**CINEMA BOOKING SYSTEM**

**A MINI PROJECT REPORT**

**Submitted by**

**ARAVIND A S (953622104006)**

**BACHELOR OF ENGINEERING**

**IN**

**COMPUTER SCIENCE AND ENGINEERING**



# RAMCO INSTITUTE OF TECHNOLOGY, RAJAPALAYAM

# JUNE 2023

**CS3481 Database Management Systems Laboratory**

# BONAFIDE CERTIFICATE

Certified that this project report titled **“Cinema Booking System”** is the bonafide work of **Aravind A S (953622104006)** who carried out the project work under my supervision.

|  |  |
| --- | --- |
| **SIGNATURE**  **Dr.K.Vijayalakshmi, M.E, Ph.D. HEAD OF THE DEPARTMENT**  Professor & Head  Department of Computer Science and Engineering  Ramco Institute of Technology  North Venganallur Village  Rajapalayam-626117.  INTERNAL EXAMINER | **SIGNATURE**  **Mrs.K.Jeyakarthika, M.Tech.**  **SUPERVISOR**  Assistant Professor  Department of Computer Science and Engineering  Ramco Institute of Technology  North Venganallur Village Rajapalayam-626117.  EXTERNAL EXAMINER |

# TABLE OF CONTENTS

|  |  |  |
| --- | --- | --- |
| S.NO | CONTENT | PAGE NO |
| 1. | INTRODUCTION | 5 |
| 2. | PROJECT DESCRIPTION | 5 |
| 3. | MODULES | 6 |
| 4. | TECHNOLOGY STACK USEDMYSQL CONNECTIVITYER DIAGRAM | 6  6  10 |
| 5. | CODING | 15 |
| 6. | OUTPUT SCREENSHOT | 39 |
| 7. | CONCLUSION | 44 |

# 

# ABSTRACT

The Cinema Booking System is a comprehensive application designed to streamline the process of reserving movie tickets, enhancing the overall user experience for both cinema administrators and customers. Utilizing JavaFX for the front-end interface and MySQL Workbench for the back-end database management, the system ensures a seamless and efficient booking process. The core functionalities of the system include user authentication for administrators and customers, movie and theatre management, real-time seat availability tracking, booking reservations, and customer reviews.

The database schema is meticulously structured with the following tables: admins, customers, movies, theatres, bookings, and reviews. Each table serves a specific purpose, such as storing user credentials, movie details, theatre information, booking records, and customer feedback. The integration of these tables facilitates efficient data handling and retrieval, ensuring the system's robustness and reliability.

This project aims to provide a user-friendly platform that caters to the needs of modern cinema-goers and administrators, simplifying the complexities involved in movie ticket bookings. The implementation of the Cinema Booking System highlights the synergy between a well-designed front-end interface and a robust back-end database, demonstrating the potential of combining JavaFX and MySQL in creating effective and scalable applications.

# INTRODUCTION:

In today's digital age, the demand for efficient and user-friendly cinema booking systems has surged, reflecting the broader trend of digital transformation across industries. Traditional methods of booking movie tickets, such as in-person purchases or phone reservations, have become less practical and desirable. Modern consumers seek convenience, speed, and reliability in their transactions, necessitating the development of advanced systems that cater to these expectations. The Cinema Booking System project addresses this need by providing a comprehensive solution that facilitates seamless movie ticket reservations, enhancing the overall user experience for both cinema administrators and customers.

The Cinema Booking System leverages JavaFX for its front-end interface and MySQL Workbench for back-end database management. This combination ensures a visually appealing and intuitive user experience while maintaining robust data management and security. Key functionalities of the system include secure user authentication, efficient management of movies and theatres, real-time seat availability tracking, and an easy-to-use booking process. Additionally, the system allows customers to provide reviews, contributing to a more engaging and interactive platform. By integrating these features, the Cinema Booking System aims to streamline the booking process, ensuring a hassle-free and efficient experience for all users involved.

# PROJECT DESCRIPTION:

The Cinema Booking System is a comprehensive application designed to streamline the movie ticket reservation process for both cinema administrators and customers. Developed using JavaFX for the front-end interface and MySQL Workbench for the back-end database management, the system ensures a seamless and efficient user experience. Key features include secure user authentication, allowing both administrators and customers to register and log in securely, and real-time seat availability tracking to prevent double bookings. Administrators have the ability to manage movie and theatre details, including updating and deleting entries as needed, while customers can easily select and book their desired seats for a chosen movie and theatre. Additionally, the system enables customers to leave reviews for movies, fostering a sense of community and engagement.

The database schema is designed to facilitate efficient data handling and retrieval, comprising tables for storing administrator and customer credentials, movie details, theatre information, booking records, and customer reviews. This robust architecture ensures data integrity and security, providing a reliable platform for managing cinema operations. The integration of JavaFX and MySQL provides a powerful combination of a rich, interactive front-end and a reliable, scalable back-end, demonstrating the potential for creating effective and user-friendly applications. Overall, the Cinema Booking System offers a modern solution to the complexities of movie ticket reservations, enhancing the overall user experience and simplifying cinema management.

# 3. MODULES:

**Database Connection Module**

Handles SQL database connections using JDBC, ensuring secure and efficient operations. Provides methods to establish and close connections.

**SQL Query Module**

Contains SQL queries to retrieve student registration numbers and update grades, managing essential database transactions.

**Main Application Module**

Integrates GUI and database operations. Manages user interactions and executes SQL queries.

**User Interface Module**

Provides a JavaFX-based GUI, displaying student registration numbers, and includes buttons for adding and updating students.

**Event Handling Module**

Manages user interactions with the GUI, updating the interface based on input and database operations.

# TECHNOLOGY STACK:

Programming Language: Javafx

Serving as the backbone for backend logic, integrating seamlessly with the GUI and ensuring optimal system performance.

**Packages Used:**

Java SQL Imports:

**java.sql.Connection:** Represents a connection to a relational database.

**java.sql.PreparedStatement**: Used for precompiled SQL statements.

**java.sql.ResultSet:** Represents the result set of a database query.

**java.sql.ResultSetMetaData:** Provides information about the columns in a ResultSet.

**java.sql.SQLException:** Handles exceptions related to database operations.

**java.sql.Statement:** Used for executing SQL statements

**Java Util Imports:**

**java.util.ArrayList**: A dynamic array for storing elements.

**java.util.Optional:** Represents an optional value, useful for cases where a value might be absent.

**java.util.logging.Level:** Defines logging levels for Logger objects.

**java.util.logging.Logger:** Used for logging messages.

**SQL DATABASE INTEGRATION:**

Database Connectivity Using JDBC

Under Dependency folder, Check whether the Connector/J file is added properly(JAR file). Then, check the Services Tab for the database that we are going to work with. It can be found by clicking other connection option.

Downloading SQL Server Management Studio (SSMS)

**Visit Microsoft's Website:** Go to the official [Microsoft SQL Server Management Studio download page](https://docs.microsoft.com/en-us/sql/ssms/download-sql-server-management-studio-ssms).

**Choose the Latest Version:** Select the latest version of SSMS available for download. Microsoft frequently updates SSMS with new features and bug fixes.

**Select the Installation Package:** Depending on your system architecture (32-bit or 64-bit), choose the appropriate installation package.

**Download and Install:** Once downloaded, run the installer and follow the on-screen instructions to complete the installation process.

Dependencies in a Java Project

**JDBC Driver:** Add the JDBC driver JAR file (sqljdbc.jar or similar) to your project's build path or include it as a dependency in your build tool (e.g., Maven, Gradle).

**JavaFX (if using JavaFX):** Include JavaFX libraries in your project’s build path or use a build tool to manage dependencies (e.g., Maven, Gradle).

Database Schema Creation

This example demonstrates how to create a simple login table (Login) in your SQL Server database using JDBC and SQL statements.

Additional Considerations

**Security:** Avoid hardcoding sensitive information like usernames and passwords directly in your code. Consider using environment variables or a secure configuration management approach.

**Error Handling:** Implement more robust error handling and logging mechanisms as per your application's requirements.

Dependencies in a Java Project

**JDBC Driver:** Add the JDBC driver JAR file (sqljdbc.jar or similar) to your project's build path or include it as a dependency in your build tool (e.g., Maven, Gradle).

