duration);
            statement.setInt(3, TheaterId);

            // Execute SQL statement
            int rowsAffected = statement.executeUpdate();
            if (rowsAffected > 0) {
                System.out.println("Movie added successfully.");
            } else {
```

```
            System.out.println("Failed to add movie.");
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
}

// Method to edit movie details
public void editMovie() {
    try {
        Scanner scanner = new Scanner(System.in);
        System.out.println("Enter movie ID to edit:");
        int movieId = scanner.nextInt();
        scanner.nextLine(); // Consume newline character

        System.out.println("Enter new movie name:");
        String movieName = scanner.nextLine();
        System.out.println("Enter new movie duration:");
        int duration = scanner.nextInt();

        // Prepare SQL statement
        String sql = "UPDATE MovieDetails SET MovieName = ?,
Duration_in_min = ? WHERE MovieId = ?";
        PreparedStatement statement = connection.prepareStatement(sql);
        statement.setString(1, movieName);
        statement.setInt(2, duration);
        statement.setInt(3, movieId);

        // Execute SQL statement
        int rowsAffected = statement.executeUpdate();
        if (rowsAffected > 0) {
            System.out.println("Movie updated successfully.");
        } else {
            System.out.println("Failed to update movie.");
        }
    } catch (SQLException e) {
```

```
            e.printStackTrace();
        }
    }

    // Method to remove a movie
    public void removeMovie() {
        try {
            Scanner scanner = new Scanner(System.in);
            System.out.println("Enter movie ID to remove:");
            int movieId = scanner.nextInt();

            // Prepare SQL statement
            String sql = "DELETE FROM MovieDetails WHERE MovieId = ?";
            PreparedStatement statement = connection.prepareStatement(sql);
            statement.setInt(1, movieId);

            // Execute SQL statement
            int rowsAffected = statement.executeUpdate();
            if (rowsAffected > 0) {
                System.out.println("Movie removed successfully.");
            } else {
                System.out.println("Failed to remove movie.");
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}
```

4.5.MAKING PAYMENT:

```
package com.raman;

import java.io.IOException;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.SQLException;
import java.util.Scanner;

public class MakePayment {

    static Scanner scanner = new Scanner(System.in);
    Countdown count = new Countdown();
//    ReduceSeats reduceSeats = new ReduceSeats();
    UserControl userControl = new UserControl();
    Connection connection = new JDBC().establishConnection();

    public static String modeOfPayment = null;

//    int seats = userControl.numberOfSeats;
//    int id = userControl.theaterId;


    public void selectPaymentMethod(){

        // Prompt user to proceed for payment

        BookingStatus bookingStatus = new BookingStatus();
        System.out.println("Do you want to proceed for payment? (yes/no)");
        String proceedChoice = scanner.nextLine();

        if (proceedChoice.equalsIgnoreCase("yes")) {
            // Present payment options
            System.out.println("Choose payment method: ");
            System.out.println("1. Cash");
            System.out.println("2. Online");
```

```
            int paymentMethod = scanner.nextInt();

            switch (paymentMethod) {
                case 1:
                    modeOfPayment="Cash";
                    new MakePayment().payByCash();
                    break;
                case 2:
                    modeOfPayment="Online";
                    new MakePayment().payOnline();
                    break;
                default:
                    System.out.println("Invalid payment method selected.\n");
//                    System.out.println("Closing the task and everything...");
                    System.out.println("Thanks For using our Service !!");
                    bookingStatus.bookingStatusCancelled();
                    System.exit(0);

            }
        } else {
            System.out.println("Payment cancelled.\n");
//            System.out.println("Closing the task and everything...");
            System.out.println("Thanks For using our Service !!");
            bookingStatus.bookingStatusCancelled();
            System.exit(0);

