LONDON METROPOLITAN UNIVERSITY

Module Code & Module Title

CS5003NI Data Structure and Specialist Programming

30% Individual Coursework

Final Submission

Academic Semester: AY 2025/2026

Student Name: Abiram Bhatta
London Met ID: 24046558
College ID: NP01AI4A20144

Assignment Due Date: May 17, 2026 (6:00pm)

Assignment Submission Date: May 17, 2026

Word Count: 9711

GIT HUB LINK: Link

I confirm that I understand my coursework needs to be submitted online via Google Classroom under the relevant module page before the deadline in order for my assignment to be accepted and marked. I am fully aware that late submissions will be treated as non-submission and a marks of zero will be awarded.

Table of Contents

Table of Contents

Table of Tables

1. Introduction

1.1 Project Title

MovieMint: A Dynamic Movie Booking Management System.

1.2 Purpose of the Project

MovieMint is a full-stack cinema booking and management platform designed to digitize the main operations of a small or medium cinema environment. The application allows customers to create accounts, sign in securely, browse available movies, view detailed movie information, select seats, book tickets, view confirmed bookings, and manage their profile details. It also provides a protected administrative area where cinema staff can manage movies, users, bookings, show schedules, ticket prices, hall layouts, and dashboard statistics.

The purpose of the project is not only to provide an online ticket booking interface but also to demonstrate a structured Java web application that uses data structures, database relationships, object-oriented programming, validation, exception handling, and role-based access control. The system replaces disconnected manual tasks with a centralized application where information is stored in MySQL and presented through dynamic JSP pages.

1.3 Intended Audience

The primary audience consists of cinema customers who need a convenient way to check movies, select show times, and book seats without depending on a physical ticket counter. The secondary audience consists of cinema administrators and management staff who need a reliable platform for adding movies, reviewing bookings, updating user roles, configuring hall layouts, adjusting seat prices, and monitoring revenue. The system is also suitable as a learning project because it combines practical web development with database design, Java classes, MVC organization, and testing.

1.4 Aims

- To create a practical web-based cinema booking platform that supports both customer and administrator workflows.
- To apply Java, JSP, Servlets, JDBC, and MySQL in a complete MVC-based application.
- To implement dynamic booking logic, including seat selection, seat categories, price calculation, and booking confirmation.
- To demonstrate secure access control through sessions, authentication filtering, user roles, and password reset support.
- To provide management features that help administrators supervise movies, show times, users, bookings, and revenue information.

1.5 Objectives

- Design a relational database that stores users, movies, show times, bookings, hall configurations, and global system settings.
- Create registration, login, logout, profile update, and password reset functions for users.
- Protect private and administrative routes through session validation and role-based authorization.
- Allow customers to browse now showing and upcoming movies using search, genre, language, and date filters.
- Provide a movie detail page that displays movie metadata, poster information, trailer access, synopsis, cast, runtime, format, and age rating.
- Build a dynamic seat map that shows available, selected, and already booked seats for the selected show.
- Calculate ticket cost using standard, premium, recliner, and VIP pricing rules.
- Allow administrators to add, update, hide, and schedule movies using a management interface.
- Use dashboard charts and summary cards to display revenue, bookings, users, movie performance, and recent activity.
- Test the application with functional, validation, exception handling, security, and user interface test cases.

1.6 System Overview

The system follows a monolithic MVC design. Models represent the main data entities, DAO classes communicate with MySQL, controllers process HTTP requests, JSP pages display the interface, and an authentication filter protects private routes. The implementation is organised into config, controllers, dao, filter, model, service, and util packages under src/main/java/com/moviebooking, while JSP pages and static assets are placed under src/main/webapp.

2. Problem Statement

2.1 Limitations of Existing Systems

- **Manual ticket booking:** Customers may be required to visit the cinema counter, ask staff about show times, wait in queues, and rely on verbal confirmation of seat availability. This increases service time and creates a greater risk of communication or recording errors.
- **Fragmented information:** Movie details, customer records, bookings, and schedules may be stored in separate spreadsheets or paper-based records. This fragmentation makes searching, reporting, updating, and maintaining consistent information more difficult.
- **Weak seat availability control:** Without a dynamic seat map, staff may accidentally assign the same seat twice or provide unclear information about available seats.
- **Limited administration:** Basic systems often support booking but do not provide full management of movies, users, halls, pricing, show times, and revenue analytics from one dashboard.
- **Poor security:** If roles and sessions are not handled properly, normal users may access administrative pages or sensitive functions.
- **Low reporting value:** Manual systems do not easily show revenue trends, most booked movies, recent bookings, or ticket category distribution.
- **Inconsistent validation:** Invalid names, emails, phone numbers, dates, prices, and passwords can enter the system if controller and DAO checks are missing.

2.2 Proposed Solution

MovieMint addresses these problems by providing a centralized web application supported by MySQL. Customers interact with public and protected user pages, while administrators use a separate dashboard and management pages. The booking process is data-driven: seats, prices, hall layouts, movie schedules, and already booked seats are loaded from the database and converted into dynamic page data. This reduces manual mistakes and creates a clearer booking experience.

The application also improves maintainability through MVC separation. Servlet controllers do not directly contain presentation markup, JSP pages do not perform database operations, and DAO classes isolate SQL behaviour. This division makes the system easier to test, debug, extend, and evaluate as a software engineering artefact.

2.3 Scope of the System

The current scope includes user registration, login, password reset, movie browsing, movie detail viewing, ticket booking, booking history, profile management, administrative dashboard, movie management, user management, booking management, settings management, hall layout configuration, pricing configuration, analytics display, and error pages. The system is intended as a complete web-based application for cinema operations rather than a simple static booking form.

3. Wireframe and Prototype Design

The wireframe design defines the visible structure of the system before implementation. It shows the expected navigation, form placement, content hierarchy, booking flow, and administrative workflows. The updated wireframes cover authentication, customer pages, administrator pages, support pages, and error handling pages.

Wireframe approach: The wireframes are low-fidelity black-and-white screens, which makes them useful for focusing on layout, navigation, and page functionality before final styling is applied.

3.1 Login page

The login screen provides email and password fields, links to registration and password recovery, and a clear entry point for existing users.

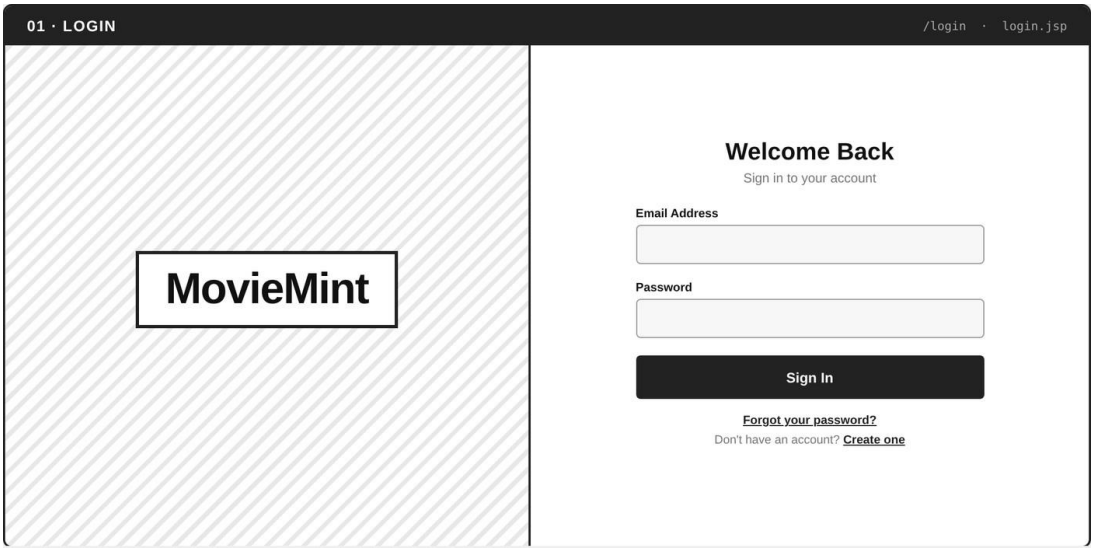


Figure 1: Login page wireframe

3.2 Registration page

The registration screen captures the user details needed to create an account and introduces the customer to the MovieMint system.

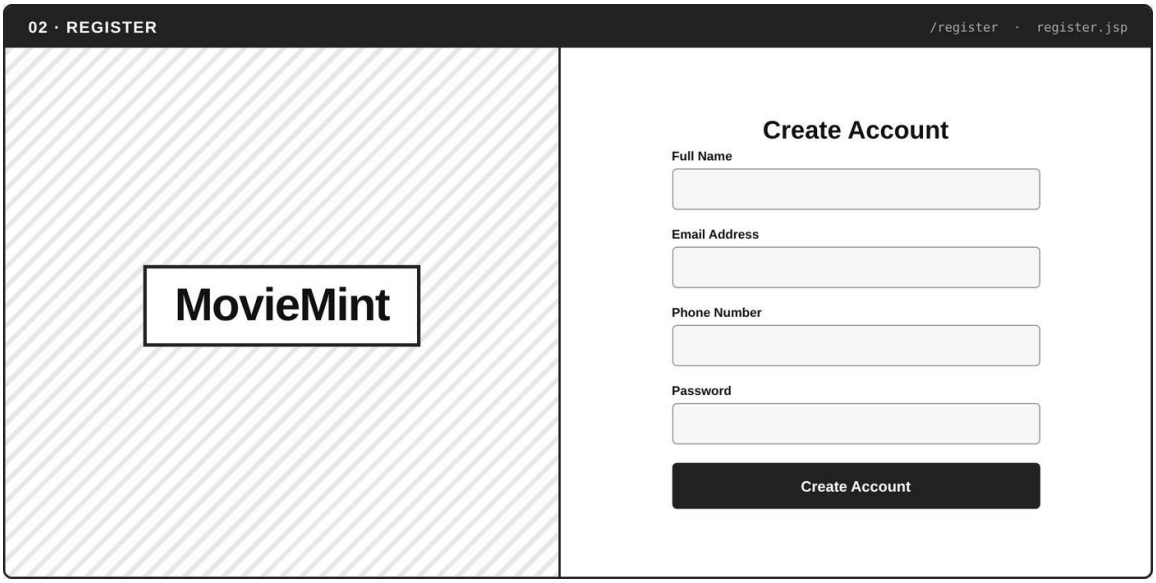


Figure 2: Registration page wireframe

3.3 Forgot password page

The password recovery design shows the email, OTP verification, and new password states used for controlled account recovery.

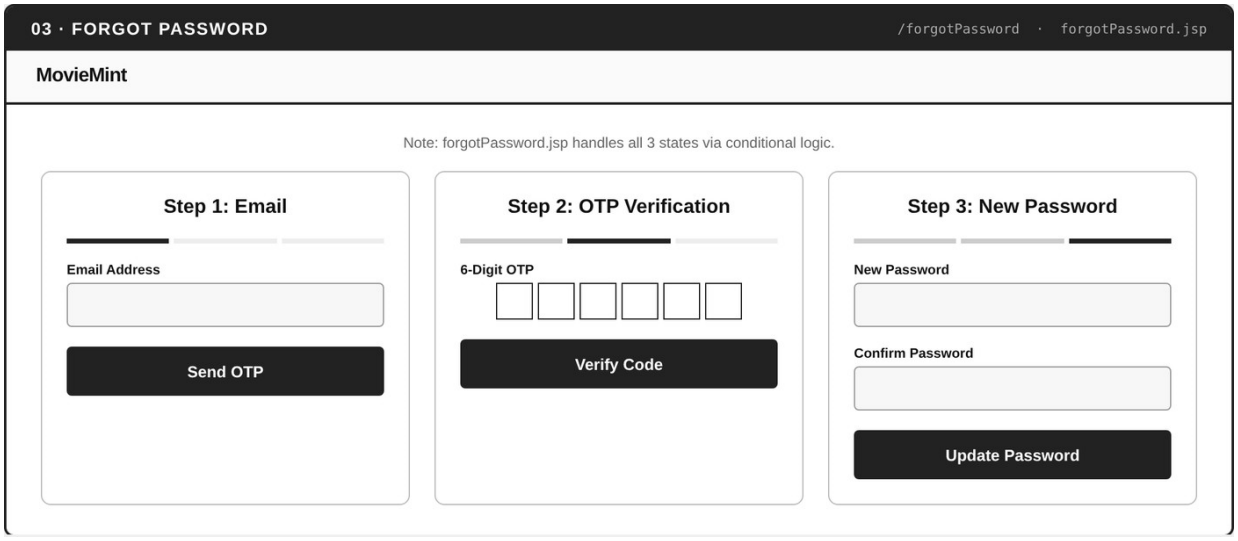


Figure 3: Forgot password page wireframe

3.4 User home page

The user home page welcomes the signed-in customer and highlights recently added or now-showing movies.

Figure 4: User home page wireframe

3.5 Browse movies page

The browse page provides movie discovery with status tabs, date filters, search, genre filtering, and language filtering.

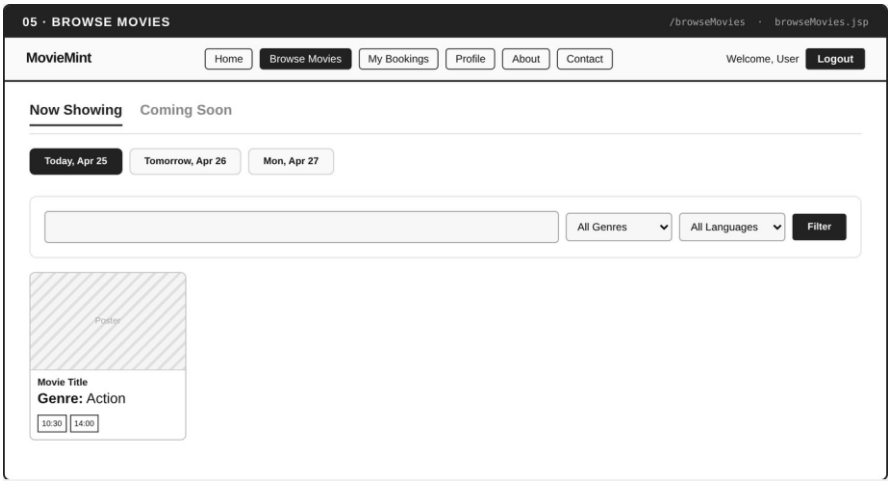


Figure 5: Browse movies page wireframe

3.6 Movie details page

The movie detail screen presents poster, trailer access, synopsis, cast, director, format, language, runtime, and booking actions.

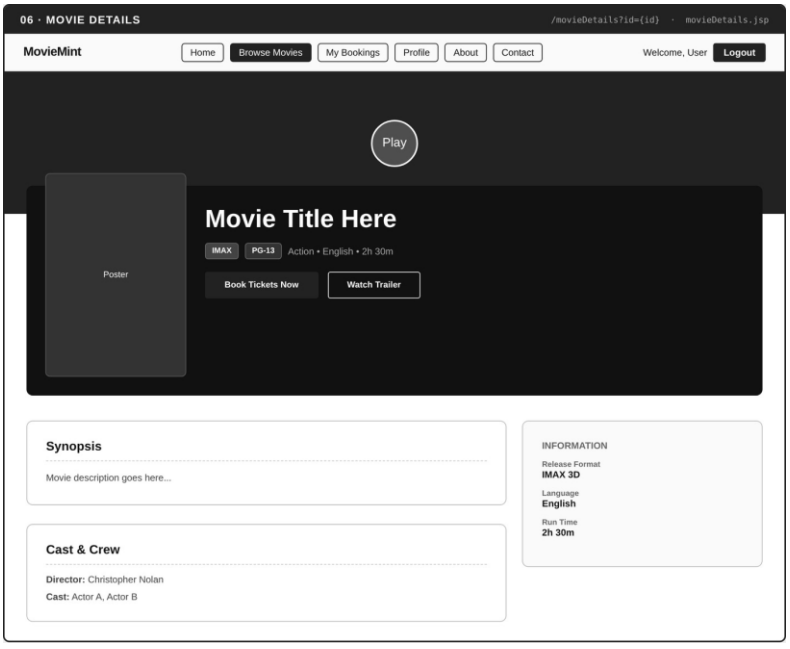


Figure 6: Movie details page wireframe

3.7 Book ticket page

The booking screen displays an interactive seat map, visual seat states, selected seats, seat tier, price per seat, and total price.

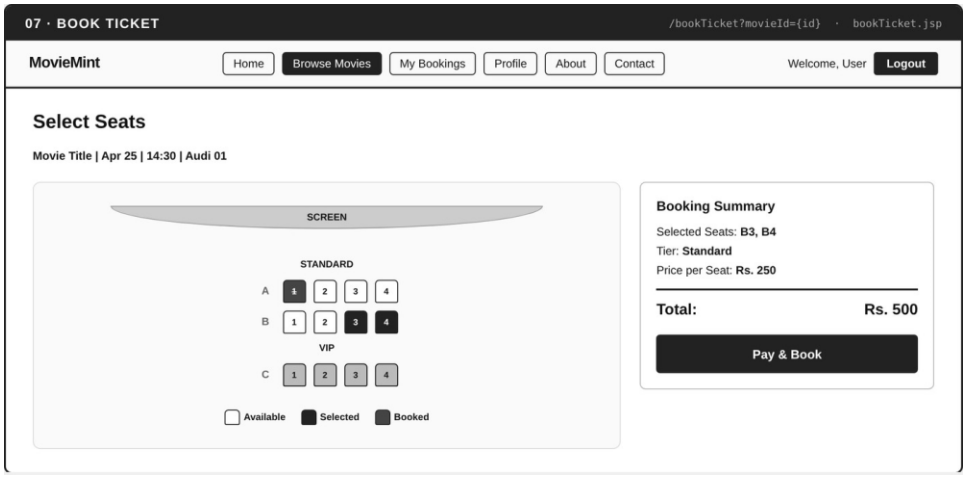


Figure 7: Book ticket page wireframe

3.8 Ticket confirmation page

The confirmation screen provides a ticket-style summary with movie title, date, time, hall, seats, QR placeholder, total price, and print action.

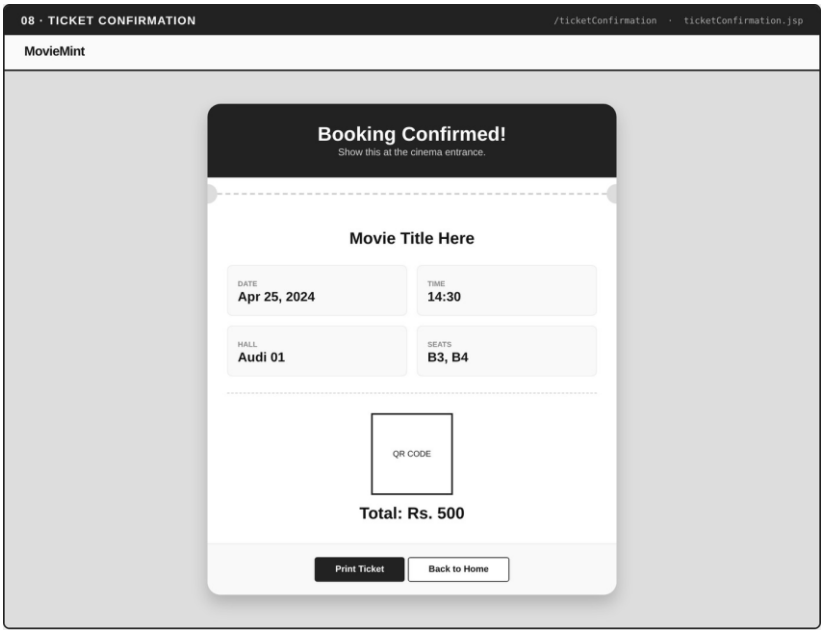


Figure 8: Ticket confirmation page wireframe

3.9 My bookings page

The booking history page allows customers to view their previous and current reservations along with status and ticket access.