**JavaFX (if using JavaFX):** Include JavaFX libraries in your project’s build path or use a build tool to manage dependencies (e.g., Maven, Gradle).

**VIEWING TABLE CONTENT**

To view table content:

1. **ADMINS TABLE:**

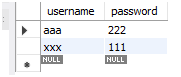
**admins Table:** Stores administrator credentials, including username and password. Ensures data integrity with a primary key on the username.

CREATE TABLE IF NOT EXISTS admins (

username VARCHAR(255) PRIMARY KEY,

password VARCHAR(255)

);



1. **CUSTOMERS TABLE:**

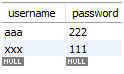
**customers Table:** Stores customer credentials, including username and password. Ensures data integrity with a primary key on the username.

CREATE TABLE IF NOT EXISTS customers (

username VARCHAR(255) PRIMARY KEY,

password VARCHAR(255)

);



1. **MOVIES TABLE:**

**movies Table:** Stores movie details, including the name of the movie. Ensures data integrity with a primary key on the name.

CREATE TABLE IF NOT EXISTS movies (

name VARCHAR(255) PRIMARY KEY

);



1. **THEATRES TABLE :**

**theatres Table:** Stores theatre information, including a unique ID, movie name, theatre name, and available seats. Ensures data integrity with a primary key on the ID and a foreign key constraint referencing the movie name in the movies table.

CREATE TABLE IF NOT EXISTS theatres (

id INT AUTO\_INCREMENT PRIMARY KEY,

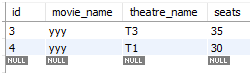
movie\_name VARCHAR(255),

theatre\_name VARCHAR(255),

seats INT,

FOREIGN KEY (movie\_name) REFERENCES movies(name)

);



1. **BOOKING TABLE:**

**booking Table:** Stores booking records, including movie name, theatre name, customer name, and number of seats booked. Ensures data integrity with foreign key constraints referencing the movies and customers tables.

CREATE TABLE IF NOT EXISTS bookings (

movie\_name VARCHAR(255),

theatre\_name VARCHAR(255),

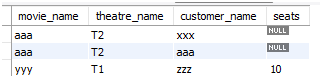
customer\_name VARCHAR(255),

seats INT,

FOREIGN KEY (movie\_name) REFERENCES movies(name),

FOREIGN KEY (customer\_name) REFERENCES customers(username)

);



1. **REVIEWS TABLE:**

**review Table:** Stores customer reviews for movies, including the customer name, movie name, and review text. Ensures data integrity with foreign key constraints referencing the customers and movies tables.

CREATE TABLE IF NOT EXISTS reviews (

customer\_name VARCHAR(255),

movie\_name VARCHAR(255),

review VARCHAR(255),

FOREIGN KEY (customer\_name) REFERENCES customers(username),

FOREIGN KEY (movie\_name) REFERENCES movies(name)

);



1. **NORMALIZATION**

First Normal Form (1NF)

* Each column in a table contains only atomic (indivisible) values, and no column contains multiple values.
* admins, customers, movies, theatres, bookings, and reviews tables each have primary keys (username, username, name, id, and a composite key for bookings and reviews respectively).

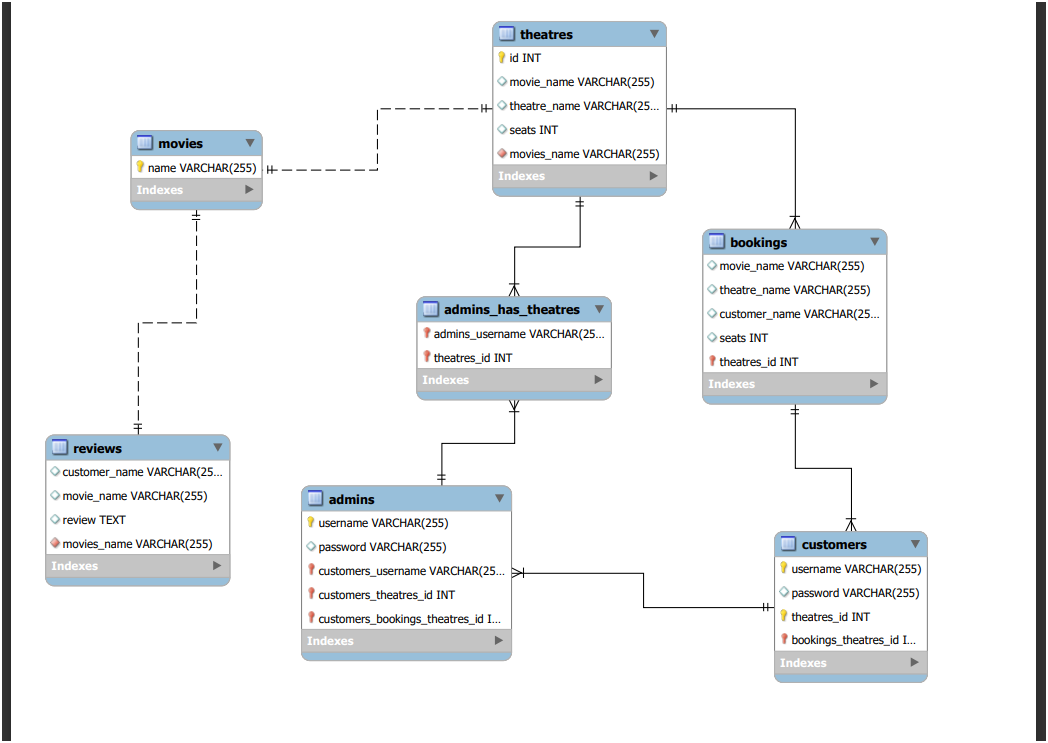
Second Normal Form (2NF)

* A table is in 2NF if it is in 1NF and all non-key attributes are fully dependent on the primary key, avoiding partial dependencies.
* theatres table: movie\_name (foreign key) is dependent on the primary key id. This means movie\_name depends on id.
* If there are multiple attributes related to a non-primary key attribute (e.g., movie\_name in the theatres table), these should be examined to ensure they are fully dependent on the primary key.

Third Normal Form

* A table is in 3NF if it is in 2NF and there are no transitive dependencies; that is, no non-key attribute is dependent on another non-key attribute.
* Ensure no transitive dependencies where non-key attributes depend on other non-key attributes.
* admins table: No transitive dependencies observed.
* customers table: No transitive dependencies observed.
* movies table: No transitive dependencies observed.
* theatres table: No transitive dependencies observed.
* bookings table: No transitive dependencies observed.
* reviews table: No transitive dependencies observed.

1. **ER DIAGRAM**



**Table: admins**

**Columns:**

1. **username**: This column serves as the primary key of the table and uniquely identifies each admin. It is a string field.
2. **password**: This column stores the encrypted password of the admin. It is a string field.

**Key Features:**

1. **Unique username**: Each admin in the table has a unique username assigned to it, ensuring no two admins have the same identifier, enabling efficient identification and retrieval.
2. **Password Security**: The password column stores encrypted passwords, ensuring secure authentication.

**Functionality:**

1. **Admin Creation**: The table allows for the creation of new admins by inserting new rows. Users can specify a unique username and assign a secure password.
2. **Admin Modification**: Users can update the password of existing admins if any changes or updates are required.
3. **Admin Deletion**: If an admin is no longer needed, users can delete the corresponding row from the table. This removes the admin from the system’s admin list.
4. **Admin Retrieval**: The table facilitates the retrieval of admin information. Users can query the table to fetch a list of all admins or search for specific admins based on their username.

**Table: customers**

**Columns:**

1. **username**: This column serves as the primary key of the table and uniquely identifies each customer. It is a string field.
2. **password**: This column stores the encrypted password of the customer. It is a string field.

**Key Features:**

1. **Unique username**: Each customer in the table has a unique username assigned to it, ensuring no two customers have the same identifier, enabling efficient identification and retrieval.
2. **Password Security**: The password column stores encrypted passwords, ensuring secure authentication.

**Functionality:**

1. **Customer Creation**: The table allows for the creation of new customers by inserting new rows. Users can specify a unique username and assign a secure password.
2. **Customer Modification**: Users can update the password of existing customers if any changes or updates are required.
3. **Customer Deletion**: If a customer is no longer needed, users can delete the corresponding row from the table. This removes the customer from the system’s customer list.
4. **Customer Retrieval**: The table facilitates the retrieval of customer information. Users can query the table to fetch a list of all customers or search for specific customers based on their username.