        }
    }


    public void payByCash() {
        System.out.println("Please Collect your ticket: ");

        addModeOfPayment(modeOfPayment);
```

```
        new BookingStatus().bookingStatusSuccessful();
        System.out.println("Thanks for being patient. Your ticket has been sent to
your provided E-mail Id.");
        new PDFgeneration();
       PDFgeneration.generatePdf();
        System.exit(0);
    }

    public void payOnline() {

        System.out.println("Please select your online payment method: ");
        System.out.println("1. Credit Card");
        System.out.println("2. Debit Card");
        System.out.println("3. Net Banking");

        int option = scanner.nextInt();

        switch (option) {
        case 1:
            modeOfPayment="Credit Card";
            payByCreditCard();
            break;
        case 2:
            modeOfPayment="Debit Card";
            payByDebitCard();
            break;
        case 3:
            modeOfPayment="Net Banking";
            payByNetBanking();
            break;
        default:
            System.out.println("Invalid online payment method selected.\n");
//            System.out.println("Closing the task and everything...");
            System.out.println("Thanks For using our Service !!");
            new BookingStatus().bookingStatusCancelled();
            System.exit(0);
```

```
}
}

public void payByCreditCard() {
    // Logic for credit card payment
    System.out.println("Enter credit card number...");
    long cardNumber = scanner.nextLong();
    addModeOfPayment(modeOfPayment);
    new BookingStatus().bookingStatusSuccessful();
    System.out.println("Payment successful!!");

    new PDFgeneration();
   PDFgeneration.generatePdf();
    System.exit(0);
}

public void payByDebitCard() {

    // Logic for debit card payment
    System.out.println("Enter debit card number...");
    long cardNumber = scanner.nextLong();
    System.out.println("Enter CVV number");
    int cvvNumber = scanner.nextInt();
    System.out.println("Enter expiry details(MM/YY)");
    scanner.nextLine();
    String expiryDetails = scanner.nextLine();

    addModeOfPayment(modeOfPayment);
    new BookingStatus().bookingStatusSuccessful();
    System.out.println("Payment successful !!");

    new PDFgeneration();
   PDFgeneration.generatePdf();

    System.exit(0);
```

```
    }

    public void payByNetBanking() {

        // Logic for net banking payment
        System.out.println("Enter net banking number");
        long netBankingNumber = scanner.nextLong();

        addModeOfPayment(modeOfPayment);
        new BookingStatus().bookingStatusSuccessful();
        System.out.println("Payment successful !!");

        new PDFgeneration();
       PDFgeneration.generatePdf();

        System.exit(0);

    }

    public void addModeOfPayment(String modeOfPayment) {

        // Assuming you have a table named TransactionalDetails with a column
named ModeOfPayment
        String updateQuery = "UPDATE TransactionalDetails SET
ModeOfPayment = ? WHERE UniqueId = ?";

        try {
            PreparedStatement preparedStatement =
connection.prepareStatement(updateQuery);
            preparedStatement.setString(1, modeOfPayment);
            preparedStatement.setString(2, UserControl.uniqueId); // Assuming
UserControl.uniqueId contains the unique identifier for the transaction
            preparedStatement.executeUpdate();
        } catch (SQLException e) {
            e.printStackTrace();
        }}}
```

4.6.PDF GENERATION:

```
package com.raman;

import java.io.*;

import com.itextpdf.text.*;
import com.itextpdf.text.pdf.*;

public class PDFgeneration {
   private static final String pdfDirectory = "c:/Invoice_PDF/";
   private static final String pdfName = "ticket1.pdf";

   static void generatePdf() {
      Document document = new Document();
      UserControl userControl = new UserControl();
      try {
         // Ensure that the directory exists
         File directory = new File(pdfDirectory);
         if (!directory.exists()) {
            directory.mkdirs(); // Create directory if it doesn't exist
         }

         // Create the PDF file
         PdfWriter pdfWriter = PdfWriter.getInstance(document, new
FileOutputStream(new File(pdfDirectory + pdfName)));
         document.open();

         document.add(new Paragraph("----------------------------------------------
-----------------------------------"));

         Font bigBoldFont = FontFactory.getFont(FontFactory.defaultEncoding,
24);
         Paragraph movieTitle = new Paragraph(userControl.movieName,
bigBoldFont);
         movieTitle.setAlignment(Element.ALIGN_LEFT);
         document.add(movieTitle);
```