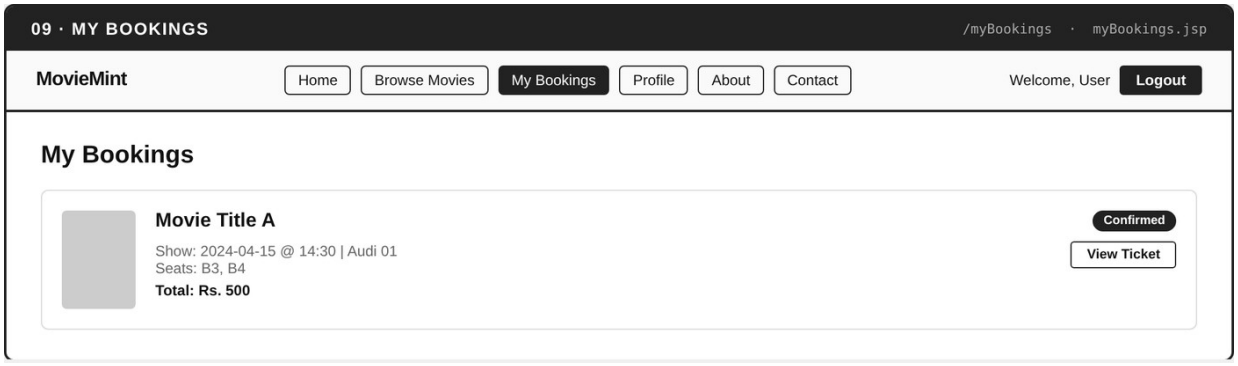


Figure 9: My bookings page wireframe

3.10 User profile page

The profile page allows customers to view and update basic account details.

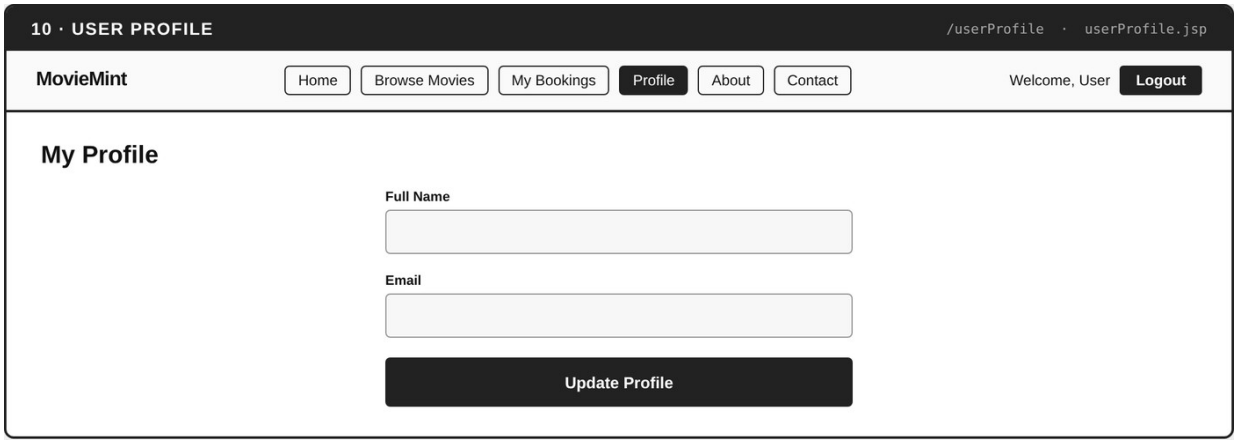


Figure 10: User profile page wireframe

3.11 Admin dashboard page

The dashboard provides performance cards, revenue charts, movie performance charts, recent bookings, and most booked movie information.

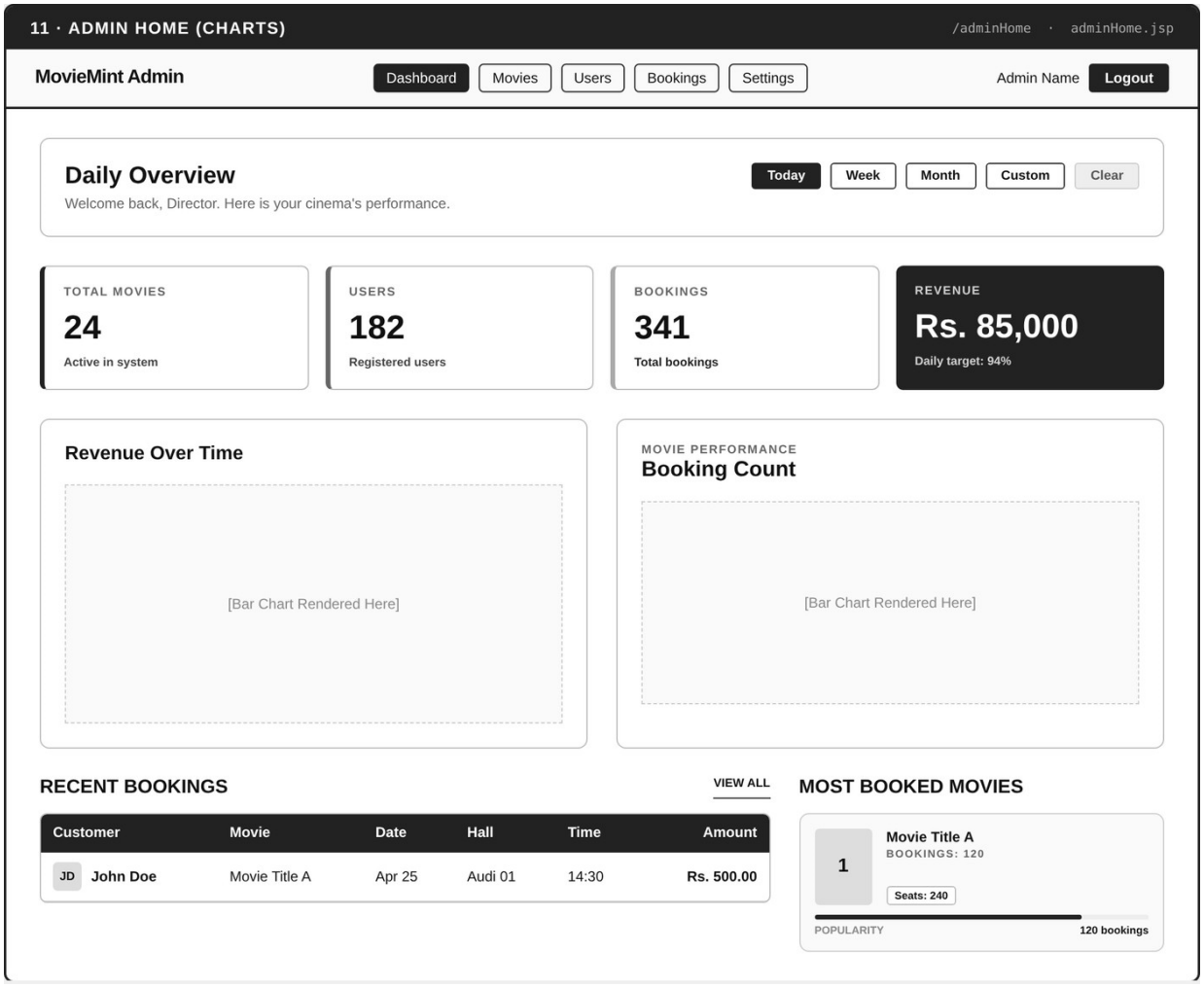


Figure 11: Admin dashboard page wireframe

3.12 Manage movies page

The movie catalog screen supports searching, filtering, status review, movie actions, and catalog-level summary cards.

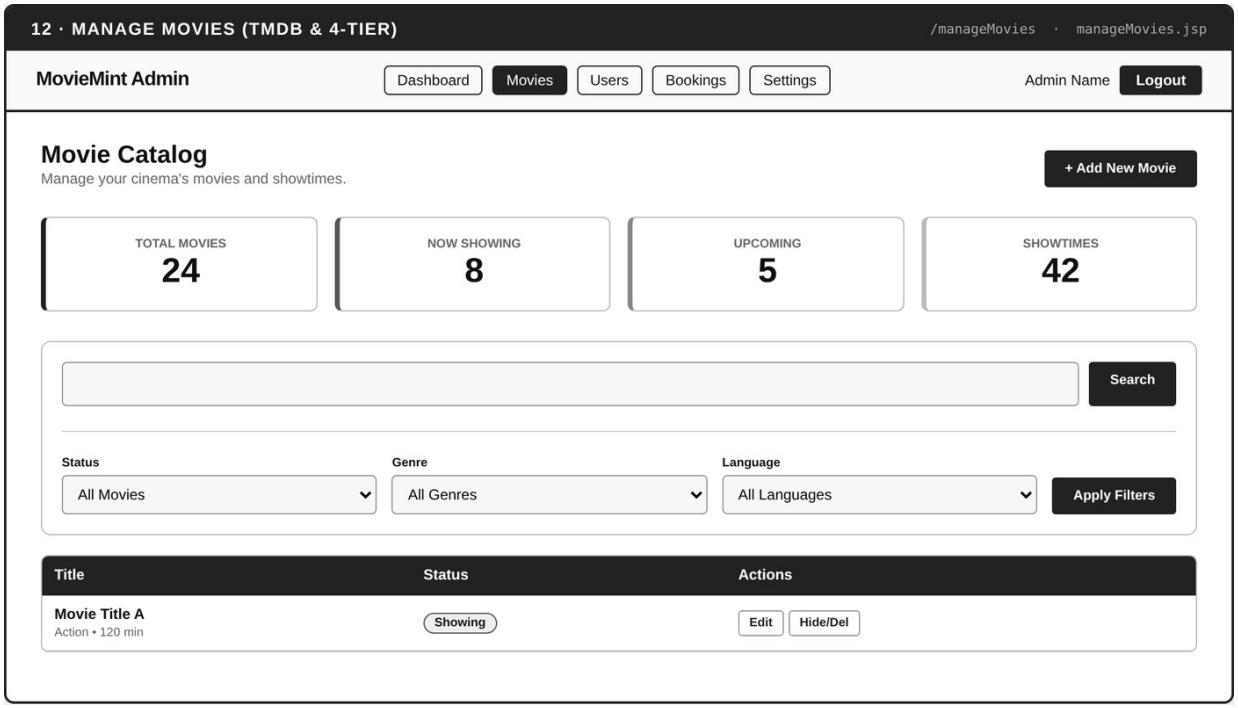


Figure 12: Manage movies page wireframe

3.13 Manage movies add/edit form

The add/edit state includes TMDB auto-fill, basic movie details, format, age rating, trailer URL, cast, description, pricing overrides, and schedule builder.

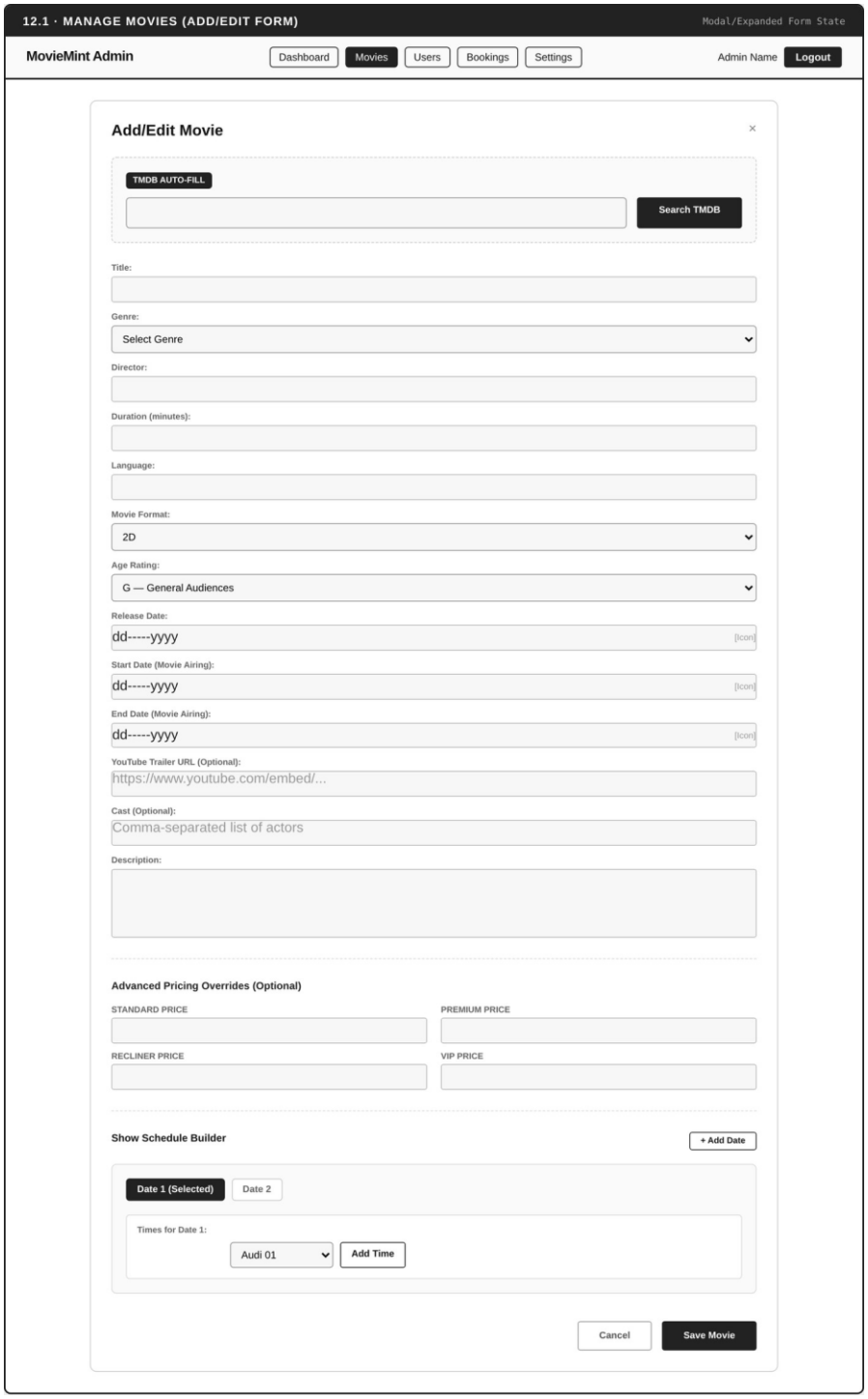


Figure 13: Manage movies add/edit form wireframe

3.14 Manage users page

The user directory provides administrator tools for viewing accounts, roles, user activity, and account status.

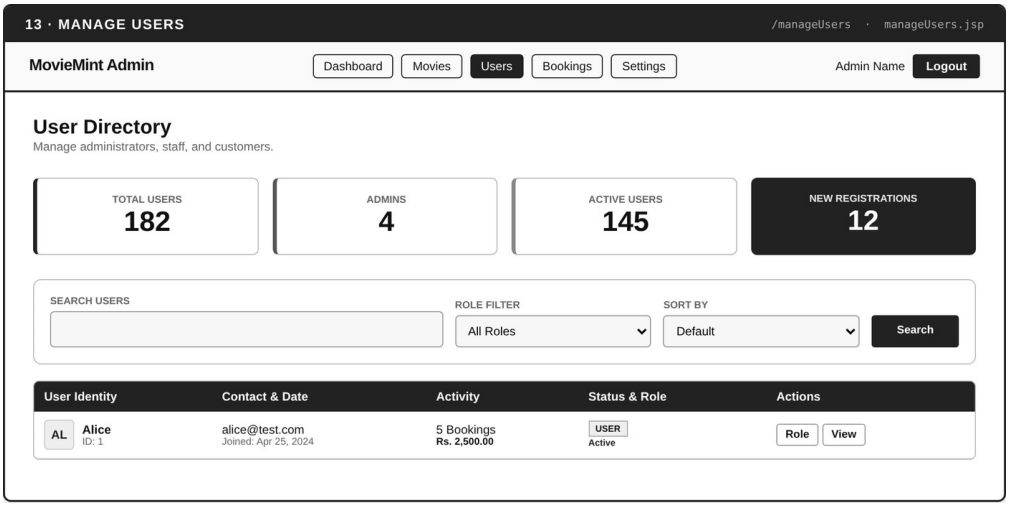


Figure 14: Manage users page wireframe

3.15 Manage bookings page

The booking management screen allows administrators to search, filter, review, update, and delete booking records.

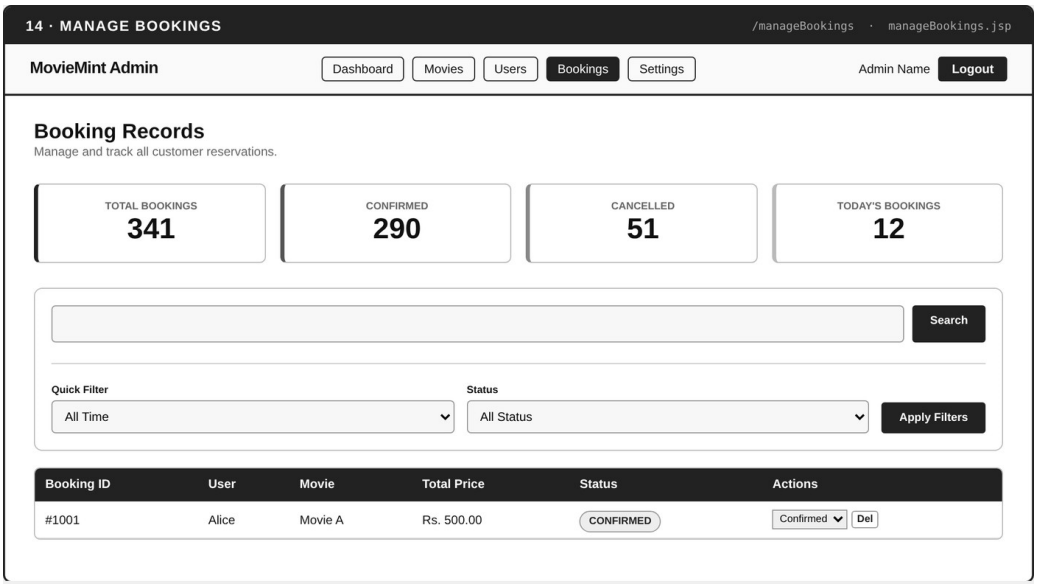


Figure 15: Manage bookings page wireframe

3.16 Manage settings page

The settings interface controls base prices, seat types, hall layouts, and seat map configuration.