**Table: movies**

**Columns:**

1. **name**: This column serves as the primary key of the table and uniquely identifies each movie. It is a string field.

**Key Features:**

1. **Unique name**: Each movie in the table has a unique name assigned to it, ensuring no two movies have the same identifier, enabling efficient identification and retrieval.

**Functionality:**

1. **Movie Creation**: The table allows for the creation of new movies by inserting new rows. Users can specify a unique name.
2. **Movie Modification**: Users can update the details of existing movies if any changes or updates are required.
3. **Movie Deletion**: If a movie is no longer needed, users can delete the corresponding row from the table. This removes the movie from the system’s movie list.
4. **Movie Retrieval**: The table facilitates the retrieval of movie information. Users can query the table to fetch a list of all movies or search for specific movies based on their name.

**Table: theatres**

**Columns:**

1. **id**: This column serves as the primary key of the table and uniquely identifies each theatre. It is an auto-incrementing integer.
2. **movie\_name**: This column stores the name of the movie being shown in the theatre. It is a string field and a foreign key referencing movies(name).
3. **theatre\_name**: This column stores the name of the theatre. It is a string field.
4. **seats**: This column indicates the seating capacity of the theatre. It is an integer field.

**Key Features:**

1. **Unique ID**: Each theatre in the table has a unique id assigned to it, ensuring no two theatres have the same identifier, enabling efficient identification and retrieval.
2. **Movie Association**: The movie\_name column links each theatre to a specific movie, facilitating the management of movie screenings.

**Functionality:**

1. **Theatre Creation**: The table allows for the creation of new theatres by inserting new rows. Users can specify a movie\_name, assign a theatre\_name, and provide details like seats.
2. **Theatre Modification**: Users can update the movie\_name, theatre\_name, or seats of existing theatres if any changes or updates are required.
3. **Theatre Deletion**: If a theatre is no longer needed, users can delete the corresponding row from the table. This removes the theatre from the system’s theatre list.
4. **Theatre Retrieval**: The table facilitates the retrieval of theatre information. Users can query the table to fetch a list of all theatres or search for specific theatres based on their id or movie\_name.

**Table: bookings**

**Columns:**

1. **movie\_name**: This column stores the name of the booked movie. It is a string field and a foreign key referencing movies(name).
2. **theatre\_name**: This column stores the name of the theatre where the booking is made. It is a string field.
3. **customer\_name**: This column stores the username of the customer who made the booking. It is a string field and a foreign key referencing customers(username).
4. **seats**: This column indicates the number of seats booked by the customer. It is an integer field.

**Key Features:**

1. **Composite Key**: The combination of movie\_name, theatre\_name, and customer\_name serves as a composite key, ensuring each booking is uniquely identified.
2. **Customer Association**: The customer\_name column links each booking to a specific customer, facilitating the management of customer bookings.

**Functionality:**

1. **Booking Creation**: The table allows for the creation of new bookings by inserting new rows. Users can specify movie\_name, theatre\_name, customer\_name, and seats.
2. **Booking Modification**: Users can update the details of existing bookings if any changes or updates are required.
3. **Booking Deletion**: If a booking is no longer needed, users can delete the corresponding row from the table. This removes the booking from the system’s booking list.
4. **Booking Retrieval**: The table facilitates the retrieval of booking information. Users can query the table to fetch a list of all bookings or search for specific bookings based on movie\_name, theatre\_name, or customer\_name.

**Table: reviews**

**Columns:**

1. **customer\_name**: This column stores the username of the customer who provided the review. It is a string field and a foreign key referencing customers(username).
2. **movie\_name**: This column stores the name of the movie being reviewed. It is a string field and a foreign key referencing movies(name).
3. **review**: This column stores the text of the review provided by the customer. It is a string field.

**Key Features:**

1. **Composite Key**: The combination of customer\_name and movie\_name serves as a composite key, ensuring each review is uniquely identified.
2. **Feedback Mechanism**: The review column allows customers to provide feedback on movies, enhancing user engagement.

**Functionality:**

1. **Review Creation**: The table allows for the creation of new reviews by inserting new rows. Users can specify customer\_name, movie\_name, and review.
2. **Review Modification**: Users can update the review of existing reviews if any changes or updates are required.
3. **Review Deletion**: If a review is no longer needed, users can delete the corresponding row from the table. This removes the review from the system’s review list.
4. **Review Retrieval**: The table facilitates the retrieval of review information. Users can query the table to fetch a list of all reviews or search for specific reviews based on customer\_name or movie\_name.

**DATABASE CONNECTION**

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.SQLException;

private void initializeDatabase() {

try {

connection = DriverManager.getConnection("jdbc:mysql://localhost:3306/cinema?zeroDateTimeBehavior=CONVERT\_TO\_NULL", "root", "50+50=100" );

createTables();

} catch (SQLException e) {

showErrorAlert("Error initializing database: " + e.getMessage());

System.exit(1);

}

}

1. CODING

package com.mycompany.dbms;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.PreparedStatement;

import java.sql.ResultSet;

import java.sql.SQLException;

import java.util.ArrayList;

import java.util.List;

import javafx.application.Application;

import javafx.beans.property.SimpleIntegerProperty;

import javafx.beans.property.SimpleStringProperty;

import javafx.collections.FXCollections;

import javafx.collections.ObservableList;

import javafx.geometry.Insets;

import javafx.scene.Scene;

import javafx.scene.control.\*;

import javafx.scene.control.cell.PropertyValueFactory;

import javafx.scene.layout.GridPane;

import javafx.scene.layout.VBox;

import javafx.scene.text.Font;

import javafx.scene.text.FontWeight;

import javafx.scene.text.Text;

import javafx.stage.Stage;

public class App extends Application {

private Connection connection;

private String currentUser;

private String bookingMovie;

private String bookingTheatre;

private Integer bookingSeats;

@Override

public void start(Stage primaryStage) {

initializeDatabase();

primaryStage.setTitle("Cinema Booking System");

GridPane grid = createMainGridPane();

Text scenetitle = new Text("Welcome to Cinema Booking System");

scenetitle.setFont(Font.font("Tahoma", FontWeight.NORMAL, 20));

grid.add(scenetitle, 0, 0, 2, 1);

addButton(grid, "Admin", 0, 1, e -> openLoginWindow(primaryStage, "Admin"));

addButton(grid, "Customer", 1, 1, e -> openCustomerWindow(primaryStage));

addButton(grid, "Exit", 1, 2, e -> primaryStage.close());

Scene scene = new Scene(grid, 400, 300);

primaryStage.setScene(scene);

primaryStage.show();

}

private void initializeDatabase() {

try {

connection = DriverManager.getConnection(

"jdbc:mysql://localhost:3306/cinema?zeroDateTimeBehavior=CONVERT\_TO\_NULL",

"root",

"50+50=100"

);

createTables();

} catch (SQLException e) {

showErrorAlert("Error initializing database: " + e.getMessage());

System.exit(1);

}

}

private GridPane createMainGridPane() {

GridPane grid = new GridPane();

grid.setHgap(10);

grid.setVgap(10);

grid.setPadding(new Insets(20, 20, 20, 20));

return grid;

}

private void addButton(GridPane grid, String text, int col, int row, javafx.event.EventHandler<javafx.event.ActionEvent> event) {

Button button = new Button(text);

button.setOnAction(event);

grid.add(button, col, row);

}

private void createTables() throws SQLException {

executeUpdate("CREATE TABLE IF NOT EXISTS admins (username VARCHAR(255) PRIMARY KEY, password VARCHAR(255))");

executeUpdate("CREATE TABLE IF NOT EXISTS customers (username VARCHAR(255) PRIMARY KEY, password VARCHAR(255))");

executeUpdate("CREATE TABLE IF NOT EXISTS movies (name VARCHAR(255) PRIMARY KEY)");

executeUpdate("CREATE TABLE IF NOT EXISTS theatres (id INT AUTO\_INCREMENT PRIMARY KEY, movie\_name VARCHAR(255), theatre\_name VARCHAR(255), seats INT)");

executeUpdate("CREATE TABLE IF NOT EXISTS bookings (movie\_name VARCHAR(255), theatre\_name VARCHAR(255), customer\_name VARCHAR(255), seats INT)");

executeUpdate("CREATE TABLE IF NOT EXISTS reviews (customer\_name VARCHAR(255), movie\_name VARCHAR(255), review TEXT)");