```
        document.add(new Paragraph("\n"));

        Font smallFont = FontFactory.getFont(FontFactory.defaultEncoding, 14);
        Paragraph genre = new Paragraph(userControl.genre, smallFont);
        genre.setAlignment(Element.ALIGN_LEFT);
        document.add(genre);

        document.add(new Paragraph("----------------------------------------------
-----------------------------------"));
        document.add(new Paragraph("\n"));

        Paragraph timing = new Paragraph(userControl.showTiming,
bigBoldFont);
        timing.setAlignment(Element.ALIGN_LEFT);
        document.add(timing);

        document.add(new Paragraph("\n"));

        Paragraph theatre = new Paragraph(userControl.theaterName,
bigBoldFont);
        theatre.setAlignment(Element.ALIGN_LEFT);
        document.add(theatre);

        Paragraph date = new Paragraph(userControl.showDate, bigBoldFont);
        date.setAlignment(Element.ALIGN_LEFT);
        document.add(date);

        Paragraph numberOfTickets = new Paragraph("Seats Booked:
"+userControl.numberOfSeats, bigBoldFont);
        numberOfTickets.setAlignment(Element.ALIGN_LEFT);
        document.add(numberOfTickets);

        document.add(new Paragraph("\n"));
        document.add(new Paragraph("\n"));
```

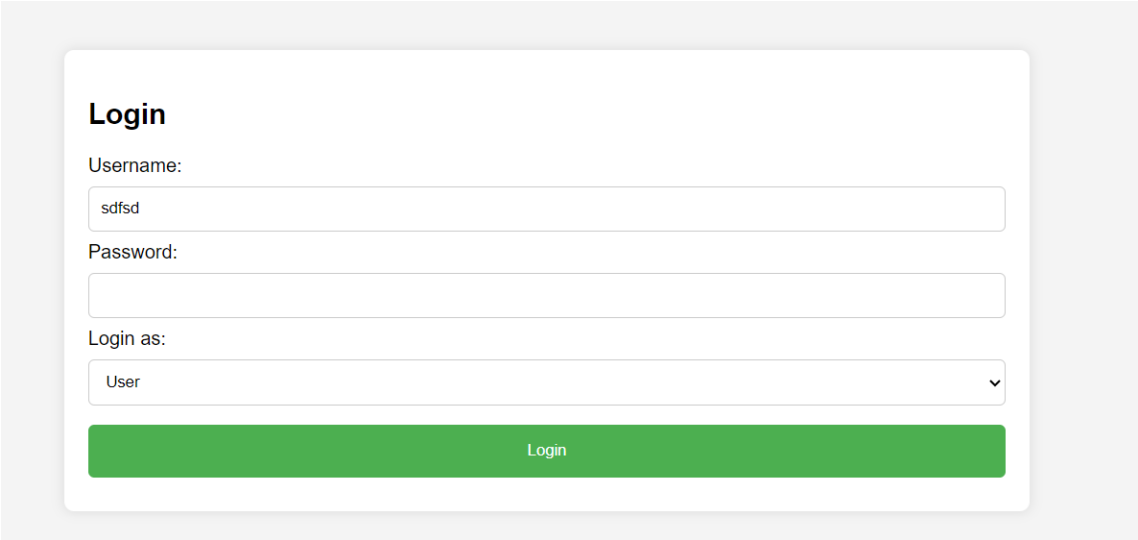
```
            Paragraph bookingId = new Paragraph("Booking ID :
WGR"+userControl.uniqueId, bigBoldFont);
            bookingId.setAlignment(Element.ALIGN_CENTER);
            document.add(bookingId);

            document.add(new Paragraph("\n"));

            Paragraph ticketPrice = new Paragraph("Tickets:
"+userControl.numberOfSeats+", Price: "+userControl.finalPrice, smallFont);
            document.add(ticketPrice);

            Paragraph cf = new Paragraph("Convenience Fees - 0.00", smallFont);
            document.add(cf);