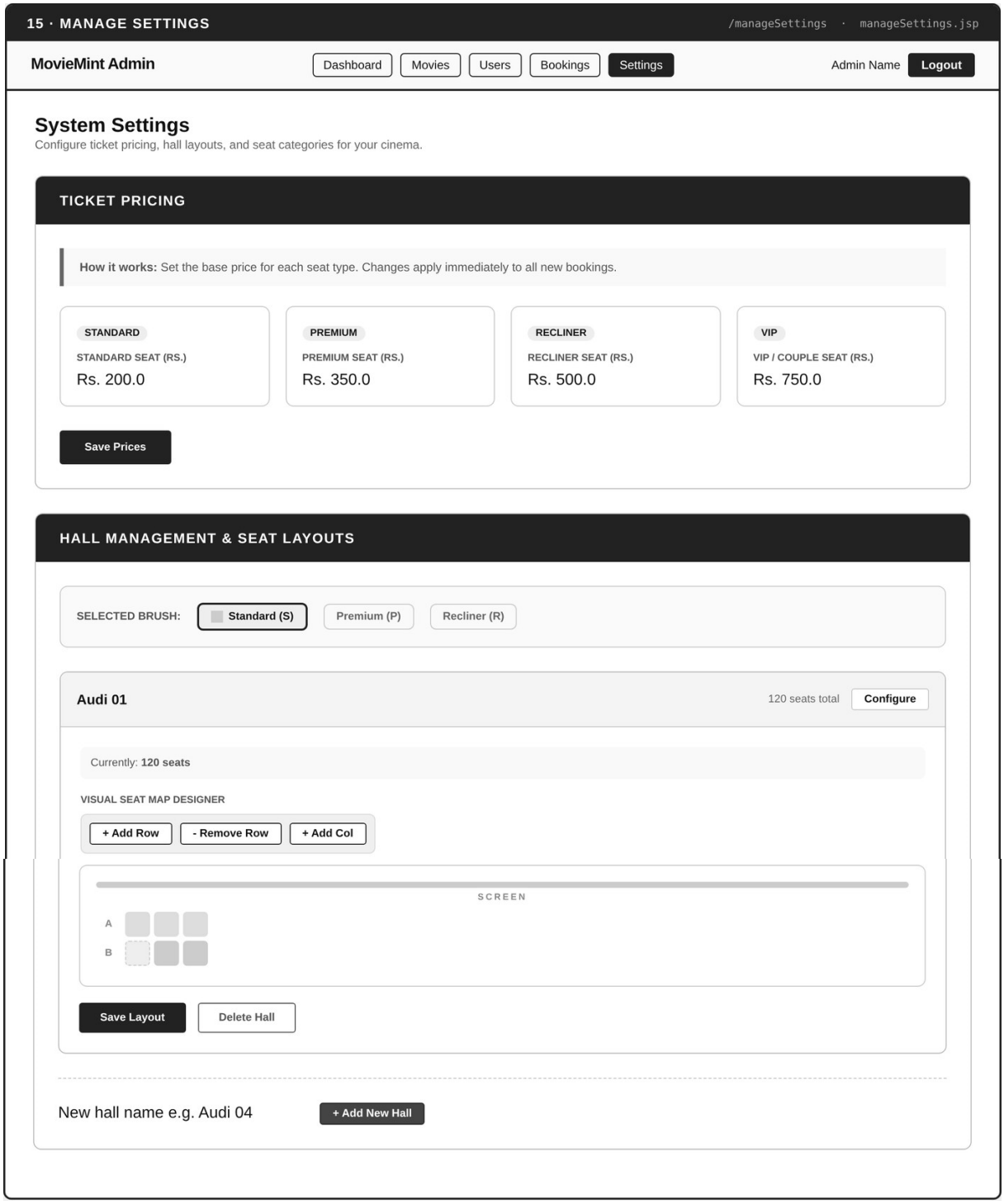


Figure 16: Manage settings page wireframe

3.17 About page

The about page explains the system purpose and gives the application a public information area.

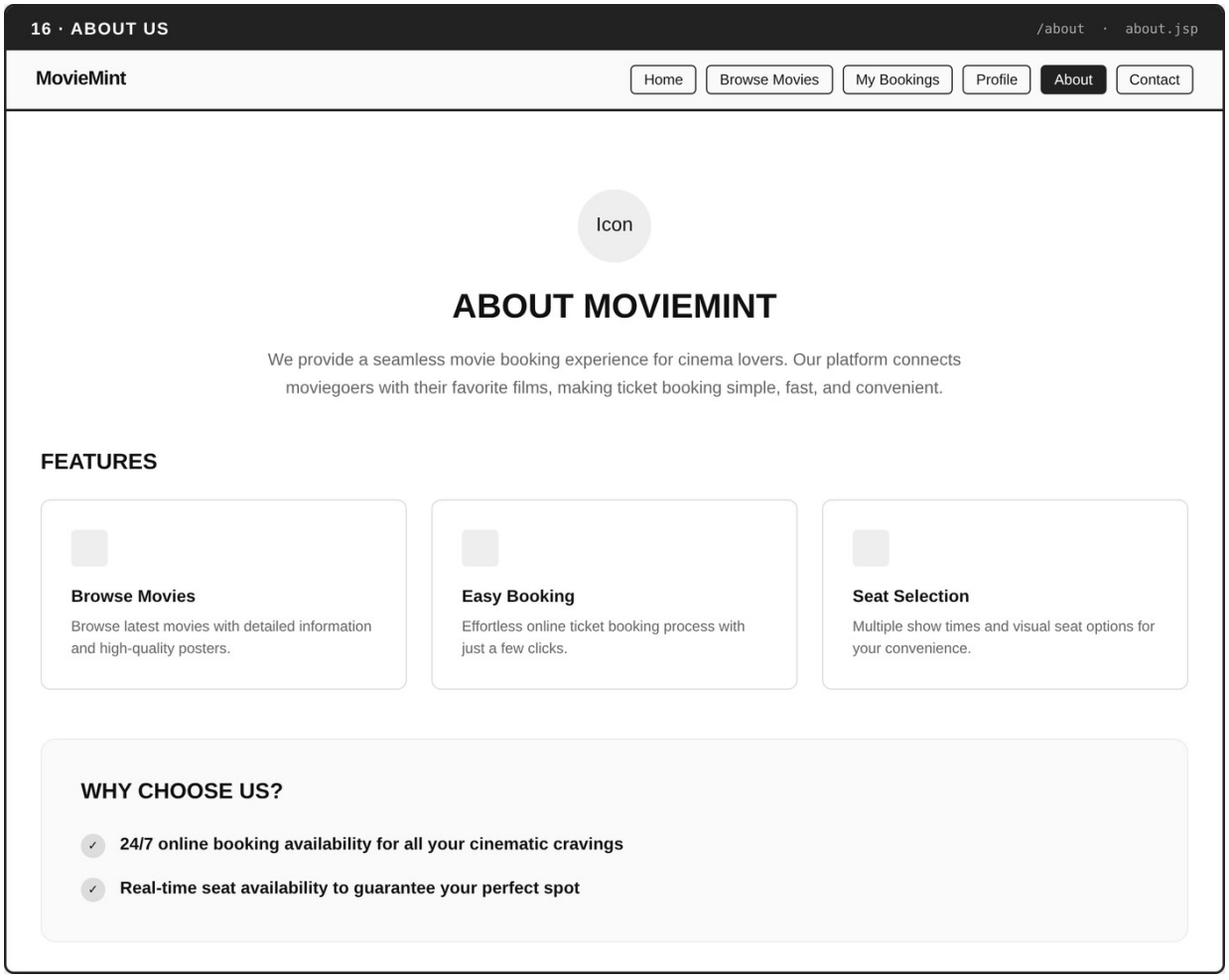


Figure 17: About page wireframe

3.18 Contact page

The contact page provides an inquiry form and support information for customer communication.

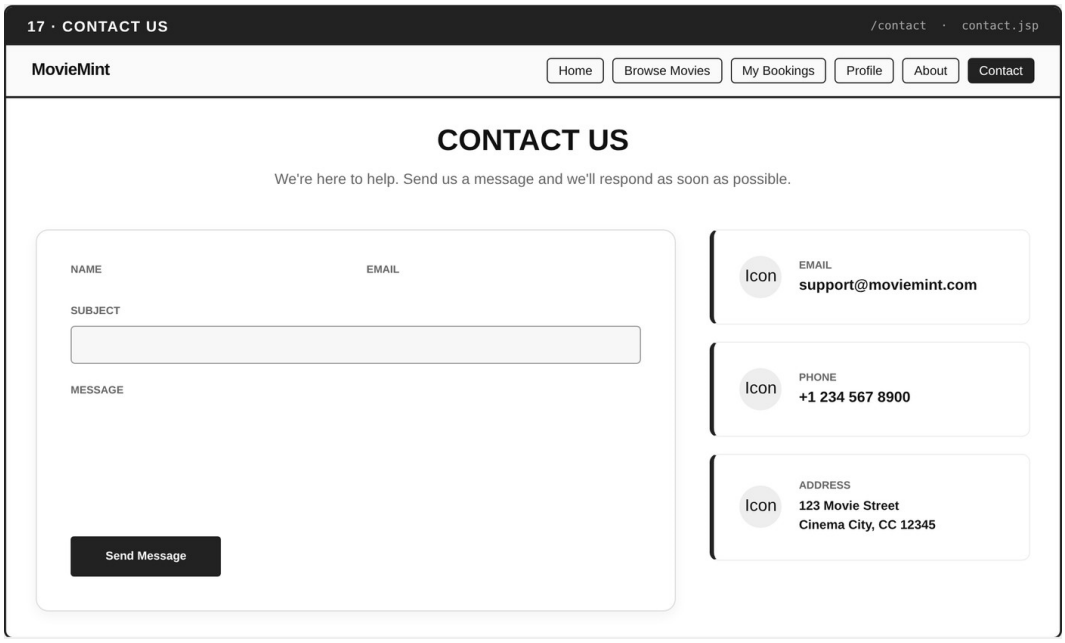


Figure 18: Contact page wireframe

3.19 Error 500 page

The server error page gives a clean fallback when an unexpected internal problem occurs.

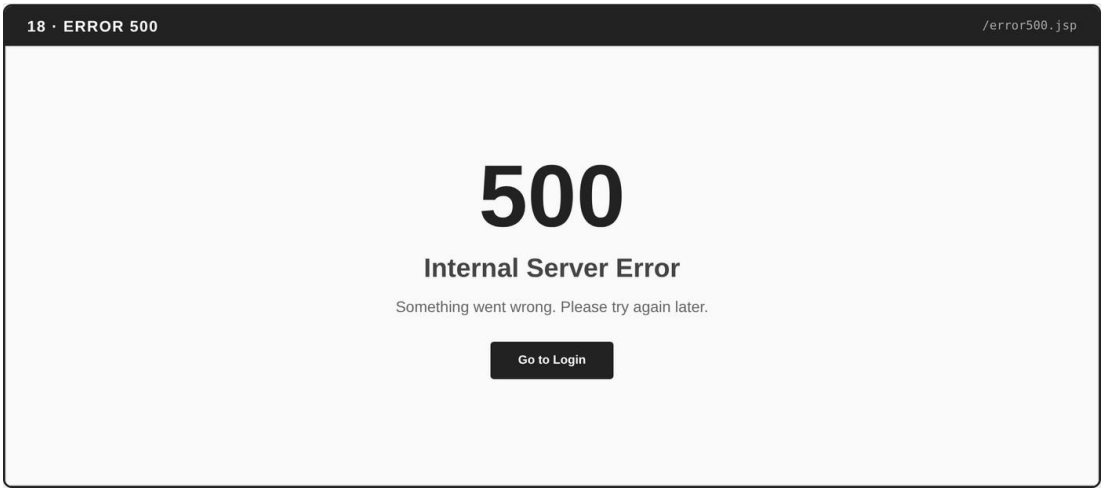


Figure 19: Error 500 page wireframe

3.20 Error 404 page

The not found page improves user experience when a user reaches an invalid or missing route.

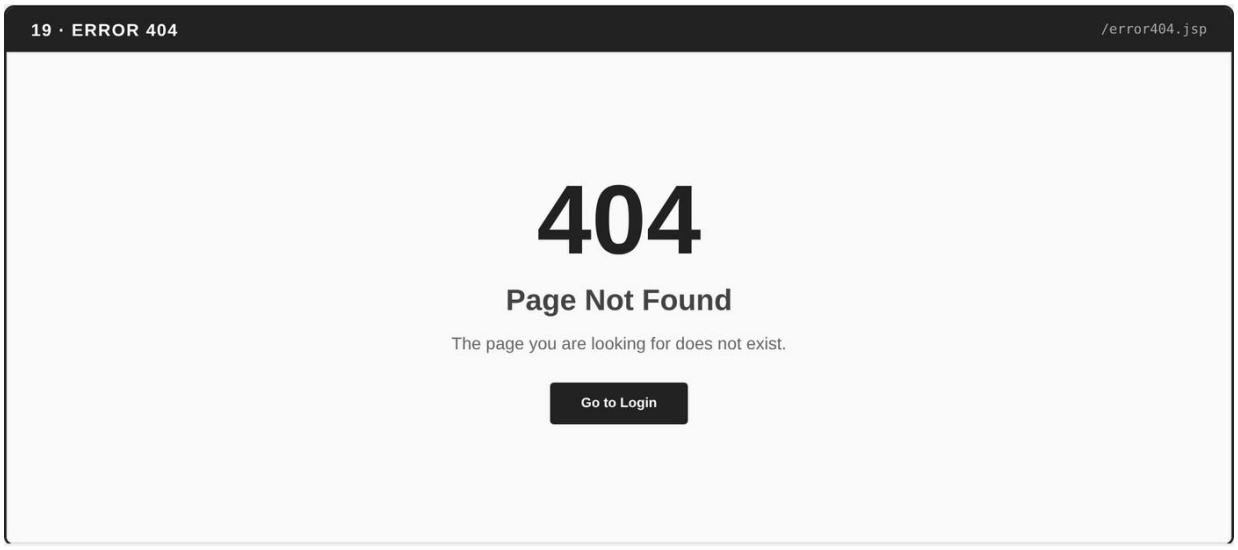


Figure 20: Error 404 page wireframe

4. Java Classes and Methods Explanation

The Java application is organized around MVC. Model classes hold data, DAO classes perform database access, controller servlets coordinate requests and responses, utility classes provide reusable support functions, and JSP pages render the final interface. This structure improves readability because each layer has a defined responsibility.

4.1 Class Diagram

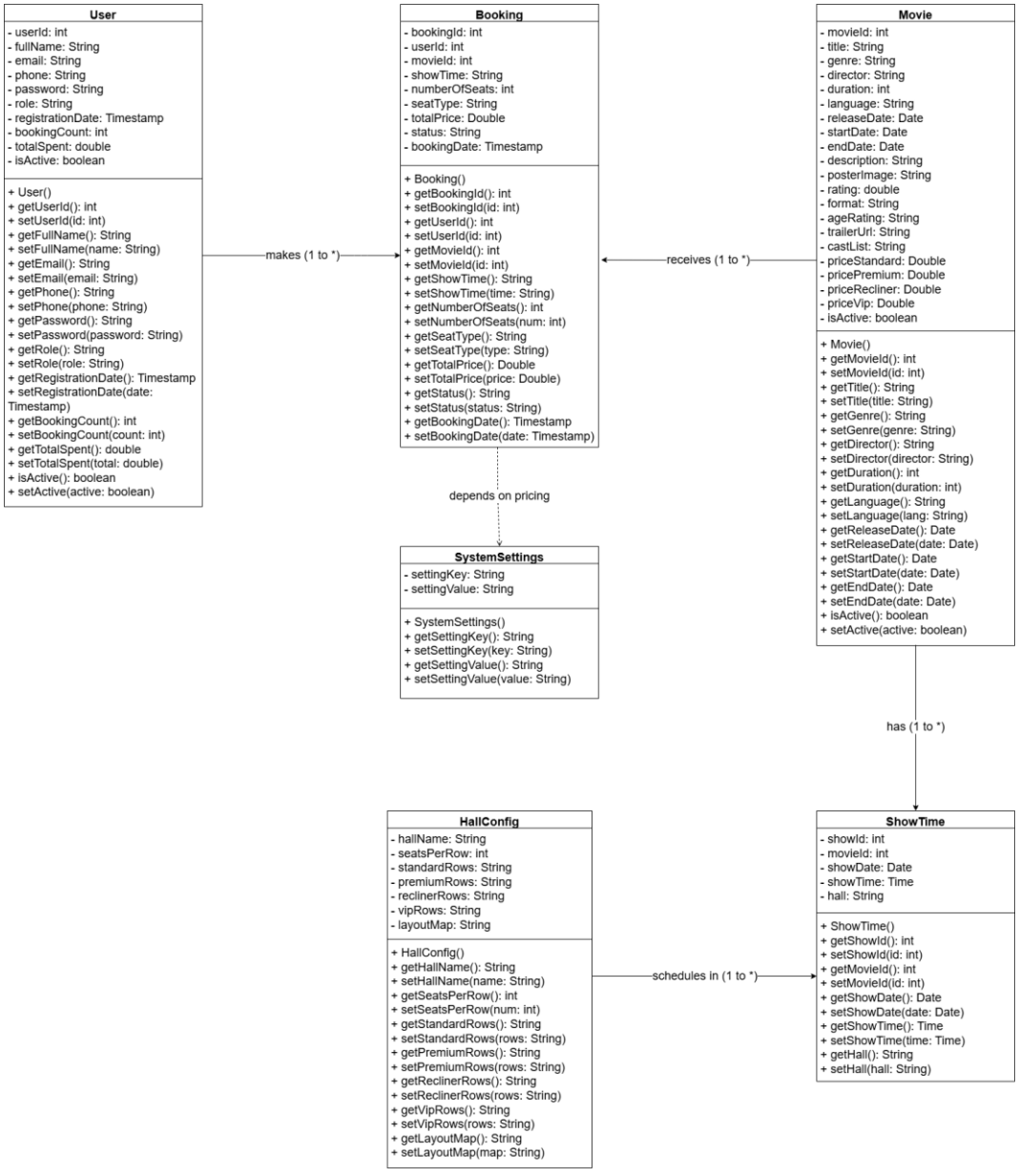


Figure 21: Main MVC class diagram

4.2 Main Java Classes and Methods

The main Java classes are grouped below according to their MVC responsibility. This structure keeps the explanation close to the design of the implemented system because each class is described in relation to its layer, purpose, and the methods that support its behaviour. The class explanations remain concise, while the added method explanations show how the Java code supports authentication, catalogue management, scheduling, booking, analytics, email delivery, and system configuration.

4.2.1 Model Classes

The model layer defines the data objects that move between servlets, DAO classes, services, and JSP pages. These classes mainly contain constructors, getter methods, setter methods, and state fields that represent database records. They allow the rest of the application to work with structured Java objects instead of unorganised request parameters or raw database rows.

User: Represents a customer or administrator account. It stores the user identifier, full name, email address, phone number, hashed password, role, account creation date, and password reset fields.