}

private void executeUpdate(String query) throws SQLException {

try (PreparedStatement statement = connection.prepareStatement(query)) {

statement.executeUpdate();

}

}

private void openLoginWindow(Stage primaryStage, String userType) {

Stage loginStage = createLoginStage(userType + " Login", userType + " Login", (username, password) -> authenticateUser(username, password, userType.equals("Admin") ? "admins" : "customers"), primaryStage, userType);

loginStage.show();

}

private Stage createLoginStage(String title, String header, LoginHandler loginHandler, Stage primaryStage, String userType) {

Stage loginStage = new Stage();

loginStage.setTitle(title);

GridPane loginGrid = createMainGridPane();

addTitle(loginGrid, header);

Label usernameLabel = new Label("Username:");

loginGrid.add(usernameLabel, 0, 1);

TextField usernameField = new TextField();

loginGrid.add(usernameField, 1, 1);

Label passwordLabel = new Label("Password:");

loginGrid.add(passwordLabel, 0, 2);

PasswordField passwordField = new PasswordField();

loginGrid.add(passwordField, 1, 2);

addButton(loginGrid, "Login", 1, 3, e -> {

try {

String username = usernameField.getText();

String password = passwordField.getText();

if (loginHandler.authenticate(username, password)) {

currentUser = username;

loginStage.close();

if (bookingMovie != null && bookingTheatre != null) {

makeBooking(currentUser, bookingMovie, bookingTheatre, bookingSeats);

showAlert(Alert.AlertType.INFORMATION, "Success", "Booking made successfully");

bookingMovie = null;

bookingTheatre = null;

bookingSeats = null;

} else if (userType.equals("Admin")) {

openAdminWindow(primaryStage);

}

} else {

showAlert(Alert.AlertType.ERROR, "Error", "Invalid username or password");

}

} catch (SQLException ex) {

showErrorAlert("Error during login: " + ex.getMessage());

}

});

if (!userType.equals("Admin")) {

addButton(loginGrid, "Register", 0, 3, e -> openRegisterWindow(loginStage, title, userType));

}

Scene loginScene = new Scene(loginGrid, 400, 300);

loginStage.setScene(loginScene);

return loginStage;

}

private void openRegisterWindow(Stage loginStage, String title, String userType) {

Stage registerStage = new Stage();

registerStage.setTitle(title + " Register");

GridPane registerGrid = createMainGridPane();

addTitle(registerGrid, title + " Register");

Label usernameLabel = new Label("Username:");

registerGrid.add(usernameLabel, 0, 1);

TextField usernameField = new TextField();

registerGrid.add(usernameField, 1, 1);

Label passwordLabel = new Label("Password:");

registerGrid.add(passwordLabel, 0, 2);

PasswordField passwordField = new PasswordField();

registerGrid.add(passwordField, 1, 2);

addButton(registerGrid, "Register", 1, 3, e -> {

try {

String username = usernameField.getText();

String password = passwordField.getText();

if (userType.equals("Admin")) {

registerUser(username, password, "admins");

} else {

registerUser(username, password, "customers");

}

registerStage.close();

loginStage.show();

showAlert(Alert.AlertType.INFORMATION, "Success", userType + " registered successfully");

} catch (SQLException ex) {

showErrorAlert("Error during registration: " + ex.getMessage());

}

});

Scene registerScene = new Scene(registerGrid, 400, 300);

registerStage.setScene(registerScene);

registerStage.show();

}

private void openAdminWindow(Stage primaryStage) {

Stage adminStage = createManagementStage("Admin Panel", primaryStage, "Manage Customers", "Manage Movies");

adminStage.show();

}

private Stage createManagementStage(String title, Stage primaryStage, String button1Text, String button2Text) {

Stage stage = new Stage();

stage.setTitle(title);

VBox vbox = new VBox();

vbox.setSpacing(10);

vbox.setPadding(new Insets(20, 20, 20, 20));

Text titleText = new Text(title);

titleText.setFont(Font.font("Tahoma", FontWeight.NORMAL, 20));

vbox.getChildren().add(titleText);

Button button1 = new Button(button1Text);

button1.setMinWidth(200);

button1.setOnAction(e -> openWindow(button1Text));

vbox.getChildren().add(button1);

Button button2 = new Button(button2Text);

button2.setMinWidth(200);

button2.setOnAction(e -> openWindow(button2Text));

vbox.getChildren().add(button2);

Button logoutButton = new Button("Logout");

logoutButton.setMinWidth(200);

logoutButton.setOnAction(e -> {

currentUser = null;

stage.close();

primaryStage.show();

});

vbox.getChildren().add(logoutButton);

Scene scene = new Scene(vbox, 400, 300);

stage.setScene(scene);

return stage;

}

private void openCustomerWindow(Stage primaryStage) {

Stage customerStage = new Stage();

customerStage.setTitle("Customer Panel");

VBox vbox = new VBox();

vbox.setSpacing(10);

vbox.setPadding(new Insets(20, 20, 20, 20));

Text title = new Text("Customer Panel");

title.setFont(Font.font("Tahoma", FontWeight.NORMAL, 20));

vbox.getChildren().add(title);

Button viewMoviesButton = new Button("View Movies");

viewMoviesButton.setMinWidth(200);

viewMoviesButton.setOnAction(e -> openViewMoviesWindow());

vbox.getChildren().add(viewMoviesButton);

Button makeBookingButton = new Button("Make Booking");

makeBookingButton.setMinWidth(200);

makeBookingButton.setOnAction(e -> openMakeBookingWindow());

vbox.getChildren().add(makeBookingButton);

Button reviewButton = new Button("Review");

reviewButton.setMinWidth(200);

reviewButton.setOnAction(e -> openReviewWindow());

vbox.getChildren().add(reviewButton);

Button backButton = new Button("Back");

backButton.setMinWidth(200);

backButton.setOnAction(e -> customerStage.close());

vbox.getChildren().add(backButton);

Scene scene = new Scene(vbox, 400, 300);

customerStage.setScene(scene);

customerStage.show();

}

private void openViewMoviesWindow() {

Stage viewMoviesStage = new Stage();

viewMoviesStage.setTitle("View Movies");

VBox vbox = new VBox();

vbox.setSpacing(10);

vbox.setPadding(new Insets(20, 20, 20, 20));

Text title = new Text("Movies");

title.setFont(Font.font("Tahoma", FontWeight.NORMAL, 20));

vbox.getChildren().add(title);

TableView<MovieTheatre> tableView = createMovieTheatreTableView();

vbox.getChildren().add(tableView);

updateTableView(tableView);

Button backButton = new Button("Back");

backButton.setOnAction(e -> viewMoviesStage.close());

vbox.getChildren().add(backButton);

Scene viewMoviesScene = new Scene(vbox, 400, 500);

viewMoviesStage.setScene(viewMoviesScene);

viewMoviesStage.show();

}

private void openMakeBookingWindow() {

Stage makeBookingStage = new Stage();

makeBookingStage.setTitle("Make Booking");

VBox vbox = new VBox();

vbox.setSpacing(10);

vbox.setPadding(new Insets(20, 20, 20, 20));

Text title = new Text("Make Booking");

title.setFont(Font.font("Tahoma", FontWeight.NORMAL, 20));

vbox.getChildren().add(title);

Label customerNameLabel = new Label("Customer Name:");

vbox.getChildren().add(customerNameLabel);

TextField customerNameField = new TextField();

customerNameField.setText(currentUser);

customerNameField.setDisable(true);

vbox.getChildren().add(customerNameField);

Label movieNameLabel = new Label("Movie Name:");

vbox.getChildren().add(movieNameLabel);

ComboBox<String> movieComboBox = new ComboBox<>();

try {

List<String> movieNames = getAllMovieNames();

movieComboBox.getItems().addAll(movieNames);

} catch (SQLException e) {

showErrorAlert("Error retrieving movie names: " + e.getMessage());

}

vbox.getChildren().add(movieComboBox);