            Paragraph ac = new Paragraph("Additional Charges - 00.00", smallFont);
            document.add(ac);

            document.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

4.7.DATABASE CONNECTIVITY(JDBC):

```
package com.raman;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

public class JDBC {

	Connection establishConnection() {
	String url = "jdbc:mysql://localhost:3306/MovieTicketBooking";
	String user = "localhost";
	String password = "PPN/2ndyr";

	try {
	   Class.forName("com.mysql.cj.jdbc.Driver");
	   Connection connection = DriverManager.getConnection(url, user,
password);
	   return connection;



	} catch (ClassNotFoundException | SQLException e) {
	   e.printStackTrace();
	   return null;
	}

    }


}
```

5. OUTPUT:

5.1.LOGIN PAGE



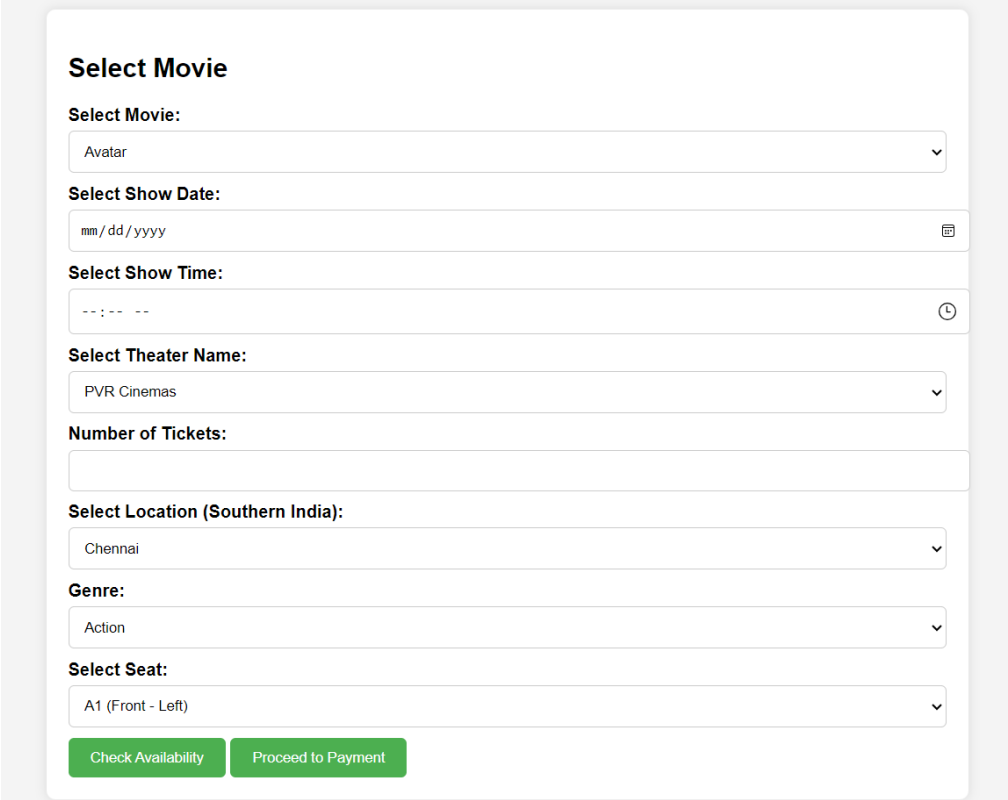
5.2.MOVIE DETAILS

Movie ID	Movie Name	Genre	Duration	Price	Theater ID	Theater Name	Location	Show Date 1	Show Da
1	Inception	Science Fiction	148	300	1	MartinCine	Coimbatore	2024-03-19	2024-03-
2	The Shawshank Redemption	Drama	142	400	2	LeoCine	Chennai	2024-03-19	2024-03-
3	The Dark Knight	Action, Crime, Drama	152	350	3	NolanCine	Coimbatore	2024-03-19	2024-03-
4	Pulp Fiction	Crime, Drama	154	500	4	ChrisCine	Bangalore	2024-03-19	2024-03-
5	The Godfather	Crime, Drama	175	450	5	LokiCine	Hyderabad	2024-03-19	2024-03-