Method explanation:

- **Constructors** - create User objects when records are loaded from the database or when new registration data is prepared for insertion.
- **Getter methods** - such as getUserId(), getFullName(), getEmail(), getPhone(), getPassword(), getRole(), getCreatedAt(), getResetOtp(), and getOtpExpiry() expose user data to DAO, servlet, and JSP layers.
- **Setter methods** - such as setFullName(), setEmail(), setPhone(), setPassword(), setRole(), setResetOtp(), and setOtpExpiry() update user object state before persistence or display.

Movie: Represents a movie catalogue item. It stores title, genre, director, runtime, language, release and showing dates, description, poster image, format, age rating, trailer URL, cast list, pricing overrides, and active status.

Method explanation:

- **Constructors** - create Movie objects for admin catalogue operations, public movie listings, and detailed movie views.
- **Getter methods** - such as getMovieId(), getTitle(), getGenre(), getDirector(), getDuration(), getLanguage(), getReleaseDate(), getStartDate(), getEndDate(), getPosterImage(), getTrailerUrl(), and getCastList() provide movie information to JSP pages and controller logic.
- **Pricing access methods** - such as getPriceStandard(), getPricePremium(), getPriceRecliner(), and getPriceVip() support the four-tier ticket pricing calculation on the booking page.
- **State methods** - such as isActive() and setActive() support the soft-delete approach by hiding inactive movies without removing historical booking data.

Booking: Represents a ticket transaction. It stores the booking identifier, user reference, movie reference, show-time description, selected seats, total price, booking status, and booking date.

Method explanation:

- **Constructors** - prepare Booking objects when a user confirms a ticket or when existing bookings are loaded for customer and admin views.
- **Getter methods** - such as getBookingId(), getUserId(), getMovieId(), getShowTime(), getNumberOfSeats(), getSeatType(), getTotalPrice(), getStatus(), and getBookingDate() expose booking details to ticket confirmation pages, booking history, and dashboard analytics.
- **Setter methods** - such as setSeatType(), setTotalPrice(), and setStatus() allow the booking state to be prepared or changed before it is written back to the database.

ShowTime: Represents a scheduled screening slot by connecting a movie with a date, time, and hall.

Method explanation:

- **Getter methods** - such as getShowId(), getMovieId(), getShowDate(), getShowTime(), and getHall() provide schedule information for movie details, browsing, and booking workflows.
- **Setter methods** - such as setMovieId(), setShowDate(), setShowTime(), and setHall() prepare schedule data entered by the administrator before validation and database insertion.

HallConfig: Represents the physical hall layout, including seats per row, row category mappings, and the layout map used to generate the dynamic seat plan.

Method explanation:

- **Getter methods** - such as getHallName(), getSeatsPerRow(), getStandardRows(), getPremiumRows(), getReclinerRows(), getVipRows(), and getLayoutMap() provide the structure required to render the seat map.
- **Setter methods** - such as setSeatsPerRow(), setStandardRows(), setPremiumRows(), setReclinerRows(), setVipRows(), and setLayoutMap() allow the admin settings module to update auditorium configuration.
- **Layout-related accessors** - support the row and seat grid used by the booking page, where the seating plan is treated like a structured two-dimensional layout for rendering and category mapping.

4.2.2 DAO Classes

The DAO layer is responsible for SQL execution and persistence. It separates database code from servlets, allowing controllers to request application data without directly managing SQL statements. DAO methods also convert database result sets into Java model objects and return collections such as ArrayList where multiple records are needed.

UserDao: Performs registration, email and ID lookup, profile update, password update, OTP storage, OTP verification, and admin user listing.

Method explanation:

- **registerUser(User user)** - inserts a new user record after servlet-level validation has confirmed that the submitted account details are acceptable.
- **getUserByEmail(String email)** - retrieves a user by email address for login verification, duplicate account checking, and password reset workflows.
- **getUserById(int userId)** - loads a specific user record for profile display or administrative review.
- **updateUser(User user)** - updates editable profile fields such as full name and phone number while keeping the user identifier consistent.
- **updatePassword(...)** - stores the new hashed password after a successful password reset or account update.
- **storeOtp(...) and verifyOtp(...)** - support the forgot-password process by saving the reset code, checking the submitted code, and validating expiry time.
- **getAllUsers()** - returns an ArrayList of users for the admin user-management page.

MovieDao: Performs movie creation, update, search, filtering, detail retrieval, active movie loading, and soft-delete operations.

Method explanation:

- **addMovie(Movie movie)** - inserts a new movie record, including metadata, release dates, poster path, trailer URL, and optional price overrides.
- **updateMovie(Movie movie)** - updates existing catalogue information without changing the primary movie identifier used by related records.
- **getMovieById(int movieId)** - retrieves a single movie for the movie details page, edit form, and booking workflow.
- **getActiveMovies()** - loads only movies marked as active so hidden or soft-deleted records do not appear to customers.
- **searchMovies(...) and filterMovies(...)** - return ArrayList-based movie results according to keyword, genre, language, date, or availability conditions.

- **softDeleteMovie(int movieId)** - sets the active state to false rather than permanently deleting the record, preserving historical booking and revenue integrity.

BookingDao: Creates booking records, retrieves customer booking history, loads admin booking lists, checks booked seats, updates status, and calculates dashboard revenue summaries.

Method explanation:

- **createBooking(Booking booking)** - inserts a confirmed ticket transaction with user, movie, seats, show-time description, total price, status, and booking date.
- **getBookingsByUserId(int userId)** - returns an ArrayList of bookings for the My Bookings page.
- **getAllBookings()** - loads booking records for administrative monitoring and status management.
- **getBookedSeats(...)** - retrieves already-reserved seat identifiers for a selected movie, date, time, and hall so the booking page can block unavailable seats.
- **updateBookingStatus(...)** - changes the booking state, for example from Confirmed to Cancelled, while keeping the transaction record for audit purposes.
- **getTotalRevenue(), getRecentBookings(), and getTopBookedMovies()** - execute aggregation queries and return totals, lists, or HashMap-style chart data for dashboard analytics.

ShowTimeDao: Adds and retrieves show-time records, removes old schedules when required, and checks hall/time conflicts before a schedule is saved.

Method explanation:

- **addShowTime(ShowTime showTime)** - stores a new screening slot after the admin schedule builder has collected the date, time, movie, and hall.
- **getShowTimesByMovieId(int movieId)** - loads available showtimes for a selected movie so customers can choose an appropriate screening.

- **deleteShowTimesByMovieId(int movieId)** - removes or refreshes old schedule entries when an administrator updates the movie schedule.
- **hasScheduleConflict(...)** - checks whether the selected hall is already occupied by another movie and applies the required cleaning buffer before accepting a new showtime.

HallConfigDao: Loads and updates hall layout rules, row category mappings, and seat-map configuration records.

Method explanation:

- **getHallConfig(String hallName)** - loads the auditorium layout required to construct the dynamic seat map.
- **getAllHallConfigs()** - retrieves all available hall layouts for the admin settings page.
- **updateHallConfig(HallConfig config)** - saves changes to seat rows, category mappings, and layout data.

GlobalSettingsDao: Loads and updates shared application settings such as default ticket prices and available hall names.

Method explanation:

- **getSetting(String key)** - retrieves a single system setting such as a default ticket price.
- **updateSetting(String key, String value)** - saves a modified setting from the admin settings page.
- **getTicketPrices()** - loads the default Standard, Premium, Recliner, and VIP price values used when a movie does not provide its own price overrides.

DBConnection: Creates JDBC connections to the MySQL database using environment configuration supplied by EnvLoader, with local fallback support for development.

Method explanation:

- **getConnection()** - opens a JDBC connection using the configured database URL, username, and password.
- **Connection handling logic** - centralises database connectivity so DAO classes do not duplicate driver-loading and connection-string code.

4.2.3 Controller Servlets

Controller servlets receive HTTP requests, validate submitted data, call DAO or service classes, place results into request or session scope, and forward users to the correct JSP view. This keeps request-handling logic separate from presentation code. In most controller classes, doGet() is used to prepare and display a page, while doPost() processes submitted form data and performs state-changing operations.

LoginServlet: Validates credentials, creates authenticated sessions, stores role information, and redirects users to the correct dashboard.

Method explanation:

- **doGet()** - forwards unauthenticated users to the login view.
- **doPost()** - reads email and password values, calls UserDao for account lookup, verifies the password, stores user details in the session, and redirects based on role.

RegisterServlet: Validates account details, checks duplicate users, hashes the password, and stores a new user record.

Method explanation:

- **doGet()** - displays the registration form.
- **doPost()** - validates name, email, phone, password, and confirmation fields; checks for duplicates through UserDao; hashes the password; and inserts the new account.

ForgotPasswordServlet: Coordinates email submission, OTP generation, OTP verification, expiry checking, and password reset.

Method explanation:

- **doGet()** - loads the password reset page and displays the correct reset stage.
- **doPost()** - handles the reset workflow by sending OTP emails, verifying submitted OTP values, checking expiry time, and updating the password after successful verification.

LogoutServlet: Invalidates the session and redirects the user to the login page.

Method explanation:

- **doGet()** - removes the active session and prevents the user from continuing to access protected pages after logout.

UserHomeServlet, BrowseMoviesServlet, and MovieDetailsServlet: Load customer-facing movie data, filtering information, detailed movie metadata, and available show times.

Method explanation:

- **doGet() in UserHomeServlet** - loads recently added or currently available movies for the customer home page.
- **doGet() in BrowseMoviesServlet** - applies search, genre, language, date, and availability filters before forwarding movie lists to the browse page.
- **doGet() in MovieDetailsServlet** - loads full movie information and linked showtimes for the selected movie.

BookTicketServlet: Builds the booking workflow by preparing hall-layout JSON, booked-seat JSON, seat categories, dynamic price calculation, and booking confirmation logic.

Method explanation:

- **doGet()** - loads the selected movie, showtime, hall configuration, ticket prices, and currently booked seats, then forwards this data to the booking JSP.

- **doPost()** - validates selected seats, checks that the seats are still available, calculates the total using four-tier pricing, creates the booking, and forwards the user to the confirmation workflow.
- **Seat-map preparation logic** - uses hall configuration and booked-seat collections to generate a dynamic two-dimensional seating layout on the frontend.

TicketConfirmationServlet, MyBookingsServlet, and UserProfileServlet: Support confirmed ticket display, customer booking history, and profile update workflows.

Method explanation:

- **TicketConfirmationServlet doGet()** - loads the latest confirmed booking and displays ticket information such as movie, time, hall, seats, and total price.
- **MyBookingsServlet doGet()** - loads the current user booking history through BookingDao and forwards an ArrayList of booking records to the JSP.
- **UserProfileServlet doGet() and doPost()** - display user details and process validated profile updates.

AdminHomeServlet: Prepares dashboard statistics, recent bookings, revenue totals, movie performance data, and chart-ready values.

Method explanation:

- **doGet()** - loads totals for users, movies, bookings, and revenue, retrieves recent booking activity, and prepares chart labels and values using DAO aggregation results.
- **Analytics preparation logic** - uses ArrayList and HashMap-style data structures to pass grouped revenue, movie performance, and seat-category data to Chart.js on the JSP page.

ManageMoviesServlet and TmdbSearchServlet: Support movie catalogue management, TMDB-assisted metadata entry, show-time scheduling, date validation, and trailer/poster support.

Method explanation:

- **ManageMoviesServlet doGet()** - loads existing movie records, filter data, and form state for the admin catalogue page.
- **ManageMoviesServlet doPost()** - handles add, update, soft-delete, pricing override, and schedule builder operations after validation.
- **Scheduling validation logic** - checks movie duration, date range, selected hall, and the 30-minute cleaning buffer before accepting showtimes.
- **TmdbSearchServlet request method** - sends the movie search term to TMDB, parses JSON metadata, and returns auto-fill data such as title, description, poster path, release date, cast, and trailer information.

ManageUsersServlet, ManageBookingsServlet, and ManageSettingsServlet: Support administration of users, bookings, ticket prices, and hall layout configuration.

Method explanation:

- **ManageUsersServlet doGet() and doPost()** - load user records and process administrative changes such as role updates or account state management.
- **ManageBookingsServlet doGet() and doPost()** - load booking records, process booking status changes, and support administrative booking monitoring.
- **ManageSettingsServlet doGet() and doPost()** - load current global settings and save updates to ticket pricing, available halls, and hall layout configuration.

Utility and Initialization Servlets: Provide supporting endpoints for setup workflows, auxiliary data access, or application initialization while keeping the main MVC routes focused on user-facing tasks.

Method explanation:

- **Initialization methods** - prepare application defaults, verify required settings, or expose limited setup functionality during development and deployment.
- **Utility request handlers** - support small background workflows without overloading the main customer and admin controllers.

AboutServlet and ContactServlet: Serve informational pages for system context and customer communication.

Method explanation:

- **doGet()** - routes users through the servlet layer to the correct JSP page, maintaining the MVC pattern instead of exposing JSP files directly.
- **doPost() where applicable** - processes submitted contact information after validation and forwards the result message back to the view.

4.2.4 Security, Service, and Utility Classes

The supporting classes strengthen security, configuration management, email delivery, validation, and ticket generation. They reduce duplication by placing reusable logic outside the servlet controllers.

AuthFilter: Protects user and administrator routes by checking session state and role authorization before protected resources are reached.

Method explanation:

- **doFilter(ServletRequest request, ServletResponse response, FilterChain chain)** - intercepts protected requests, checks session validity, verifies the authenticated role, allows authorised requests to continue, and redirects unauthorised users.

EmailService: Sends password-reset OTP messages and booking confirmation emails through Gmail SMTP with TLS configuration and attachment support where applicable.

Method explanation:

- **sendOtpEmail(String recipient, String otp)** - builds and sends the password reset OTP message through Gmail SMTP.
- **sendBookingConfirmationEmail(...)** - sends booking confirmation details to the customer after a successful ticket reservation.
- **SMTP configuration logic** - sets host, port, TLS, authentication, sender credentials, subject, body, and optional attachment information.

EnvLoader: Reads database, email, and TMDB configuration from a .env file or environment variables so credentials are not hard-coded into the Java source code.

Method explanation:

- **loadEnv()** - loads configuration values at application startup or when configuration is first requested.
- **get(String key)** - retrieves a required configuration value such as a database URL, TMDB key, or SMTP username.
- **getOrDefault(String key, String defaultValue)** - returns a fallback value when a non-critical configuration key is absent.
- **OtpUtil:** Generates six-digit OTP values for password reset verification.
- **Method explanation:**
- **generateOtp()** - creates a random numeric reset code that can be stored with an expiry timestamp and emailed to the user.

PasswordUtil: Hashes and verifies passwords to avoid normal plain-text credential handling.

Method explanation:

- **hashPassword(String password)** - converts a raw password into a stored hashed value before database insertion.
- **verifyPassword(String rawPassword, String storedHash)** - compares a submitted password with the stored hash during login or password-sensitive workflows.

Validation: Provides reusable checks for email, phone number, password, date, and general form validity.

Method explanation:

- **isValidEmail(String email)** - checks that submitted email addresses follow an acceptable format.

- **isValidPhone(String phone)** - ensures that phone input matches the expected numeric length and structure.
- **isValidPassword(String password)** - checks password length and strength rules before registration or reset.
- **isValidDate(String date)** - supports strict date validation for movie and showtime scheduling.
- **sanitizeInput(String input)** - reduces the risk of unsafe or malformed text being processed by the system.

HtmlToPdfConverter: Converts ticket-style HTML into a PDF document for digital ticket evidence and email attachment workflows.

Method explanation:

- **convertHtmlToPdf(...)** - takes ticket-style HTML content and creates a PDF version of the ticket for download or email use.

GenerateAdminPassword: Supports creation of a hashed default administrator password during setup or maintenance.

Method explanation:

- **main(String[] args) or setup helper method** - generates a secure hash that can be used when preparing an initial administrator account.

4.3 MVC Responsibilities

- **Model:** Stores application data in plain Java objects. Actual model classes include User, Movie, Booking, ShowTime, and HallConfig.
- **DAO:** Executes SQL and converts database rows into model objects. Actual DAO classes include UserDao, MovieDao, BookingDao, ShowTimeDao, HallConfigDao, and GlobalSettingsDao.
- **Controller:** Receives browser requests, validates inputs, calls DAOs or services, and forwards to JSP views. Examples include LoginServlet, BookTicketServlet, ManageMoviesServlet, AdminHomeServlet, and TmdbSearchServlet.
- **Filter:** Checks session and role before protected pages are reached. AuthFilter protects user and administrator routes listed in the repository.
- **View:** Displays data and forms to customers and administrators. JSP pages are stored under WEB-INF/pages and are supported by CSS, JavaScript, and Chart.js assets under the webapp folder.
- **Utility and service layer:** Provides reusable helper functions and external support. Key classes include EnvLoader, PasswordUtil, OtpUtil, Validation, HtmlToPdfConverter, GenerateAdminPassword, and EmailService.

5. Test Cases

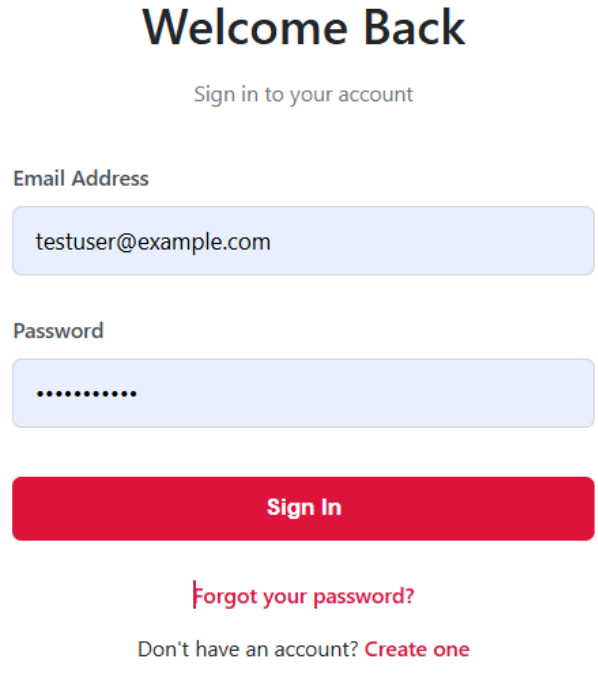
Testing was carried out to evaluate MovieMint under normal, boundary, exceptional, integration, security-sensitive, and concurrency-related conditions. The following test cases focus on the most technically significant behaviour of the Java Servlet MVC system, including dynamic seat allocation, four-tier pricing, scheduling conflict detection, API integration, database persistence, role-based access control, and responsive user interface behaviour.