Label theatreNameLabel = new Label("Theatre Name:");

vbox.getChildren().add(theatreNameLabel);

ComboBox<String> theatreComboBox = new ComboBox<>();

movieComboBox.setOnAction(e -> {

theatreComboBox.getItems().clear();

try {

List<String> theatreNames = getTheatresByMovie(movieComboBox.getValue());

theatreComboBox.getItems().addAll(theatreNames);

} catch (SQLException ex) {

showErrorAlert("Error retrieving theatres: " + ex.getMessage());

}

});

vbox.getChildren().add(theatreComboBox);

Label seatsLabel = new Label("Seats:");

vbox.getChildren().add(seatsLabel);

ComboBox<Integer> seatsComboBox = new ComboBox<>();

theatreComboBox.setOnAction(e -> {

seatsComboBox.getItems().clear();

try {

int seats = getSeatsByTheatre(theatreComboBox.getValue());

for (int i = 1; i <= seats; i++) {

seatsComboBox.getItems().add(i);

}

} catch (SQLException ex) {

showErrorAlert("Error retrieving seats: " + ex.getMessage());

}

});

vbox.getChildren().add(seatsComboBox);

Button bookButton = new Button("Book");

bookButton.setOnAction(e -> {

try {

bookingMovie = movieComboBox.getValue();

bookingTheatre = theatreComboBox.getValue();

bookingSeats = seatsComboBox.getValue();

if (bookingMovie == null || bookingTheatre == null || bookingSeats == null) {

showAlert(Alert.AlertType.ERROR, "Error", "Please select movie, theatre, and seats");

return;

}

if (currentUser == null) {

openLoginWindow(new Stage(), "Customer");

} else {

makeBooking(currentUser, bookingMovie, bookingTheatre, bookingSeats);

showAlert(Alert.AlertType.INFORMATION, "Success", "Booking made successfully");

makeBookingStage.close();

}

} catch (NumberFormatException ex) {

showAlert(Alert.AlertType.ERROR, "Error", "Invalid number of seats");

} catch (SQLException ex) {

showErrorAlert("Error making booking: " + ex.getMessage());

}

});

vbox.getChildren().add(bookButton);

Scene makeBookingScene = new Scene(vbox, 400, 500);

makeBookingStage.setScene(makeBookingScene);

makeBookingStage.show();

}

private void openReviewWindow() {

Stage reviewStage = new Stage();

reviewStage.setTitle("Review");

VBox vbox = new VBox();

vbox.setSpacing(10);

vbox.setPadding(new Insets(20, 20, 20, 20));

Text title = new Text("Review Movies and Theatres");

title.setFont(Font.font("Tahoma", FontWeight.NORMAL, 20));

vbox.getChildren().add(title);

TableView<Review> tableView = createReviewTableView();

vbox.getChildren().add(tableView);

updateReviewTableView(tableView);

Button addReviewButton = new Button("Add Review");

addReviewButton.setOnAction(e -> openAddReviewWindow());

vbox.getChildren().add(addReviewButton);

Button backButton = new Button("Back");

backButton.setOnAction(e -> reviewStage.close());

vbox.getChildren().add(backButton);

Scene reviewScene = new Scene(vbox, 400, 500);

reviewStage.setScene(reviewScene);

reviewStage.show();

}

private void openAddReviewWindow() {

Stage addReviewStage = new Stage();

addReviewStage.setTitle("Add Review");

VBox vbox = new VBox();

vbox.setSpacing(10);

vbox.setPadding(new Insets(20, 20, 20, 20));

Text title = new Text("Add Review");

title.setFont(Font.font("Tahoma", FontWeight.NORMAL, 20));

vbox.getChildren().add(title);

Label movieNameLabel = new Label("Movie Name:");

vbox.getChildren().add(movieNameLabel);

ComboBox<String> movieComboBox = new ComboBox<>();

try {

List<String> movieNames = getAllMovieNames();

movieComboBox.getItems().addAll(movieNames);

} catch (SQLException e) {

showErrorAlert("Error retrieving movie names: " + e.getMessage());

}

vbox.getChildren().add(movieComboBox);

Label reviewLabel = new Label("Review:");

vbox.getChildren().add(reviewLabel);

TextArea reviewField = new TextArea();

vbox.getChildren().add(reviewField);

Button addButton = new Button("Add");

addButton.setOnAction(e -> {

try {

String movieName = movieComboBox.getValue();

String reviewText = reviewField.getText();

addReview(currentUser, movieName, reviewText);

addReviewStage.close();

showAlert(Alert.AlertType.INFORMATION, "Success", "Review added successfully");

} catch (SQLException ex) {

showErrorAlert("Error adding review: " + ex.getMessage());

}

});

vbox.getChildren().add(addButton);

Scene addReviewScene = new Scene(vbox, 400, 500);

addReviewStage.setScene(addReviewScene);

addReviewStage.show();

}

private void showAlert(Alert.AlertType alertType, String title, String content) {

Alert alert = new Alert(alertType);

alert.setTitle(title);

alert.setContentText(content);

alert.showAndWait();

}

private void showErrorAlert(String message) {

showAlert(Alert.AlertType.ERROR, "Error", message);

}

private void addTitle(GridPane vbox, String titleText) {

Text title = new Text(titleText);

title.setFont(Font.font("Tahoma", FontWeight.NORMAL, 20));

vbox.getChildren().add(title);

}

private boolean showConfirmationDialog(String title, String message) {

Alert alert = new Alert(Alert.AlertType.CONFIRMATION);

alert.setTitle(title);

alert.setContentText(message);

ButtonType result = alert.showAndWait().orElse(ButtonType.CANCEL);

return result == ButtonType.OK;

}

public static void main(String[] args) {

launch(args);

}

interface LoginHandler {

boolean authenticate(String username, String password) throws SQLException;

}

private boolean authenticateUser(String username, String password, String tableName) throws SQLException {

String selectQuery = "SELECT \* FROM " + tableName + " WHERE username = ? AND password = ?";

try (PreparedStatement statement = connection.prepareStatement(selectQuery)) {

statement.setString(1, username);

statement.setString(2, password);

try (ResultSet resultSet = statement.executeQuery()) {

return resultSet.next();

}

}

}

private void registerUser(String username, String password, String tableName) throws SQLException {

String insertQuery = "INSERT INTO " + tableName + " (username, password) VALUES (?, ?)";

try (PreparedStatement statement = connection.prepareStatement(insertQuery)) {

statement.setString(1, username);

statement.setString(2, password);

statement.executeUpdate();

}

}

private void openWindow(String title) {

switch (title) {

case "Manage Customers":

openManageCustomersWindow();

break;

case "Manage Movies":

openManageMoviesWindow();

break;

case "View Movies":

openViewMoviesWindow();

break;

case "Make Booking":

openMakeBookingWindow();

break;

case "Review":

openReviewWindow();

break;

}

}

private void openManageCustomersWindow() {

Stage manageCustomersStage = new Stage();

manageCustomersStage.setTitle("Manage Customers");

VBox vbox = new VBox();

vbox.setSpacing(10);

vbox.setPadding(new Insets(20, 20, 20, 20));

Text title = new Text("Manage Customers");

title.setFont(Font.font("Tahoma", FontWeight.NORMAL, 20));

vbox.getChildren().add(title);

ListView<String> customerList = new ListView<>();

vbox.getChildren().add(customerList);

try {

List<String> customerDetails = getAllCustomerDetails();

customerList.getItems().addAll(customerDetails);

} catch (SQLException e) {

showErrorAlert("Error retrieving customers: " + e.getMessage());

}

Button backButton = new Button("Back");

backButton.setOnAction(e -> manageCustomersStage.close());

vbox.getChildren().add(backButton);

Scene manageCustomersScene = new Scene(vbox, 400, 400);

manageCustomersStage.setScene(manageCustomersScene);

manageCustomersStage.show();

}

private List<String> getAllCustomerDetails() throws SQLException {

String query = "SELECT c.username, b.movie\_name, b.theatre\_name FROM customers c " +

"LEFT JOIN bookings b ON c.username = b.customer\_name";

return getCustomerDetails(query);

}

private List<String> getCustomerDetails(String query) throws SQLException {

try (PreparedStatement statement = connection.prepareStatement(query);