5.3.BOOKING PROCESS:

```
PLease select the following available actions.
1. Book A ticket
2. Exit

Enter your choice: 1
Please Be patient
User Id is: 2
Press any button to proceed

Enter the name for billing purpose:
Chris
Enter movieId:
2
Enter theaterId:
2
Please Enter the date(YY-MM-DD) on which you would like to watch the movie :

Note* Kindly refer the above table to see the movie availibility
2024-03-21
Date is valid.
Select the Show Id as per the corresponding Timing

1. Morning(09:00 AM)
2. Afternoon(01:00 PM
3. Evening(05:00 PM
4. Night(09:00 PM
Please enter 1, 2, 3, or 4.
3
Enter numberOfSeats:
3
Enter Your E-mail Id:
Chris@gmail.com

Please enter a uniqueId of your choice:
3
This ID is not unique. Please try another one.
Please enter a uniqueId of your choice:
121
```

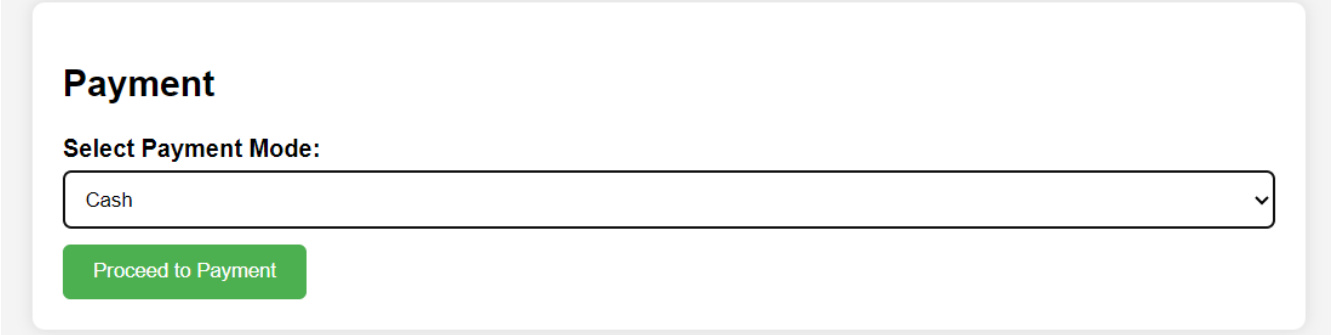
5.4.SELECTING MOVIES:



5.4.PAYMENT METHOD

```
Choose payment method:
1. Cash
2. Online
1
```

```
Please Make the payment before 30 seconds or else your transaction will be cancelled.
Note* The time will be updated after every 20 seconds.

Enter 1 to proceed to make the payment or any other number to cancel the transaction

Time remaining: 20 seconds1

Total Amount: Rs. 1200
Do you want to proceed for payment? (yes/no)
yes
```