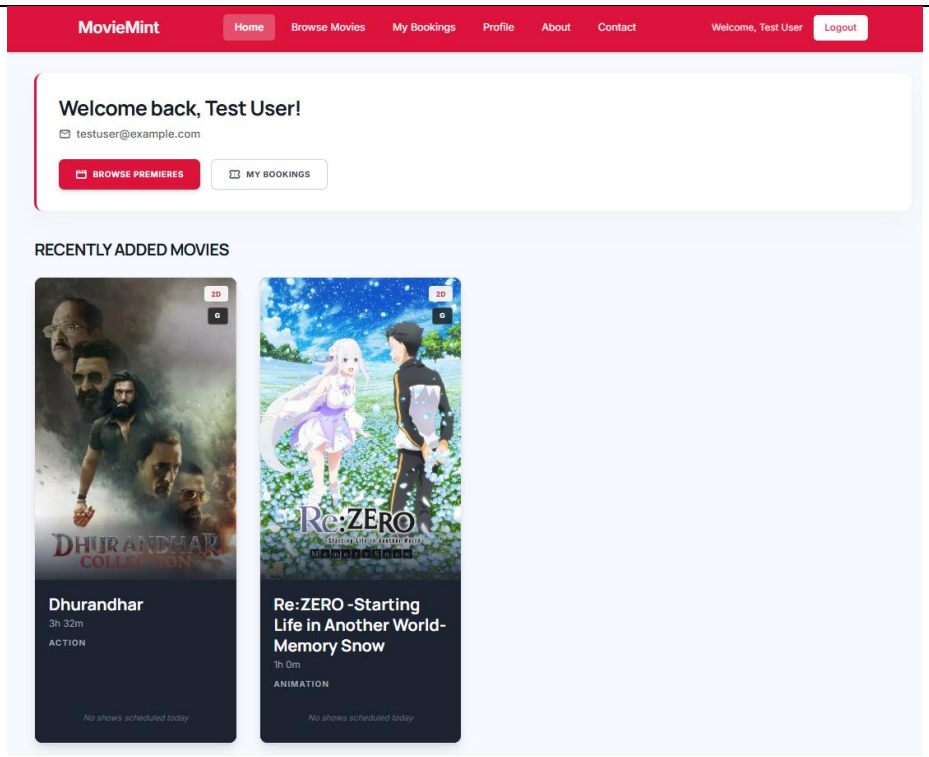
Each test case uses a consistent two-column structure so that the evidence screenshot can be inserted directly into the Actual Outcome row after practical execution.

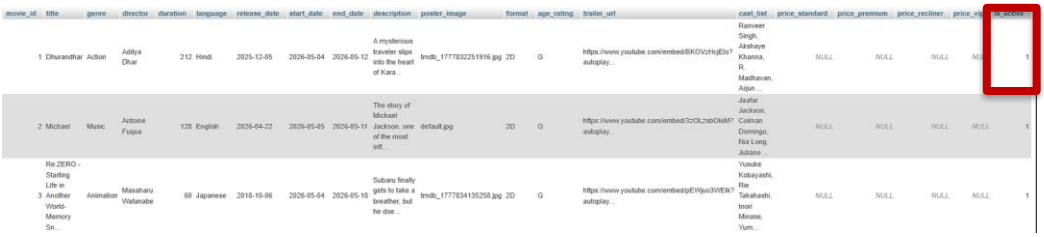
5.1 CRUD & State Management Testing

The CRUD and state management checks in Tables 1 to 5 verify that MovieMint preserves data integrity when records are created, updated, hidden, cancelled, or reclassified. These tests focus on state transitions rather than simple form submission.

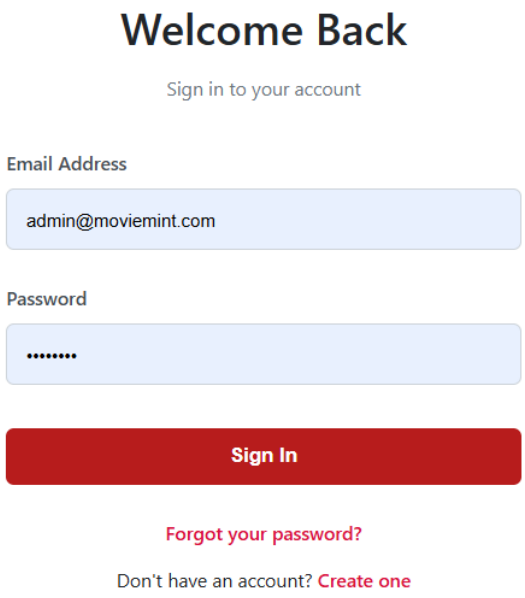
Table 1: CRUD Test - Movie Soft Delete Integrity

Field	Description
Test Name	CRUD Test - Movie Soft Delete Integrity
Given Input	Admin selects an active movie and chooses the hide option from the movie management page.
Expected Outcome	The movie record should not be permanently removed from the database. The is_active field should be updated to 0 so that the movie no longer appears in public listings while historical booking, revenue, and reporting records remain intact.
Actual Outcome	





movie_id	title	genre	director	duration	language	release_date	start_date	end_date	description	poster_image	format	age_rating	trailer_url	cast_list	price_standard	price_premium	price_recliner	price_vip	is_active
1	Dhurandhar	Action	Aditya Dhar	212	Hindi	2025-12-05	2026-05-04	2026-05-12	A mysterious traveler slips into the heart of Kara...	tmdb_1777832251916.jpg	2D	G	https://www.youtube.com/embed/BKOVzHcjEIo?autoplay...	Ranveer Singh, Akshaye Khanna, R. Madhavan, Arjun ...	NULL	NULL	NULL	NULL	1
2	Michael	Music	Antoine Fuqua	128	English	2026-04-22	2026-05-05	2026-05-11	The story of Michael Jackson, one of the most infl...	default.jpg	2D	G	https://www.youtube.com/embed/3zOLzsbOleM?autoplay...	Jaafar Jackson, Colman Domingo, Nia Long, Juliano ...	NULL	NULL	NULL	NULL	1
3	Re:ZERO -Starting Life in Another World- Memory Sn...	Animation	Masaharu Watanabe	60	Japanese	2018-10-06	2026-05-04	2026-05-18	Subaru finally gets to take a breather, but he doe...	tmdb_1777834135258.jpg	2D	G	https://www.youtube.com/embed/pEWjuo3WEtk?autoplay...	Yusuke Kobayashi, Rie Takahashi, Inori Minase, Yum...	NULL	NULL	NULL	NULL	1



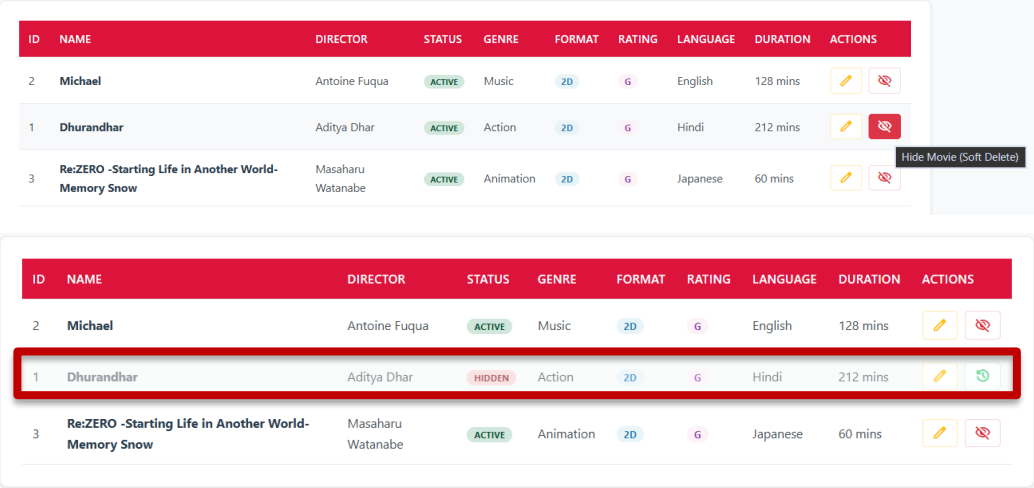
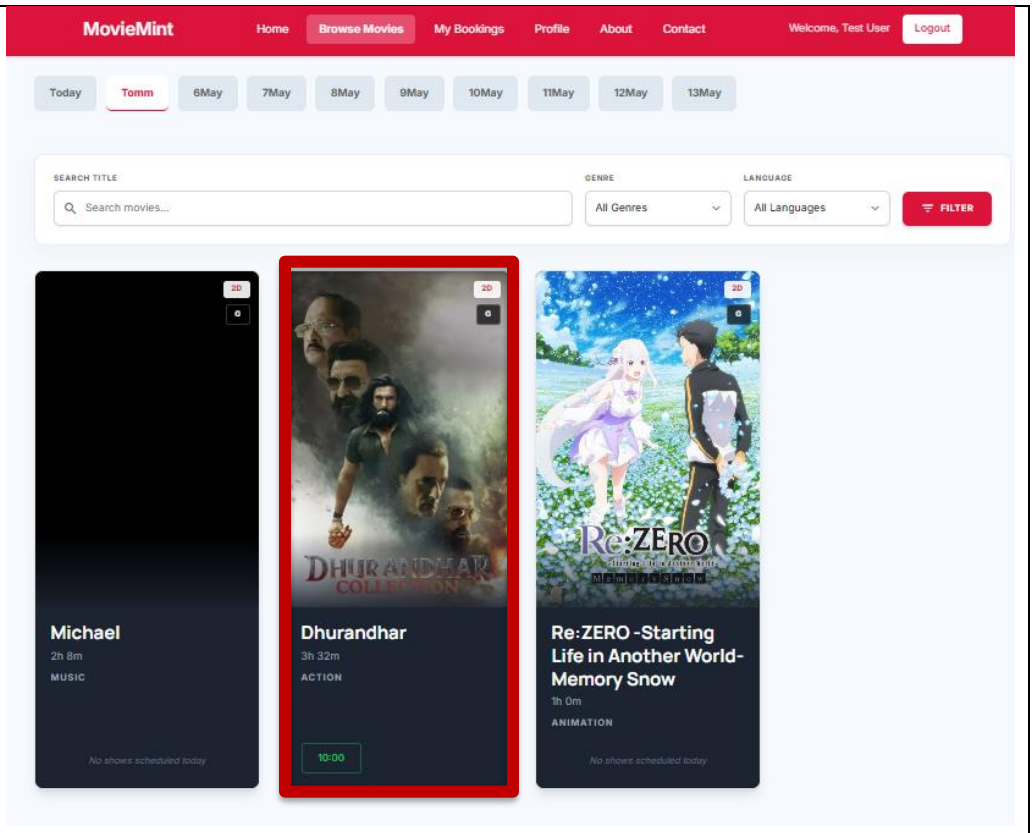
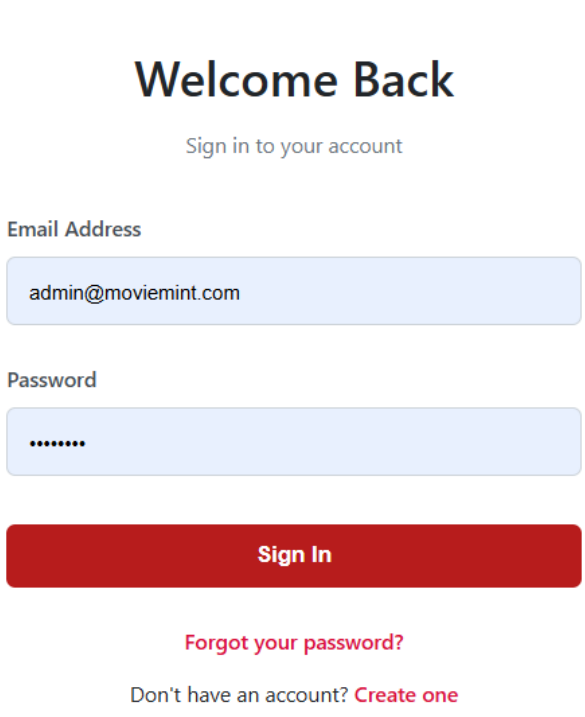
	
Test Result	Passed

Table 2: CRUD Test - Movie Record Update with Existing Showtimes

Field	Description
Test Name	CRUD Test - Movie Record Update with Existing Showtimes
Given Input	Admin updates a movie title, genre, language, trailer URL, cast list, format, and description while existing showtimes are already linked to that movie.
Expected Outcome	The updated movie details should be saved successfully without deleting or corrupting existing show_times records linked through movie_id. Public movie details and admin catalogue data should reflect the revised information.

Actual Outcome	 

Edit Movie

Current Poster:

Update Poster Image (Optional):

Choose File No file chosen

Upload JPG, PNG, or GIF to replace current image

Title:

Dhurandhar

Genre:

Action

Director:

Aditya Dhar

Duration (minutes):

212

Language:

Hindi

Edit Show Schedule:

Select dates to manage show times

Mon, May 4 | Tue, May 5 | Wed, May 6 | Thu, May 7 | Fri, May 8 | Sat, May 9 | Sun, May 10 | Mon, May 11 | Tue, May 12

Select Halls:

Audi 01 | Audi 02 | Audi 03

Audi 01

--:-- --

Add Time

10:00 AM ×

Update Movie | Cancel

Edit Movie

Current Poster:

Update Poster Image (Optional):

Choose File No file chosen

Upload JPG, PNG, or GIF to replace current image

Title:

TOY SOTRY

Genre:

FANTASY

Director:

Nayan Chhantyal

Duration (minutes):

212

Language:

Nepali

Movie Format:

2D

Edit Show Schedule:

Select dates to manage show times

Mon, May 4 | Tue, May 5 | Wed, May 6 | Thu, May 7 | Fri, May 8 | Sat, May 9 | Sun, May 10 | Mon, May 11 | Tue, May 12

Select Halls:

Audi 01 | Audi 02 | Audi 03

Audi 01

--:-- -- Add Time

10:00 AM ×

Update Movie Cancel

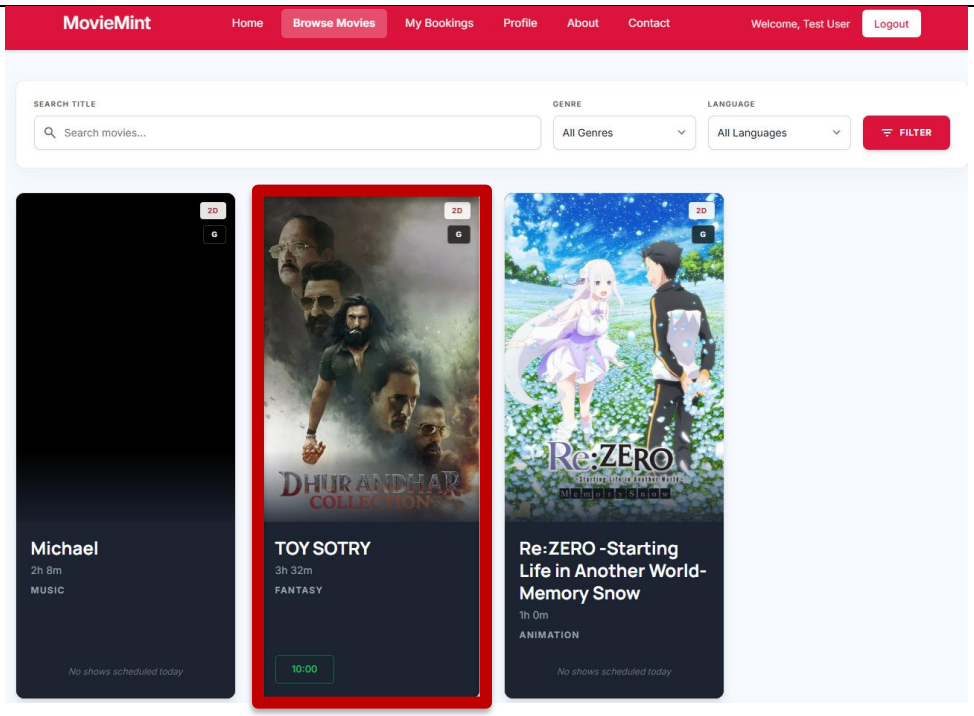
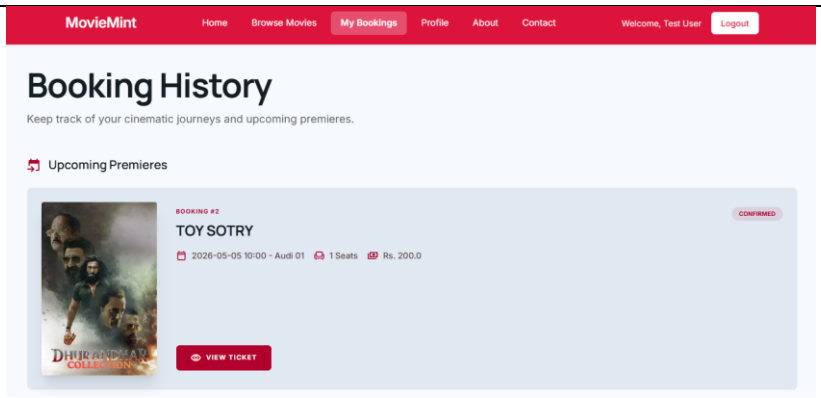
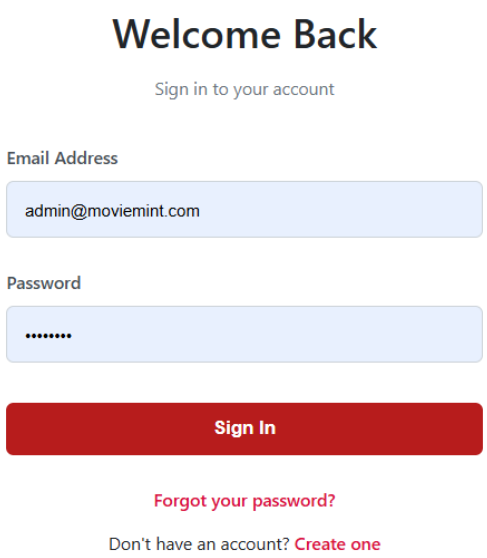
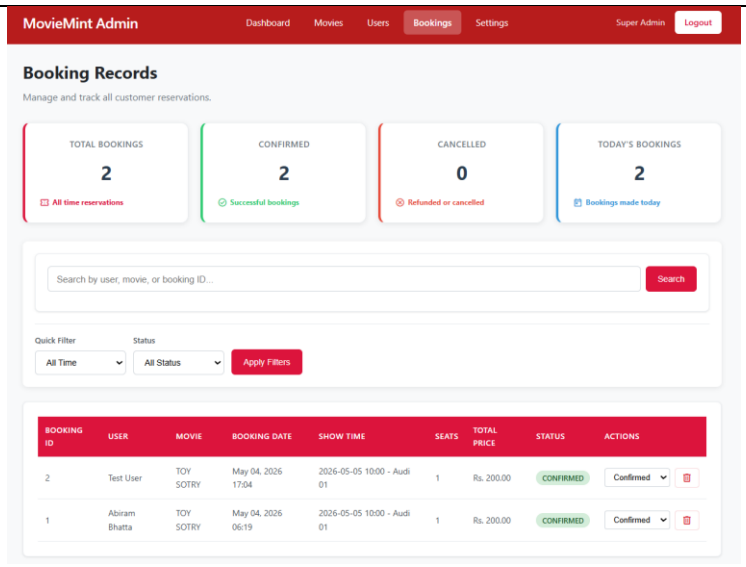
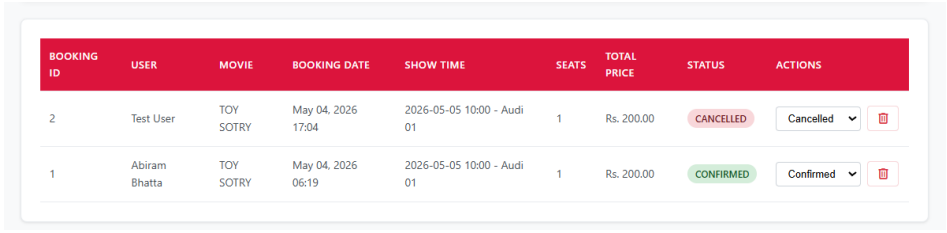
	
Test Result	Passed

Table 3: CRUD Test - Booking Status Transition

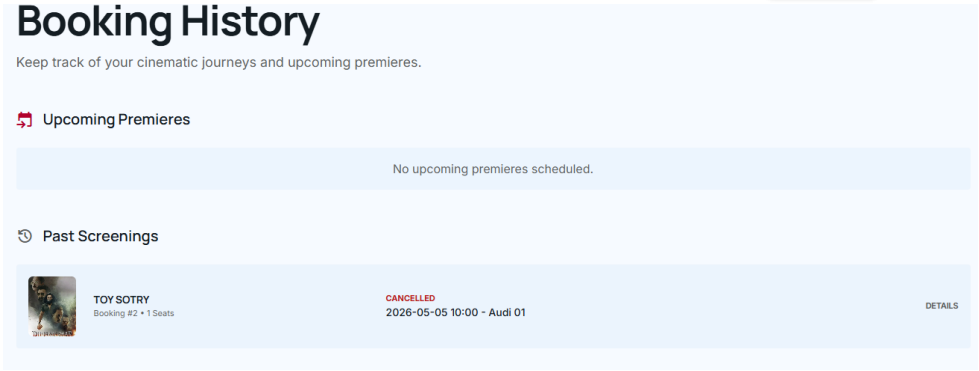
Field	Description
Test Name	CRUD Test - Booking Status Transition
Given Input	Admin changes a booking status from Confirmed to Cancelled from the manage bookings page.
Expected Outcome	The booking status should update in the database without deleting the transaction. Cancelled bookings should remain available for audit purposes but should not be treated as active confirmed reservations in operational views.