ResultSet resultSet = statement.executeQuery()) {

List<String> customerDetails = new ArrayList<>();

while (resultSet.next()) {

String customerDetail = String.format("Customer: %s, Movie: %s, Theatre: %s",

resultSet.getString("username"),

resultSet.getString("movie\_name") != null ? resultSet.getString("movie\_name") : "None",

resultSet.getString("theatre\_name") != null ? resultSet.getString("theatre\_name") : "None");

customerDetails.add(customerDetail);

}

return customerDetails;

}

}

private void openManageMoviesWindow() {

Stage manageMoviesStage = new Stage();

manageMoviesStage.setTitle("Manage Movies");

VBox vbox = new VBox();

vbox.setSpacing(10);

vbox.setPadding(new Insets(20, 20, 20, 20));

Text title = new Text("Manage Movies");

title.setFont(Font.font("Tahoma", FontWeight.NORMAL, 20));

vbox.getChildren().add(title);

TableView<MovieTheatre> tableView = createMovieTheatreTableView();

vbox.getChildren().add(tableView);

updateTableView(tableView);

Runnable refreshTableView = () -> updateTableView(tableView);

Button addMovieButton = new Button("Add Movie");

addMovieButton.setOnAction(e -> openAddMovieWindow(manageMoviesStage, refreshTableView));

vbox.getChildren().add(addMovieButton);

Button deleteMovieButton = new Button("Delete Movie");

deleteMovieButton.setOnAction(e -> {

MovieTheatre selectedMovieTheatre = tableView.getSelectionModel().getSelectedItem();

if (selectedMovieTheatre != null) {

boolean confirmed = showConfirmationDialog("Confirm Delete", "Are you sure you want to delete the selected movie?");

if (confirmed) {

try {

deleteMovie(selectedMovieTheatre.getMovieName());

refreshTableView.run();

showAlert(Alert.AlertType.INFORMATION, "Success", "Movie deleted successfully");

} catch (SQLException ex) {

showErrorAlert("Error deleting movie: " + ex.getMessage());

}

}

} else {

showAlert(Alert.AlertType.ERROR, "Error", "No movie selected");

}

});

vbox.getChildren().add(deleteMovieButton);

Button backButton = new Button("Back");

backButton.setOnAction(e -> manageMoviesStage.close());

vbox.getChildren().add(backButton);

Scene manageMoviesScene = new Scene(vbox, 600, 400);

manageMoviesStage.setScene(manageMoviesScene);

manageMoviesStage.show();

}

private void openAddMovieWindow(Stage manageMoviesStage, Runnable refreshTableView) {

Stage addMovieStage = new Stage();

addMovieStage.setTitle("Add Movie");

VBox vbox = new VBox();

vbox.setSpacing(10);

vbox.setPadding(new Insets(20, 20, 20, 20));

Text title = new Text("Add Movie");

title.setFont(Font.font("Tahoma", FontWeight.NORMAL, 20));

vbox.getChildren().add(title);

Label movieNameLabel = new Label("Movie Name:");

vbox.getChildren().add(movieNameLabel);

TextField movieNameField = new TextField();

vbox.getChildren().add(movieNameField);

List<TextField> theatreFields = new ArrayList<>();

List<TextField> seatsFields = new ArrayList<>();

addTheatreFields(vbox, theatreFields, seatsFields);

Button addTheatreButton = new Button("Add Theatre");

addTheatreButton.setOnAction(e -> addTheatreFields(vbox, theatreFields, seatsFields));

vbox.getChildren().add(addTheatreButton);

Button addButton = new Button("Add");

addButton.setOnAction(e -> {

try {

String movieName = movieNameField.getText();

if (movieName.isEmpty()) {

showAlert(Alert.AlertType.ERROR, "Error", "Movie name cannot be empty");

return;

}

addMovie(movieName);

for (int i = 0; i < theatreFields.size(); i++) {

String theatreName = theatreFields.get(i).getText();

if (theatreName.isEmpty()) {

showAlert(Alert.AlertType.ERROR, "Error", "Theatre name cannot be empty");

return;

}

int seats = Integer.parseInt(seatsFields.get(i).getText());

addTheatre(movieName, theatreName, seats);

}

addMovieStage.close();

refreshTableView.run();

showAlert(Alert.AlertType.INFORMATION, "Success", "Movie and theatres added successfully");

} catch (SQLException ex) {

showErrorAlert("Error adding movie and theatres: " + ex.getMessage());

} catch (NumberFormatException ex) {

showAlert(Alert.AlertType.ERROR, "Error", "Invalid number of seats");

}

});

vbox.getChildren().add(addButton);

Scene addMovieScene = new Scene(vbox, 400, 500);

addMovieStage.setScene(addMovieScene);

addMovieStage.show();

}

private void addTheatreFields(VBox vbox, List<TextField> theatreFields, List<TextField> seatsFields) {

Label theatreNameLabel = new Label("Theatre Name:");

vbox.getChildren().add(theatreNameLabel);

TextField theatreNameField = new TextField();

vbox.getChildren().add(theatreNameField);

theatreFields.add(theatreNameField);

Label seatsLabel = new Label("Seats:");

vbox.getChildren().add(seatsLabel);

TextField seatsField = new TextField();

vbox.getChildren().add(seatsField);

seatsFields.add(seatsField);

}

private void addMovie(String movieName) throws SQLException {

String insertQuery = "INSERT INTO movies (name) VALUES (?)";

try (PreparedStatement statement = connection.prepareStatement(insertQuery)) {

statement.setString(1, movieName);

statement.executeUpdate();

}

}

private void addTheatre(String movieName, String theatreName, int seats) throws SQLException {

String insertQuery = "INSERT INTO theatres (movie\_name, theatre\_name, seats) VALUES (?, ?, ?)";

try (PreparedStatement statement = connection.prepareStatement(insertQuery)) {

statement.setString(1, movieName);

statement.setString(2, theatreName);

statement.setInt(3, seats);

statement.executeUpdate();

}

}

public class MovieTheatre {

private final SimpleStringProperty movieName;

private final SimpleStringProperty theatreName;

private final SimpleIntegerProperty seats;

public MovieTheatre(String movieName, String theatreName, int seats) {

this.movieName = new SimpleStringProperty(movieName);

this.theatreName = new SimpleStringProperty(theatreName);

this.seats = new SimpleIntegerProperty(seats);

}

public String getMovieName() {

return movieName.get();

}

public void setMovieName(String movieName) {

this.movieName.set(movieName);

}

public String getTheatreName() {

return theatreName.get();

}

public void setTheatreName(String theatreName) {

this.theatreName.set(theatreName);

}

public int getSeats() {

return seats.get();

}

public void setSeats(int seats) {

this.seats.set(seats);

}

}

public class Review {

private final SimpleStringProperty customerName;

private final SimpleStringProperty movieName;

private final SimpleStringProperty review;

public Review(String customerName, String movieName, String review) {

this.customerName = new SimpleStringProperty(customerName);

this.movieName = new SimpleStringProperty(movieName);

this.review = new SimpleStringProperty(review);

}

public String getCustomerName() {

return customerName.get();

}

public void setCustomerName(String customerName) {

this.customerName.set(customerName);

}

public String getMovieName() {

return movieName.get();

}

public void setMovieName(String movieName) {

this.movieName.set(movieName);

}

public String getReview() {

return review.get();

}

public void setReview(String review) {

this.review.set(review);

}

}

private void deleteMovie(String movieName) throws SQLException {

String deleteTheatresQuery = "DELETE FROM theatres WHERE movie\_name = ?";

try (PreparedStatement deleteTheatresStatement = connection.prepareStatement(deleteTheatresQuery)) {

deleteTheatresStatement.setString(1, movieName);

deleteTheatresStatement.executeUpdate();

}

String deleteMovieQuery = "DELETE FROM movies WHERE name = ?";

try (PreparedStatement deleteMovieStatement = connection.prepareStatement(deleteMovieQuery)) {

deleteMovieStatement.setString(1, movieName);

deleteMovieStatement.executeUpdate();

}

}

private List<String> getAllMovieNames() throws SQLException {

String query = "SELECT name FROM movies";

try (PreparedStatement statement = connection.prepareStatement(query);

ResultSet resultSet = statement.executeQuery()) {

List<String> movieNames = new ArrayList<>();

while (resultSet.next()) {

movieNames.add(resultSet.getString("name"));