5.5.PDF GENERATED

```
Please Collect your ticket
Thanks for being patient.
```

The Dark Knight

Action, Crime, Drama

09:00 PM

NolanCine

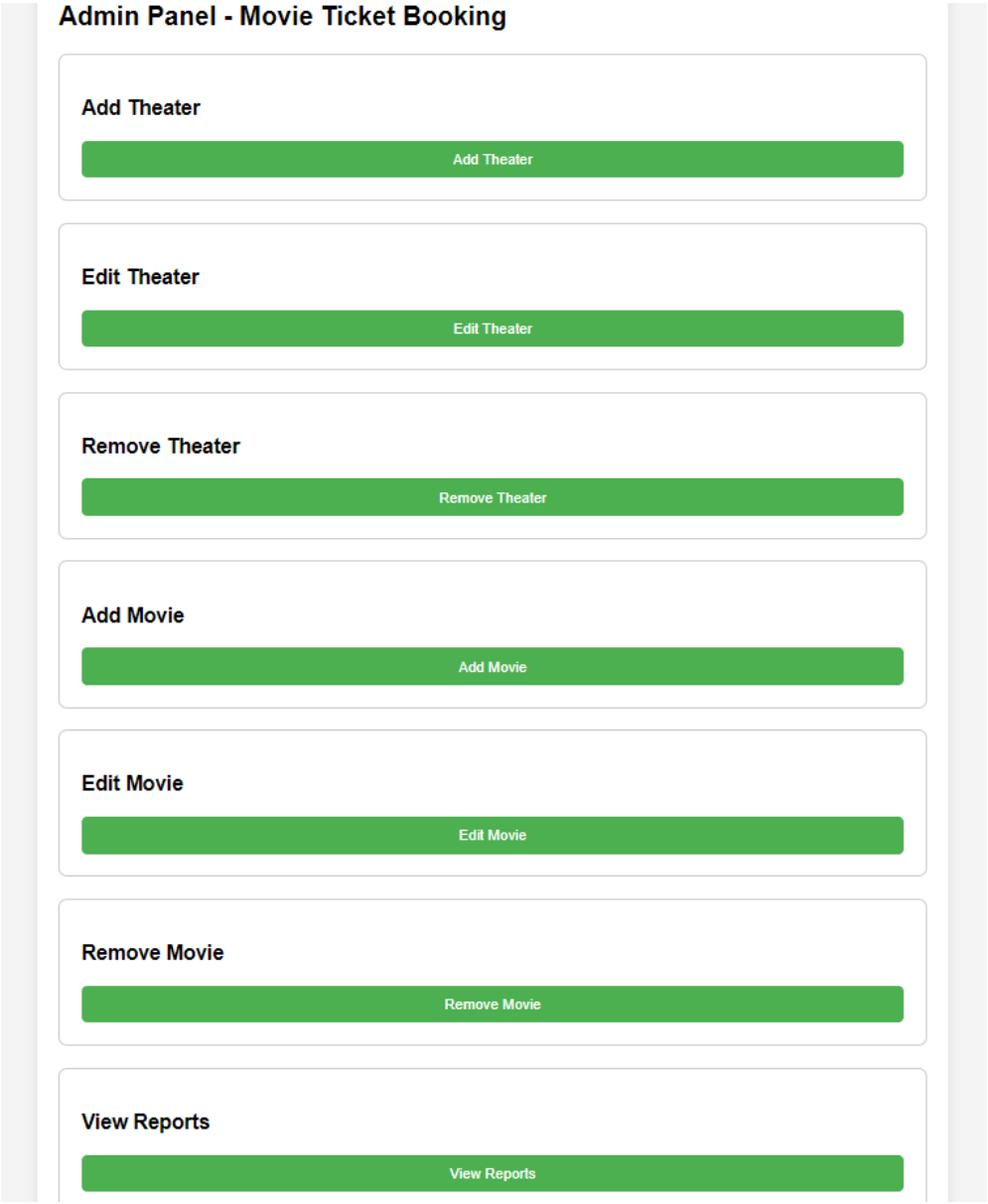
24-03-19

Seats Booked: 3

Booking ID : WGRN1thq

Tickets: 3, Price: 930
Convenience Fees - 0.00
Additional Charges - 00.00

5.6.ADMIN CONTROL



6. CONCLUSION

The **Movie Ticket Booking System** enhances the cinema experience by integrating ticket booking, movie management, and user authentication into a unified and efficient platform. It reduces manual workload, minimizes errors, and provides a seamless experience for both administrators and customers. By automating essential processes like seat reservation, schedule management, and data handling, the system ensures better decision-making and improves service quality.

This application not only streamlines operations for cinema management but also offers a scalable foundation for future enhancements, such as real-time payment integration and mobile app development, ensuring its relevance and adaptability in the evolving entertainment industry.

7. REFERENCES

- Oracle Java Documentation:
  https://docs.oracle.com/javase/
- MySQL Reference Manual:
  https://dev.mysql.com/doc/
- TutorialsPoint JSP and Servlets:
  https://www.tutorialspoint.com/
- GitHub resources and repositories.