Actual Outcome	 booking_id | user_id | movie_id | show_time | number_of_seats | seat_type | total_price | status | booking_date 1 | 3 | 1 | 2026-05-05 10:00 - Audi 01 | 1 | E6 | 200 | Confirmed | 2026-05-04 00:34:36 2 | 5 | 1 | 2026-05-05 10:00 - Audi 01 | 1 | E7 | 200 | Confirmed | 2026-05-04 11:19:42 



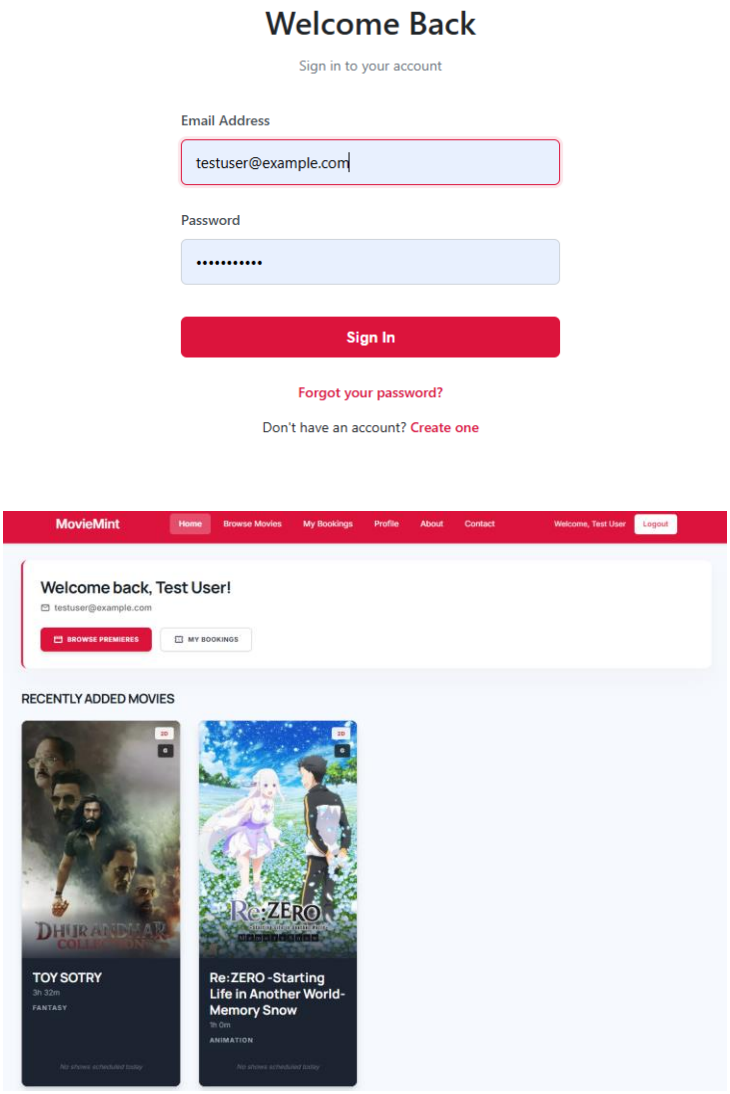


booking_id	user_id	movie_id	show_time	number_of_seats	seat_type	total_price	status	booking_date
1	3	1	2026-05-05 10:00 - Audi 01	1	E6	200	Confirmed	2026-05-04 00:34:36
2	5	1	2026-05-05 10:00 - Audi 01	1	E7	20	Cancelled	026-05-04 11:19:42

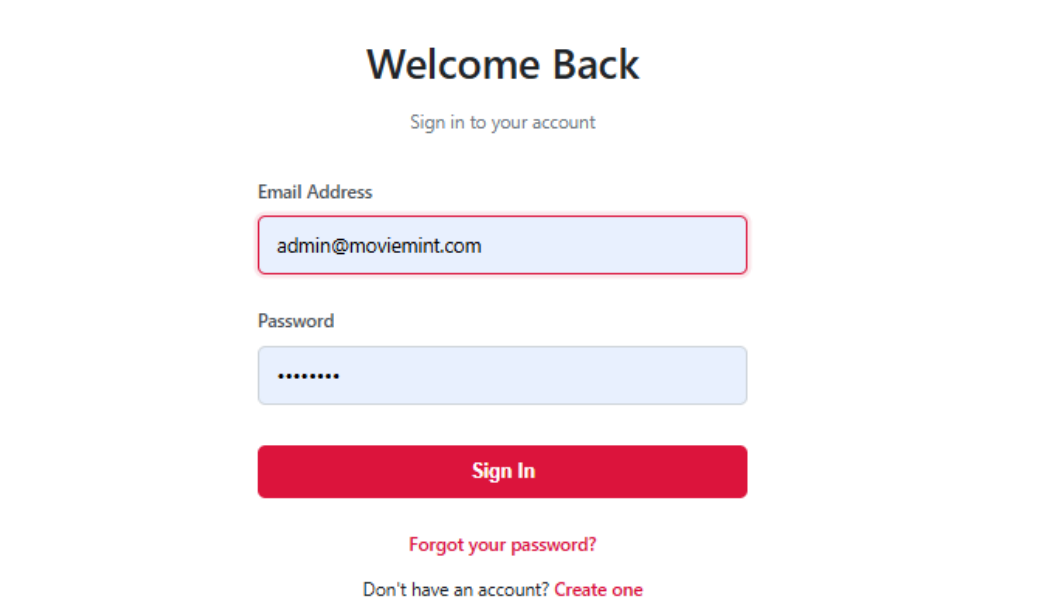


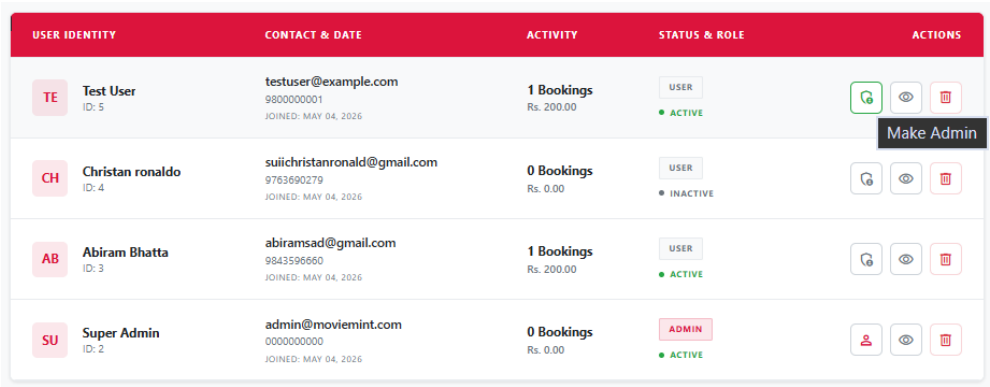
Test Result	Passed

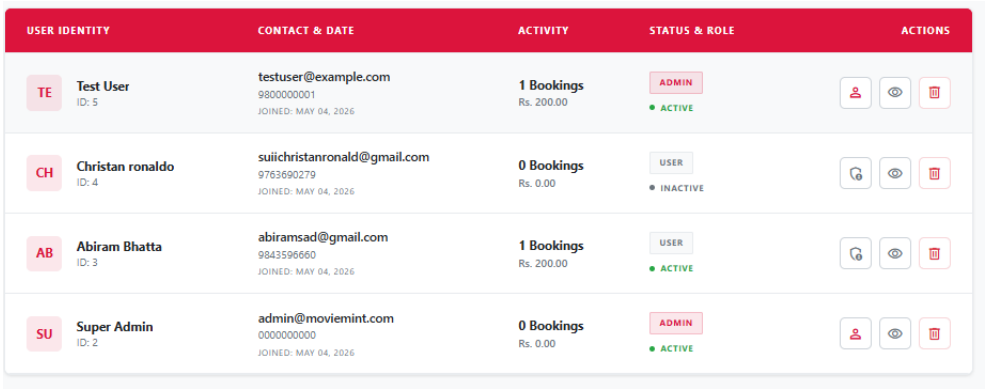
Table 4: CRUD Test - User Role Update State Persistence

Field	Description
Test Name	CRUD Test - User Role Update State Persistence
Given Input	Admin changes a registered account role from user to admin.
Expected Outcome	The new role should persist in the users table. On the next successful login, the account should be redirected according to the updated role and should only receive admin privileges after authentication has been completed.
Actual Outcome	

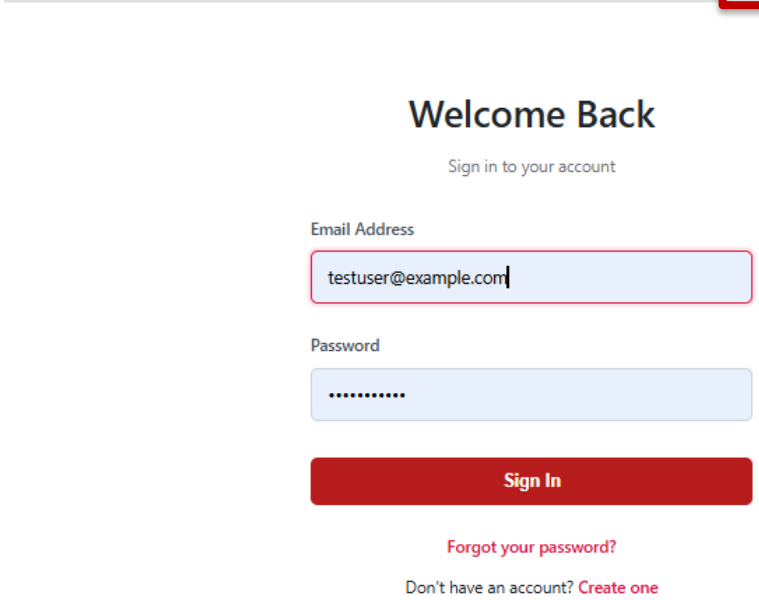
user_id	full_name	email	phone	password	role	created_at	reset_otp	otp_expiry
2	Super Admin	admin@moviemint.com	0000000000	JtFfxZi56vzSdIvoB6nfZa0sWo46GfguqvVTxklhEzrzF0yfh2...	admin	2026-05-03 22:18:26	NULL	NULL
3	Abiram Bhatta	abiramsad@gmail.com	9843596660	0HrjQPjFmlSiyGZx0sJd4xAGPlW6lRshfGV0eZvEr4c1YDUEmd...	user	2026-05-04 00:08:47	NULL	NULL
4	Christan ronaldo	suiichristanronald@gmail.com	9763690279	azSf3abJflJ2knX8tEl33vl1ZOJwo+S1J0CIrre8E4wC5nOXXu...	user	2026-05-04 00:38:47	NULL	NULL
5	Test User	testuser@example.com	9800000001	/NlGwyFswo2vKOpJqZwZLUOp1I7dTCAogNgvQL49W83zQy7T...	user	2026-05-04 09:36:58	NULL	NULL

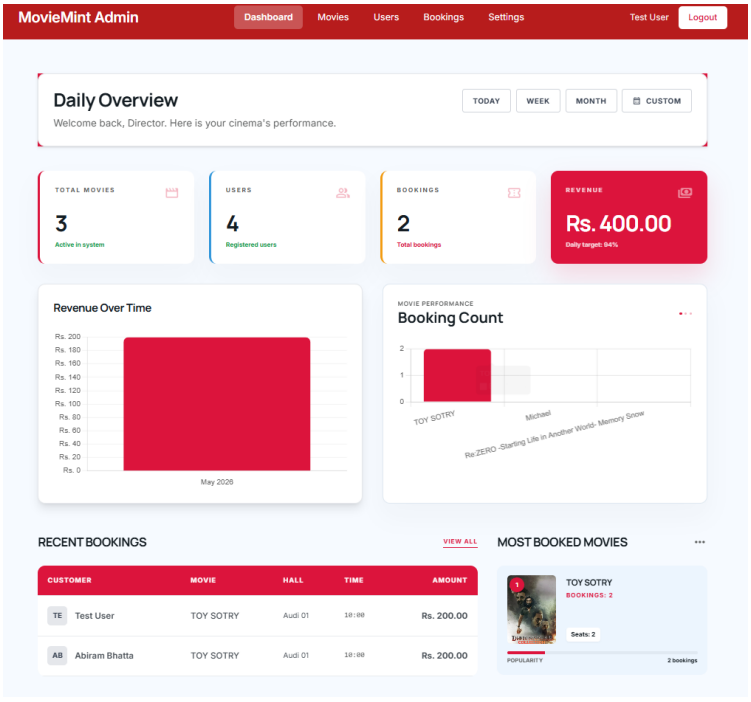






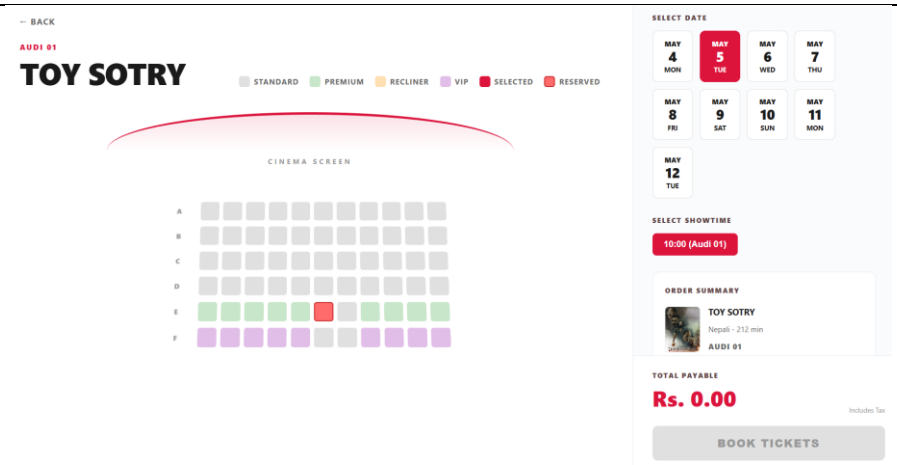
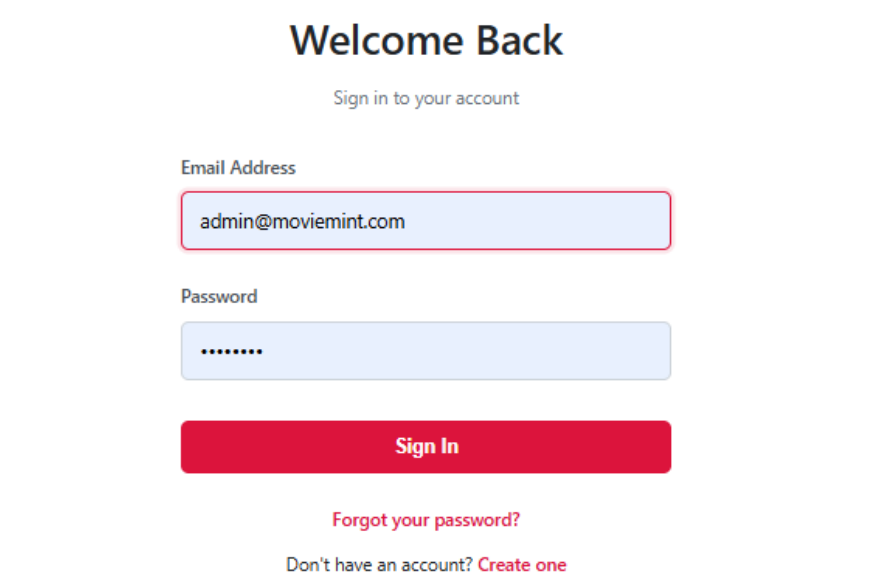
user_id	full_name	email	phone	password	role	created_at	reset_otp	otp_expiry
2	Super Admin	admin@moviemint.com	0000000000	JtFfxZi56vzSdlvoB6nfZa0sWo46GfguqvVTxklhEzrzF0yfh2...	admin	2026-05-03 22:18:26	NULL	NULL
3	Abiram Bhatta	abiramsad@gmail.com	9843596660	0HrjQPjFmlSiyGZx0sJd4xAGPlW6lRshtGV0eZvEr4c1YDUEmd...	user	2026-05-04 00:08:47	NULL	NULL
4	Christan ronaldo	suiichristanronald@gmail.com	9763690279	azSf3abJfJ2knX8tEl33vl1ZOJwo+S1J0Clrre8E4wC5nOXXu...	user	2026-05-04 00:38:47	NULL	NULL
5	Test User	testuser@example.com	9800000001	/NlGwyFswo2vKOpJqZwZLUOp1l7dTCAogNgvQL49W83zQy7T...	admin	2026-05-04 09:36:58	NULL	NULL