}

return movieNames;

}

}

private List<String> getTheatresByMovie(String movieName) throws SQLException {

String query = "SELECT theatre\_name FROM theatres WHERE movie\_name = ?";

try (PreparedStatement statement = connection.prepareStatement(query)) {

statement.setString(1, movieName);

try (ResultSet resultSet = statement.executeQuery()) {

List<String> theatreNames = new ArrayList<>();

while (resultSet.next()) {

theatreNames.add(resultSet.getString("theatre\_name"));

}

return theatreNames;

}

}

}

private int getSeatsByTheatre(String theatreName) throws SQLException {

String query = "SELECT seats FROM theatres WHERE theatre\_name = ?";

try (PreparedStatement statement = connection.prepareStatement(query)) {

statement.setString(1, theatreName);

try (ResultSet resultSet = statement.executeQuery()) {

if (resultSet.next()) {

return resultSet.getInt("seats");

}

}

}

return 0;

}

private void makeBooking(String customerName, String movieName, String theatreName, int seats) throws SQLException {

String insertQuery = "INSERT INTO bookings (customer\_name, movie\_name, theatre\_name, seats) VALUES (?, ?, ?, ?)";

try (PreparedStatement statement = connection.prepareStatement(insertQuery)) {

statement.setString(1, customerName);

statement.setString(2, movieName);

statement.setString(3, theatreName);

statement.setInt(4, seats);

statement.executeUpdate();

}

}

private void addReview(String customerName, String movieName, String reviewText) throws SQLException {

String insertQuery = "INSERT INTO reviews (customer\_name, movie\_name, review) VALUES (?, ?, ?)";

try (PreparedStatement statement = connection.prepareStatement(insertQuery)) {

statement.setString(1, customerName);

statement.setString(2, movieName);

statement.setString(3, reviewText);

statement.executeUpdate();

}

}

private TableView<MovieTheatre> createMovieTheatreTableView() {

TableView<MovieTheatre> tableView = new TableView<>();

TableColumn<MovieTheatre, String> movieNameColumn = new TableColumn<>("Movie Name");

TableColumn<MovieTheatre, String> theatreNameColumn = new TableColumn<>("Theatre Name");

TableColumn<MovieTheatre, Number> seatsColumn = new TableColumn<>("Seats");

movieNameColumn.setCellValueFactory(new PropertyValueFactory<>("movieName"));

theatreNameColumn.setCellValueFactory(new PropertyValueFactory<>("theatreName"));

seatsColumn.setCellValueFactory(new PropertyValueFactory<>("seats"));

tableView.getColumns().add(movieNameColumn);

tableView.getColumns().add(theatreNameColumn);

tableView.getColumns().add(seatsColumn);

return tableView;

}

private TableView<Review> createReviewTableView() {

TableView<Review> tableView = new TableView<>();

TableColumn<Review, String> customerNameColumn = new TableColumn<>("Customer Name");

TableColumn<Review, String> movieNameColumn = new TableColumn<>("Movie Name");

TableColumn<Review, String> reviewColumn = new TableColumn<>("Review");

customerNameColumn.setCellValueFactory(new PropertyValueFactory<>("customerName"));

movieNameColumn.setCellValueFactory(new PropertyValueFactory<>("movieName"));

reviewColumn.setCellValueFactory(new PropertyValueFactory<>("review"));

tableView.getColumns().add(customerNameColumn);

tableView.getColumns().add(movieNameColumn);

tableView.getColumns().add(reviewColumn);

return tableView;

}

private void updateTableView(TableView<MovieTheatre> tableView) {

try {

ObservableList<MovieTheatre> movieTheatreList = FXCollections.observableArrayList(getAllMovieTheatreData());

tableView.setItems(movieTheatreList);

} catch (SQLException e) {

showErrorAlert("Error retrieving movie theatre data: " + e.getMessage());

}

}

private void updateReviewTableView(TableView<Review> tableView) {

try {

ObservableList<Review> reviewList = FXCollections.observableArrayList(getAllReviews());

tableView.setItems(reviewList);

} catch (SQLException e) {

showErrorAlert("Error retrieving reviews: " + e.getMessage());

}

}

private List<MovieTheatre> getAllMovieTheatreData() throws SQLException {

List<MovieTheatre> data = new ArrayList<>();

String query = "SELECT movie\_name, theatre\_name, seats FROM theatres";

try (PreparedStatement statement = connection.prepareStatement(query);

ResultSet resultSet = statement.executeQuery()) {

while (resultSet.next()) {

String movieName = resultSet.getString("movie\_name");

String theatreName = resultSet.getString("theatre\_name");

int seats = resultSet.getInt("seats");

data.add(new MovieTheatre(movieName, theatreName, seats));

}

}

return data;

}

private List<Review> getAllReviews() throws SQLException {

List<Review> data = new ArrayList<>();

String query = "SELECT customer\_name, movie\_name, review FROM reviews";

try (PreparedStatement statement = connection.prepareStatement(query);

ResultSet resultSet = statement.executeQuery()) {

while (resultSet.next()) {

String customerName = resultSet.getString("customer\_name");

String movieName = resultSet.getString("movie\_name");

String reviewText = resultSet.getString("review");

data.add(new Review(customerName, movieName, reviewText));

}

}

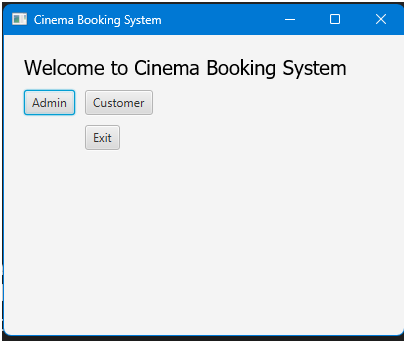
return data;