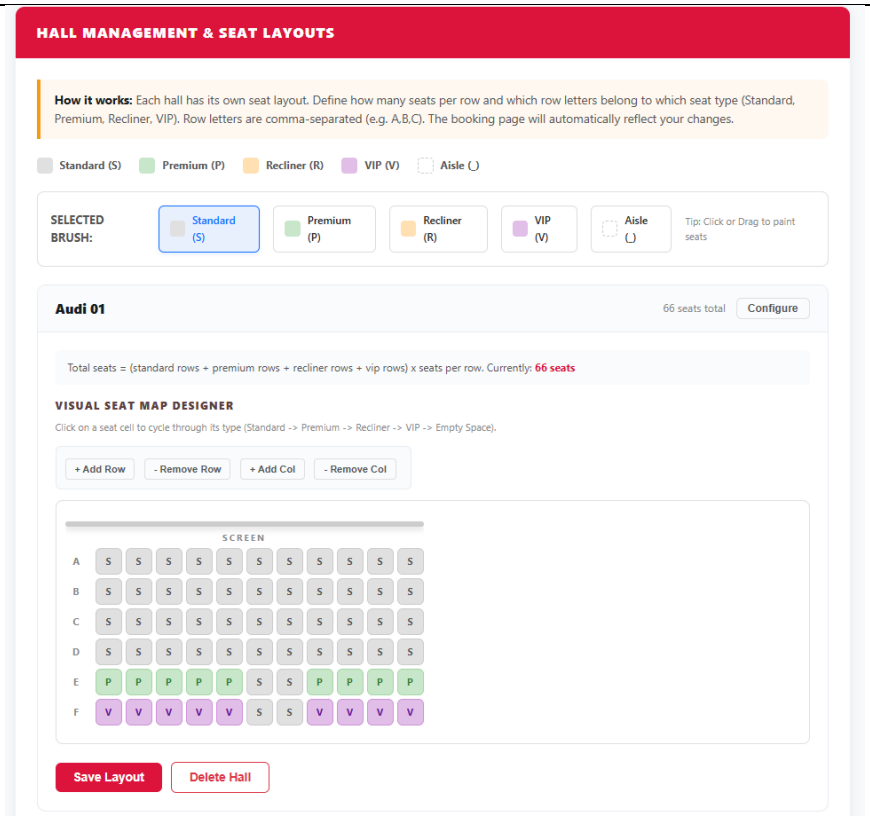


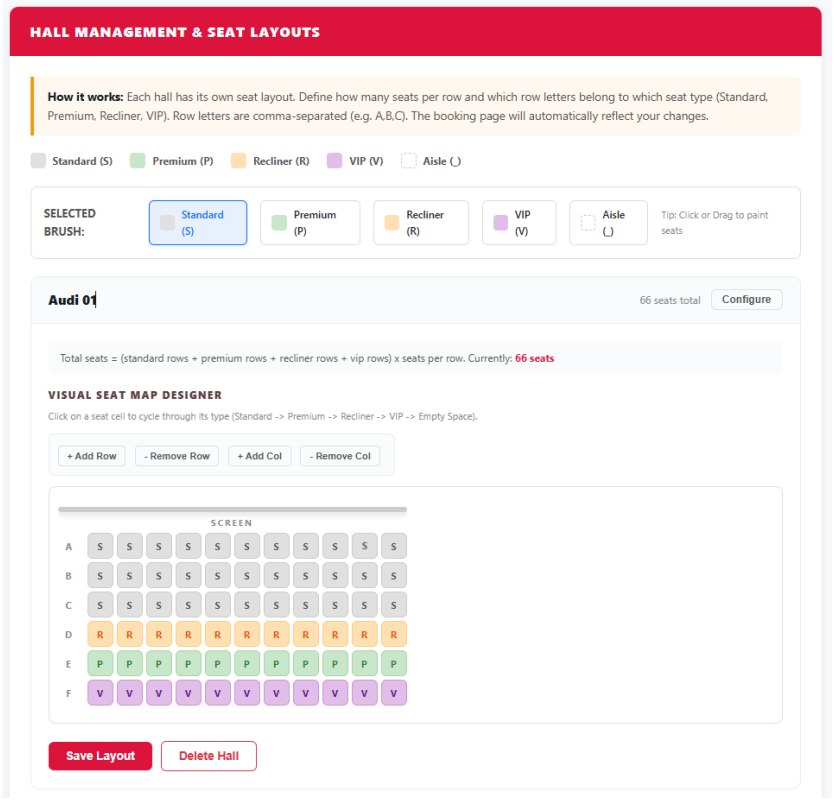


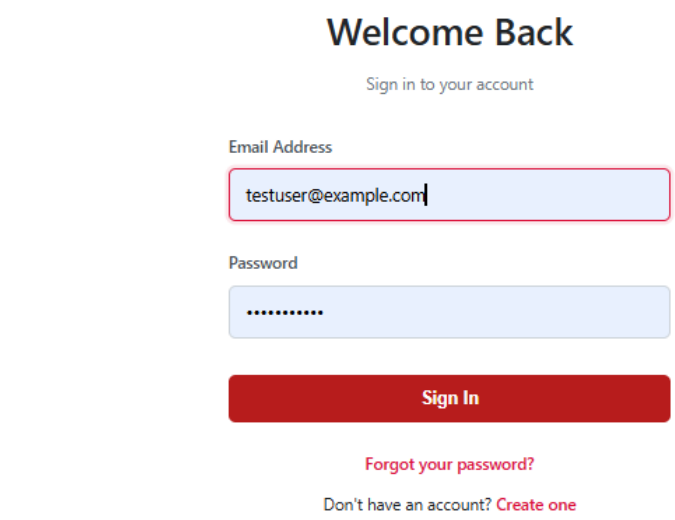
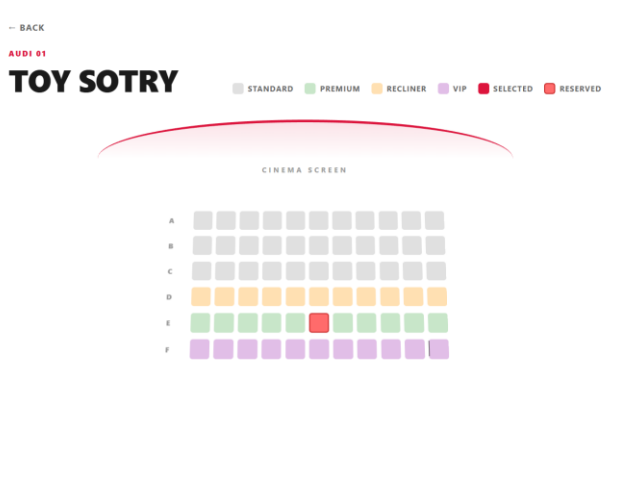
Test Result	Passed

Table 5: CRUD Test - Hall Configuration Update Persistence

Field	Description
Test Name	CRUD Test - Hall Configuration Update Persistence
Given Input	Admin updates hall configuration values including seats per row, standard rows, premium rows, recliner rows, VIP rows, and the visual layout map.
Expected Outcome	The revised hall configuration should be saved in the hall_config table and used by the booking servlet when generating future dynamic seat maps for the relevant hall.
Actual Outcome	 


HALL MANAGEMENT & SEAT LAYOUTS
How it works: Each hall has its own seat layout. Define how many seats per row and which row letters belong to which seat type (Standard, Premium, Recliner, VIP). Row letters are comma-separated (e.g. A,B,C). The booking page will automatically reflect your changes.
Standard (S)
Premium (P)
Recliner (R)
VIP (V)
Aisle (_)
SELECTED BRUSH:
Standard (S)
Premium (P)
Recliner (R)
VIP (V)
Aisle (_)
Tip: Click or Drag to paint seats
Audi 01
66 seats total
Configure
Total seats = (standard rows + premium rows + recliner rows + vip rows) x seats per row. Currently: 66 seats
VISUAL SEAT MAP DESIGNER
Click on a seat cell to cycle through its type (Standard -> Premium -> Recliner -> VIP -> Empty Space).
+ Add Row
- Remove Row
+ Add Col
- Remove Col
SCREEN
A S S S S S S S S S S S
B S S S S S S S S S S S
C S S S S S S S S S S S
D S S S S S S S S S S S
E P P P P P S S P P P P
F V V V V V S S V V V V
Save Layout
Delete Hall


HALL MANAGEMENT & SEAT LAYOUTS
How it works: Each hall has its own seat layout. Define how many seats per row and which row letters belong to which seat type (Standard, Premium, Recliner, VIP). Row letters are comma-separated (e.g. A,B,C). The booking page will automatically reflect your changes.
Standard (S)
Premium (P)
Recliner (R)
VIP (V)
Aisle (_)
SELECTED BRUSH:
Standard (S)
Premium (P)
Recliner (R)
VIP (V)
Aisle (_)
Tip: Click or Drag to paint seats
Audi 01
66 seats total
Configure
Total seats = (standard rows + premium rows + recliner rows + vip rows) x seats per row. Currently: 66 seats
VISUAL SEAT MAP DESIGNER
Click on a seat cell to cycle through its type (Standard -> Premium -> Recliner -> VIP -> Empty Space).
+ Add Row
- Remove Row
+ Add Col
- Remove Col
SCREEN
A S S S S S S S S S S S
B S S S S S S S S S S S
C S S S S S S S S S S S
D R R R R R R R R R R R
E P P P P P P P P P P P
F V V V V V V V V V V V
Save Layout
Delete Hall

	 
Test Result	Passed

5.2 Core Functional & Business Logic Testing

Tables 6 to 13 test the core business rules of the system. These cases demonstrate that MovieMint is not limited to basic CRUD operations; it also performs dynamic pricing, schedule conflict detection, concurrent booking protection, and real-time dashboard calculation.

Table 6: Functional Test - Dynamic Four-Tier Seat Pricing Calculation

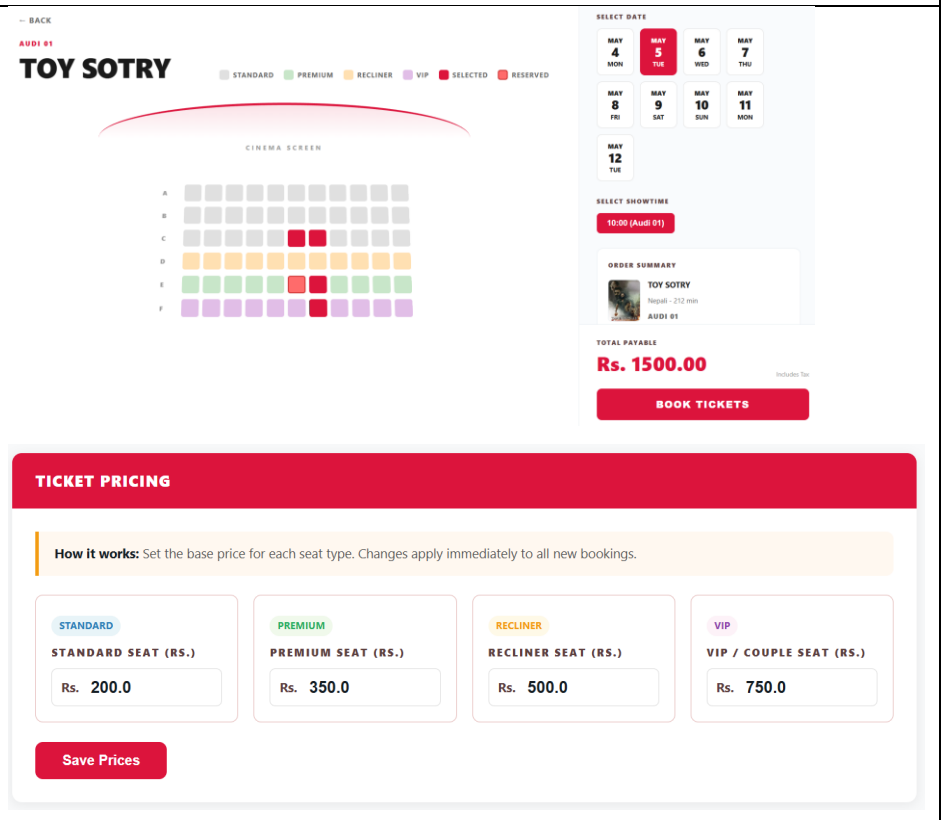
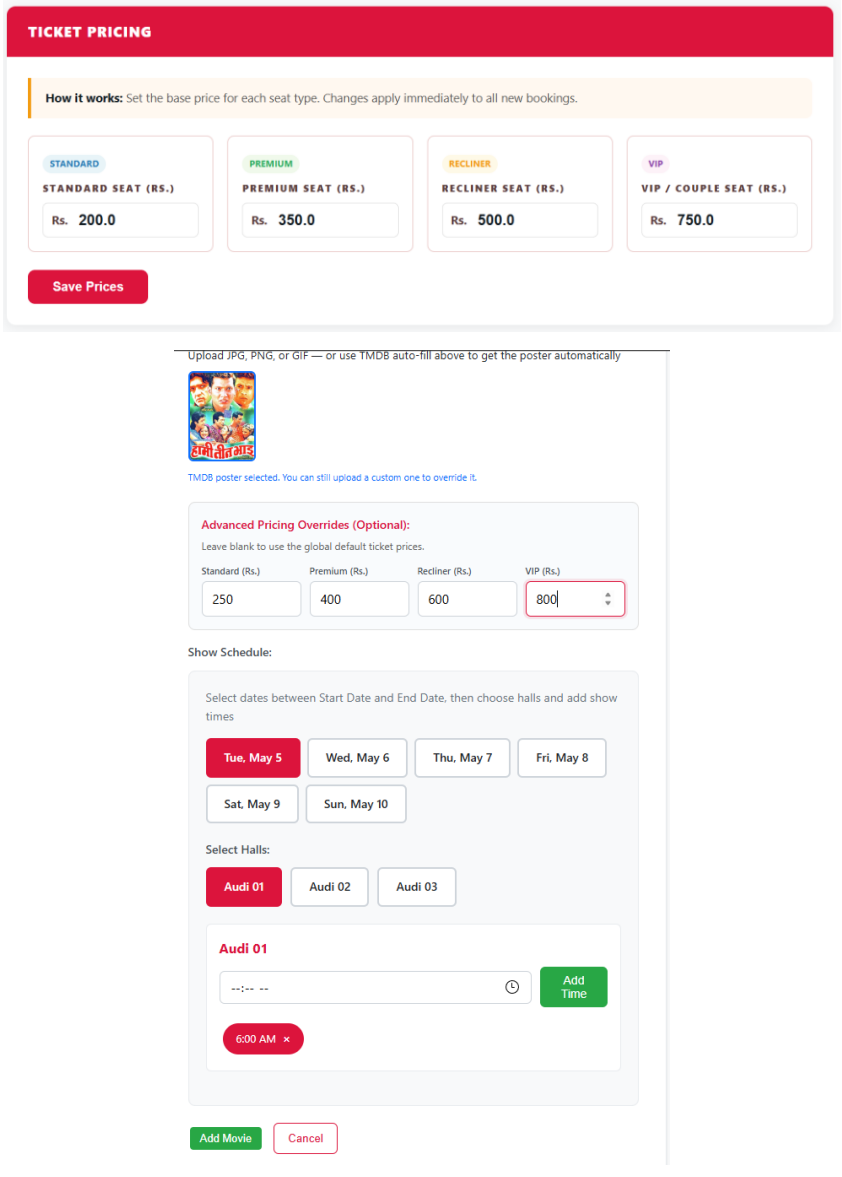
Field	Description
Test Name	Functional Test - Dynamic Four-Tier Seat Pricing Calculation
Given Input	User selects two Standard seats, one Premium seat, and one VIP seat from the dynamic seat map.
Expected Outcome	The total price should be calculated by applying the correct price for each selected seat tier. The booking summary should update immediately before the booking is confirmed.
Actual Outcome	
Test Result	Passed

Table 7: Functional Test - Movie-Specific Price Override

Field	Description
Test Name	Functional Test - Movie-Specific Price Override
Given Input	Admin creates or updates a movie with custom Standard, Premium, Recliner, and VIP prices. User then selects seats for that movie.
Expected Outcome	The booking page should prioritise movie-specific prices over global system settings. The calculated total should match the custom pricing values assigned to the selected movie.
Actual Outcome	

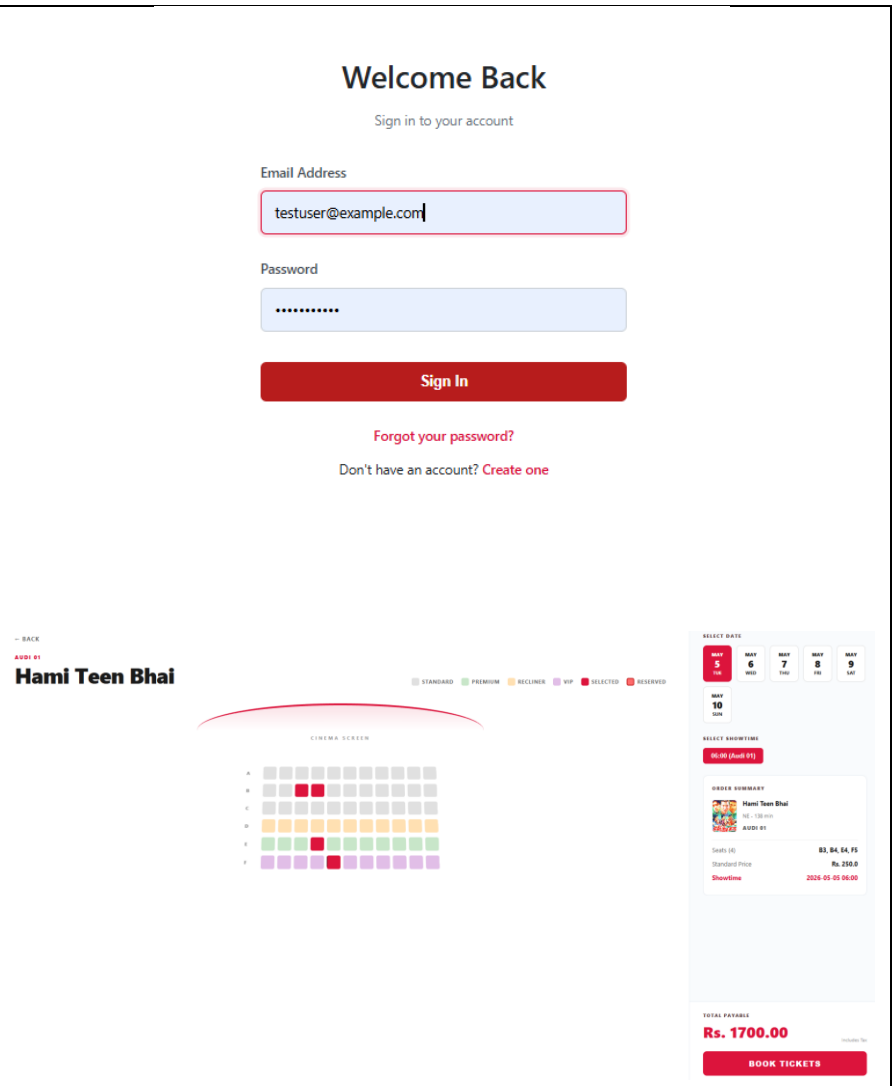
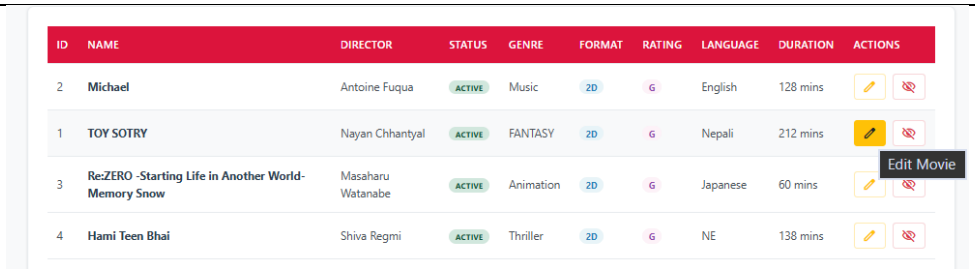
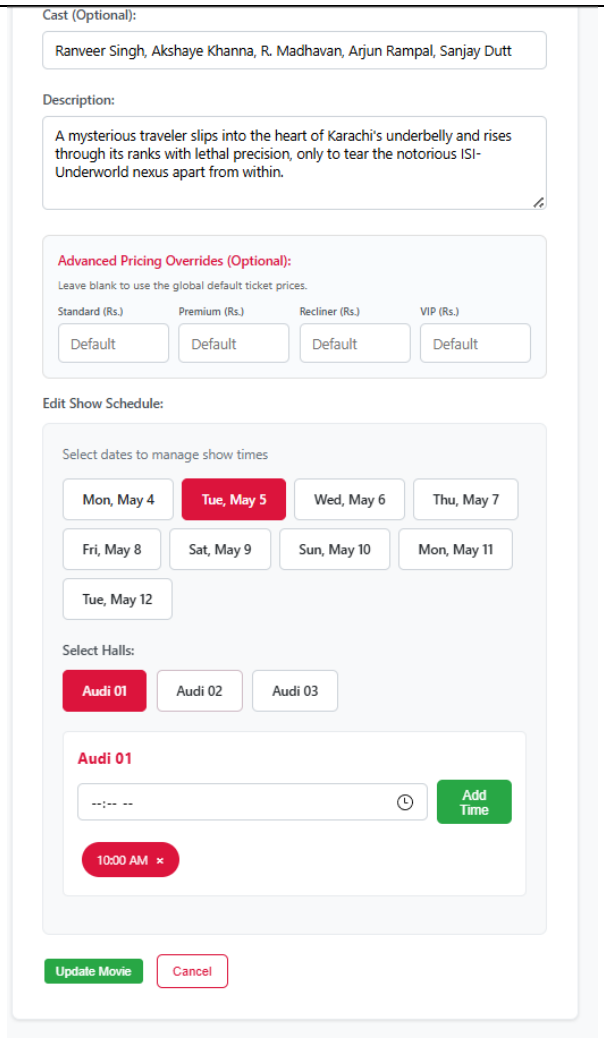
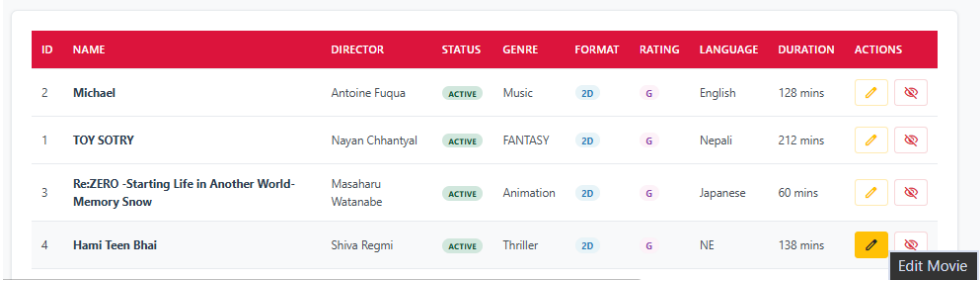
	