}

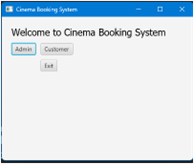
}

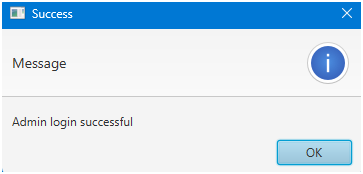
# OUTPUT

**Welcome Page**

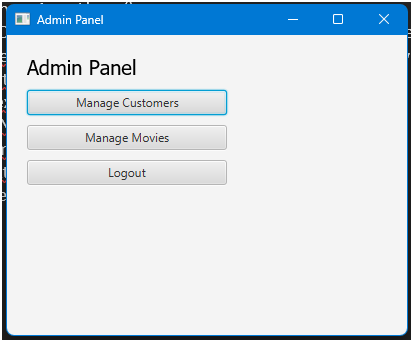


**Admin Login**

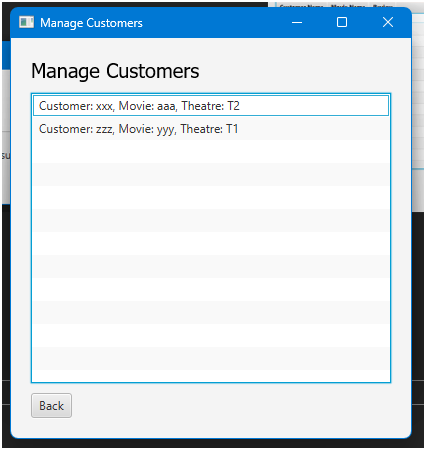
****



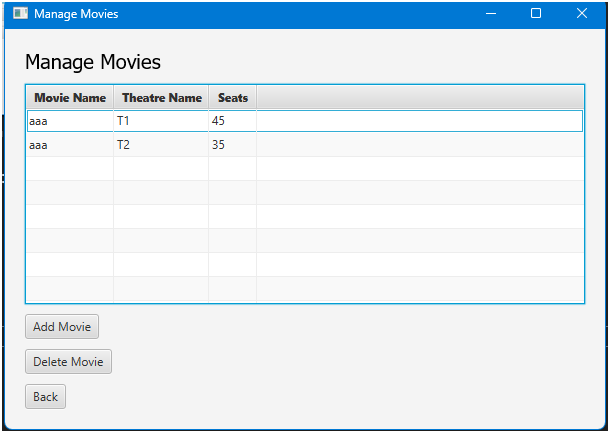
**Admin Panel**



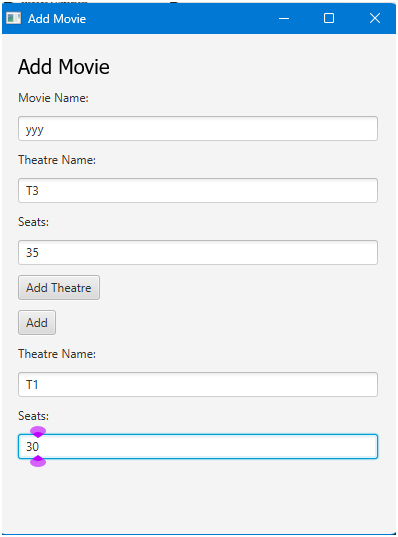
**Mange Customers**

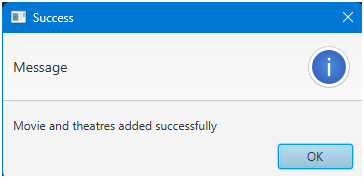


**Manage Movies**

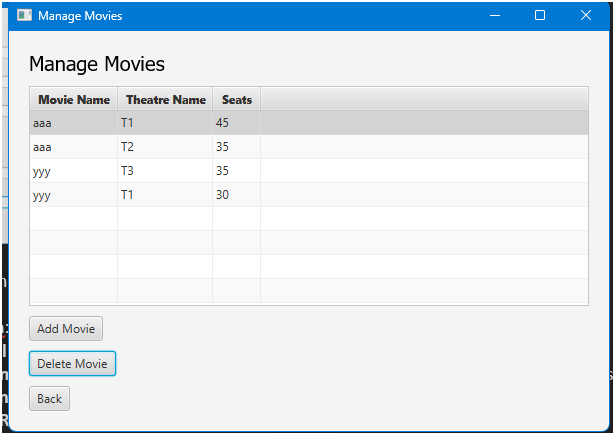


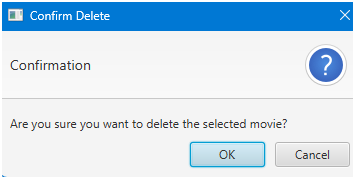
**Add Movies**

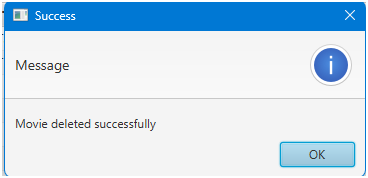




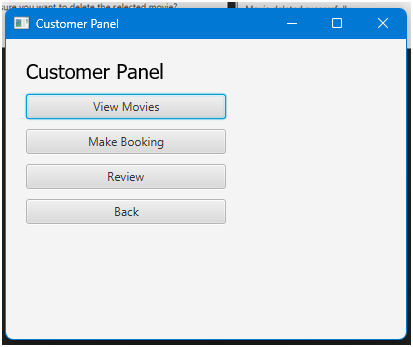
**Delete Movie**



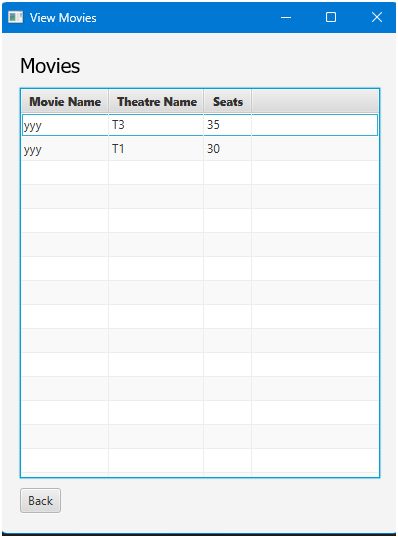




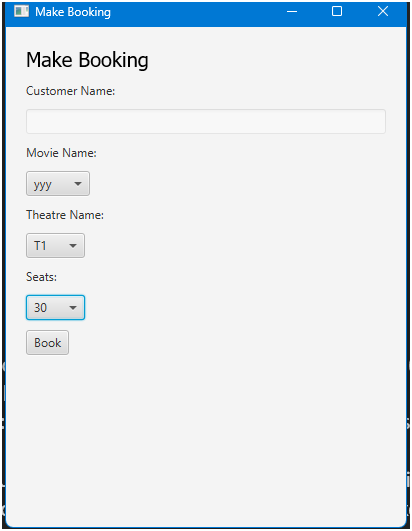
**Customer Panel**



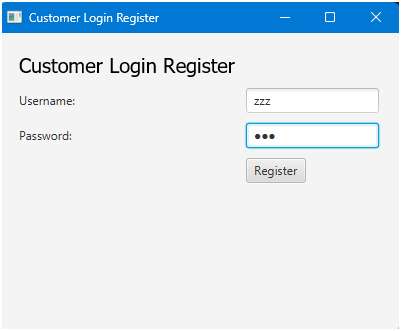
**View Booking**

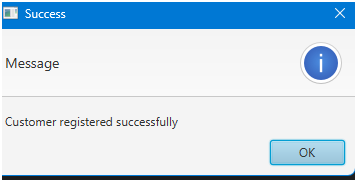


**Make Booking**

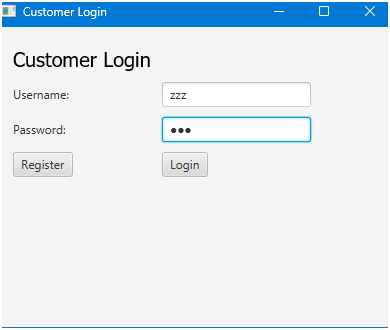


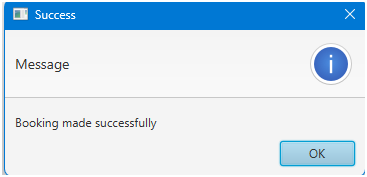
**Customer Register**



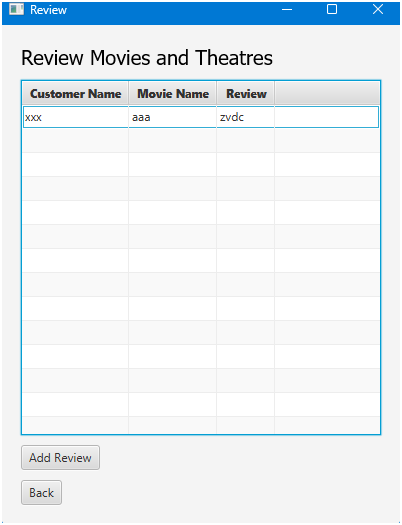


**Customer Login**

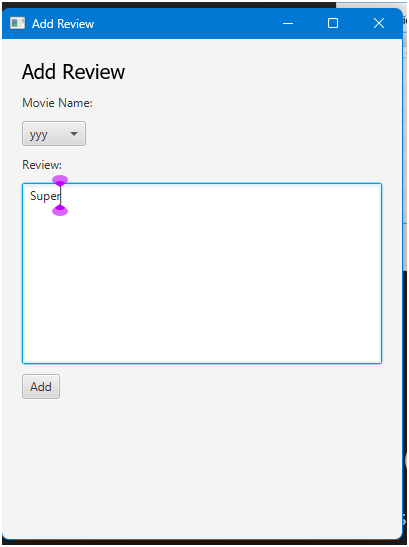


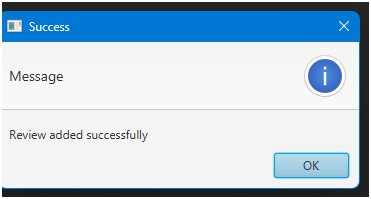


**Reviews**



**Add Review**





# CONCLUSION

The Cinema Booking System project successfully integrates JavaFX for a user-friendly interface and MySQL Workbench for robust database management, streamlining the movie ticket reservation process for both administrators and customers. The system's key functionalities—secure user authentication, movie and theatre management, real-time seat availability tracking, booking reservations, and customer reviews—offer a comprehensive and efficient solution that enhances the overall user experience. Adhering to database normalization principles ensures data integrity, reduces redundancy, and improves performance, making the system reliable and scalable. This project demonstrates the effective use of modern technologies to create a user-centric application, providing a solid foundation for future enhancements and expanded features.