Test Result	Passed

Table 8: Functional Test - Show Scheduling Conflict with 30-Minute Cleaning Buffer

Field	Description
Test Name	Functional Test - Show Scheduling Conflict with 30-Minute Cleaning Buffer
Given Input	Admin attempts to schedule a new movie in the same hall immediately after an existing movie ends without allowing the mandatory 30-minute cleaning buffer.
Expected Outcome	The scheduling logic should reject the new showtime because the hall is still unavailable. A clear conflict message should be displayed to the administrator.
Actual Outcome	


Cast (Optional):
Ranveer Singh, Akshaye Khanna, R. Madhavan, Arjun Rampal, Sanjay Dutt
Description:
A mysterious traveler slips into the heart of Karachi's underbelly and rises through its ranks with lethal precision, only to tear the notorious ISI-Underworld nexus apart from within.
Advanced Pricing Overrides (Optional):
Leave blank to use the global default ticket prices.
Standard (Rs.)
Premium (Rs.)
Recliner (Rs.)
VIP (Rs.)
Default
Default
Default
Default
Edit Show Schedule:
Select dates to manage show times
Mon, May 4
Tue, May 5
Wed, May 6
Thu, May 7
Fri, May 8
Sat, May 9
Sun, May 10
Mon, May 11
Tue, May 12
Select Halls:
Audi 01
Audi 02
Audi 03
Audi 01
--:-- --
Add Time
10:00 AM ×
Update Movie
Cancel


ID NAME DIRECTOR STATUS GENRE FORMAT RATING LANGUAGE DURATION ACTIONS
2 Michael Antoine Fuqua ACTIVE Music 2D G English 128 mins
1 TOY SOTRY Nayan Chhantyal ACTIVE FANTASY 2D G Nepali 212 mins
3 Re:ZERO -Starting Life in Another World- Memory Snow Masaharu Watanabe ACTIVE Animation 2D G Japanese 60 mins
4 Hami Teen Bhai Shiva Regmi ACTIVE Thriller 2D G NE 138 mins
Edit Movie

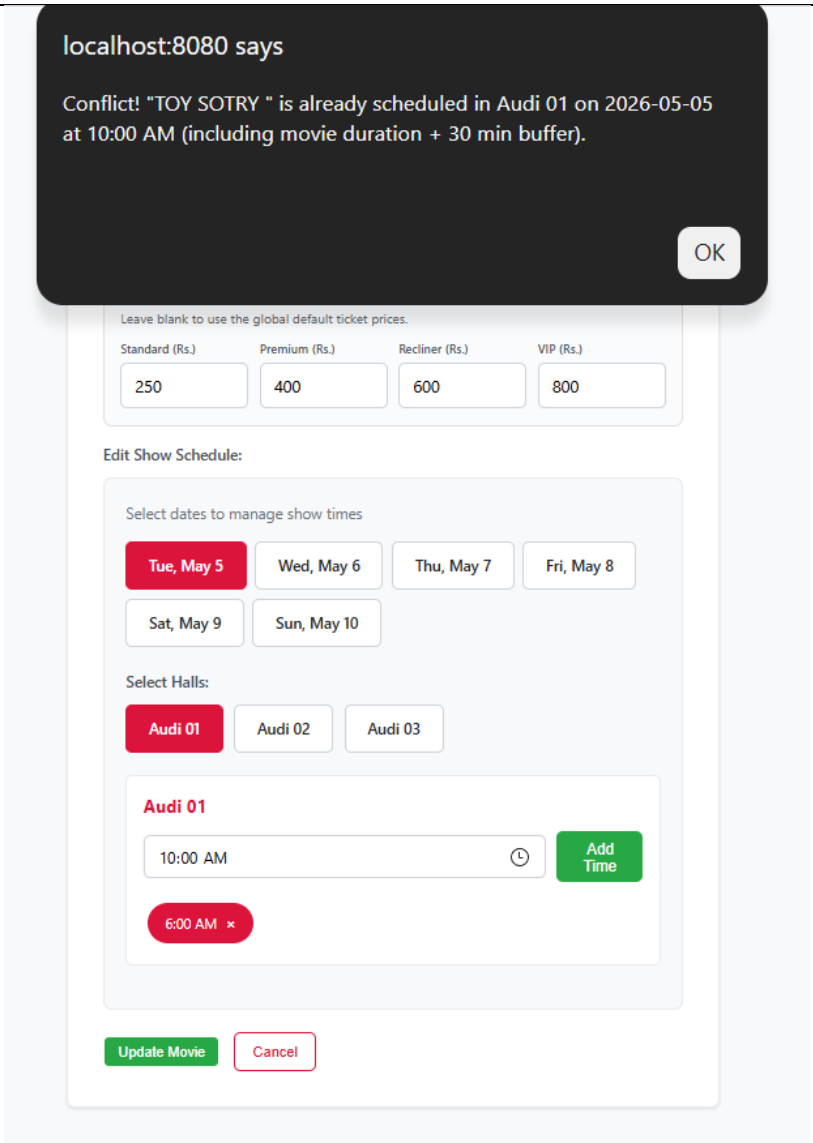
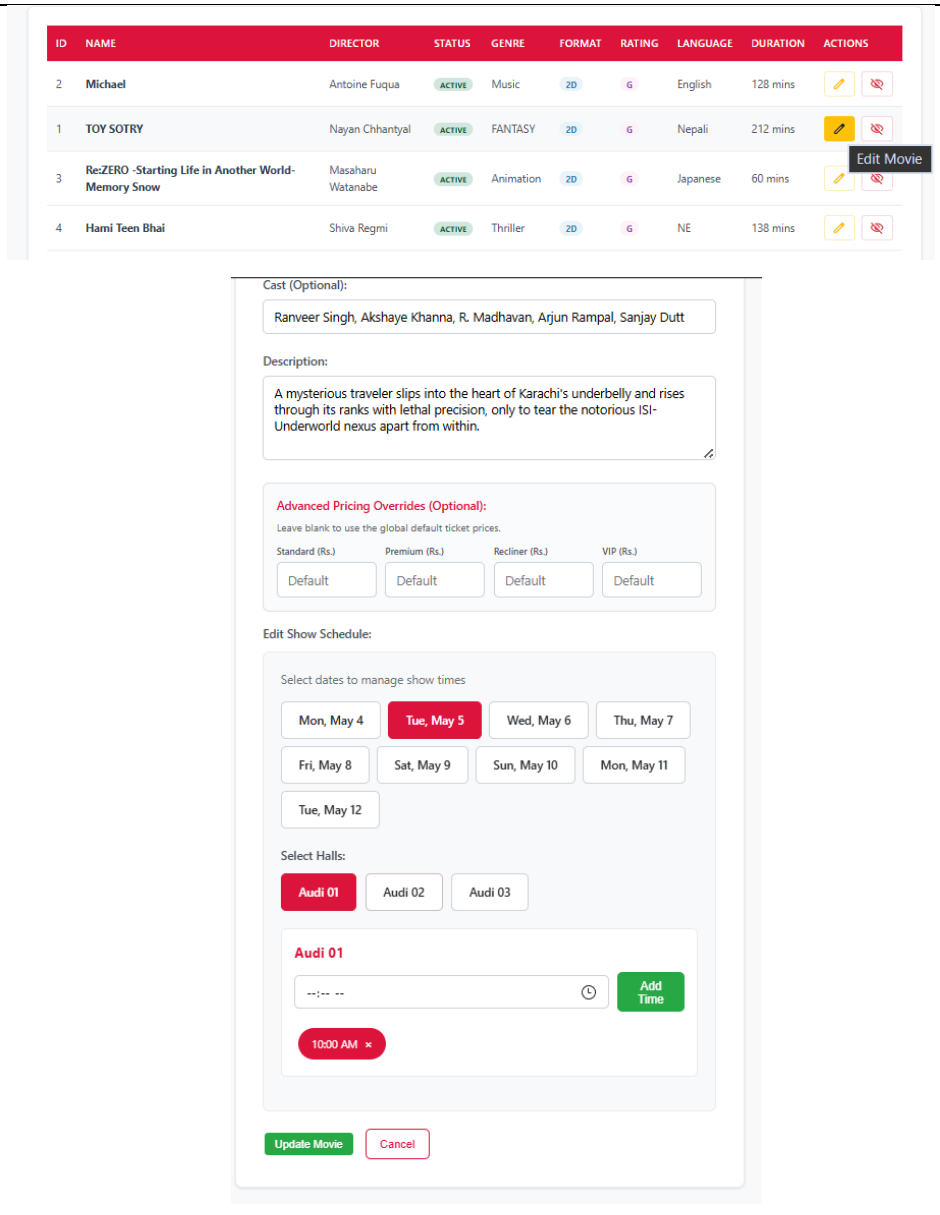
	
Test Result	Passed

Table 9: Functional Test - Valid Show Scheduling After Buffer Period

Field	Description
Test Name	Functional Test - Valid Show Scheduling After Buffer Period
Given Input	Admin schedules a movie in the same hall after the previous movie duration plus the required 30-minute buffer has passed.
Expected Outcome	The showtime should be accepted and inserted into the show_times table because the hall is available after the calculated buffer period.
Actual Outcome	

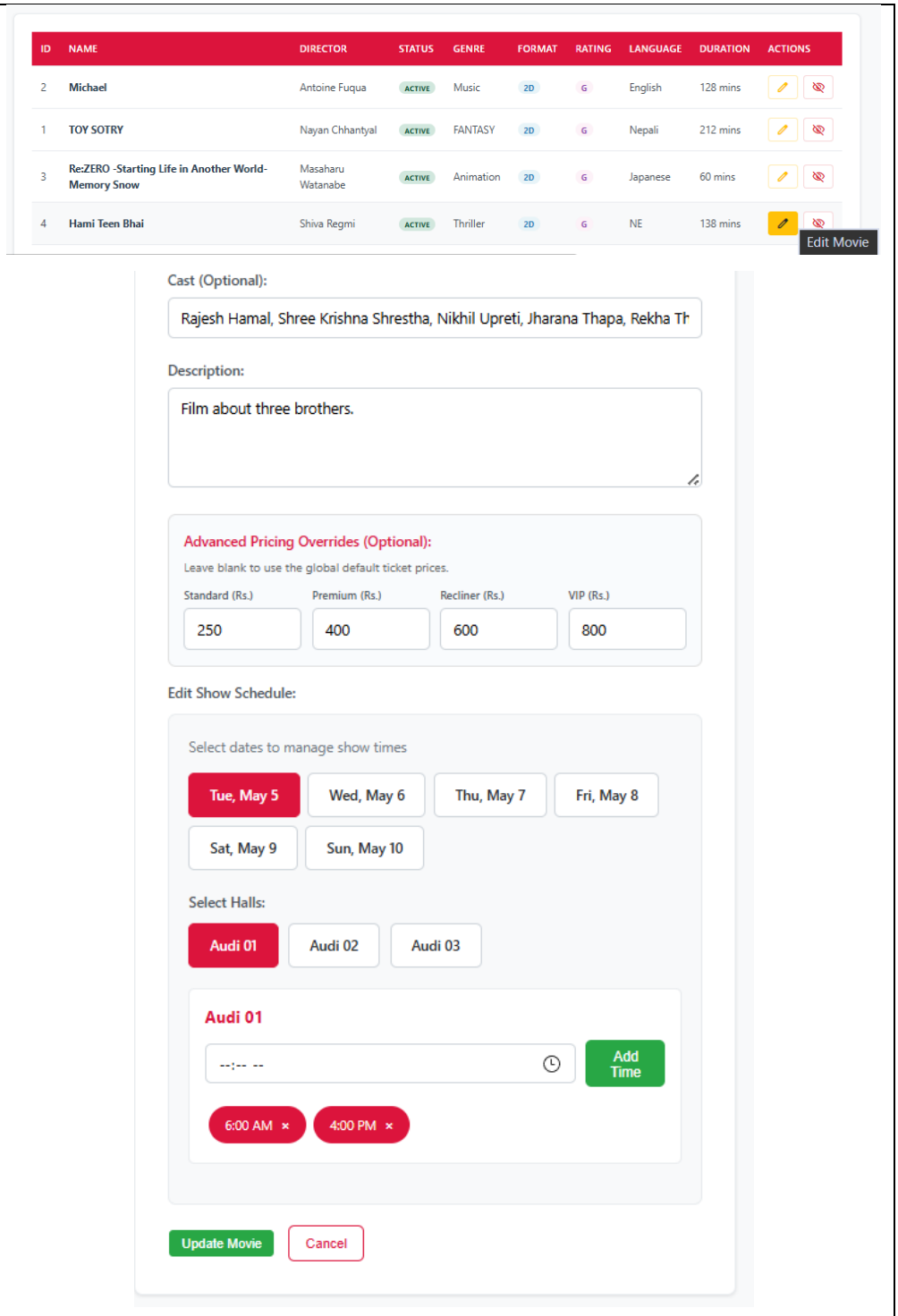
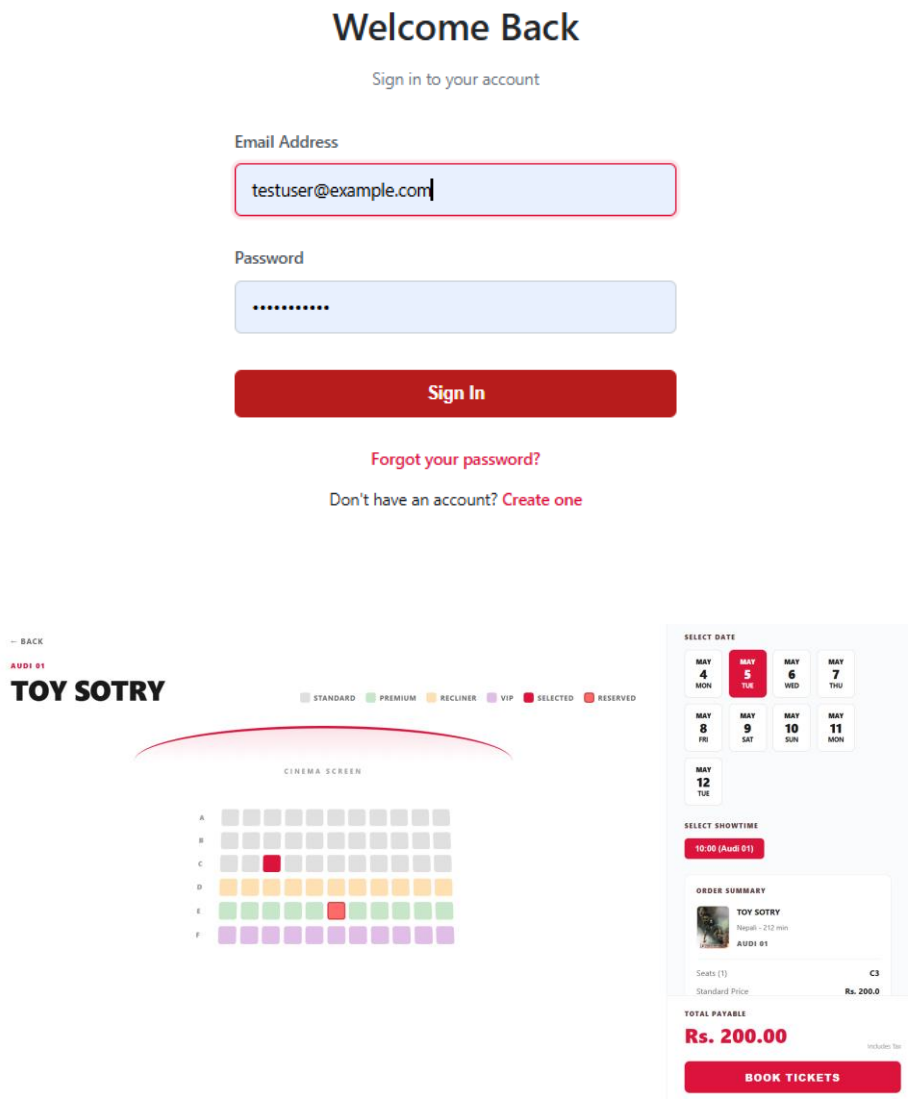
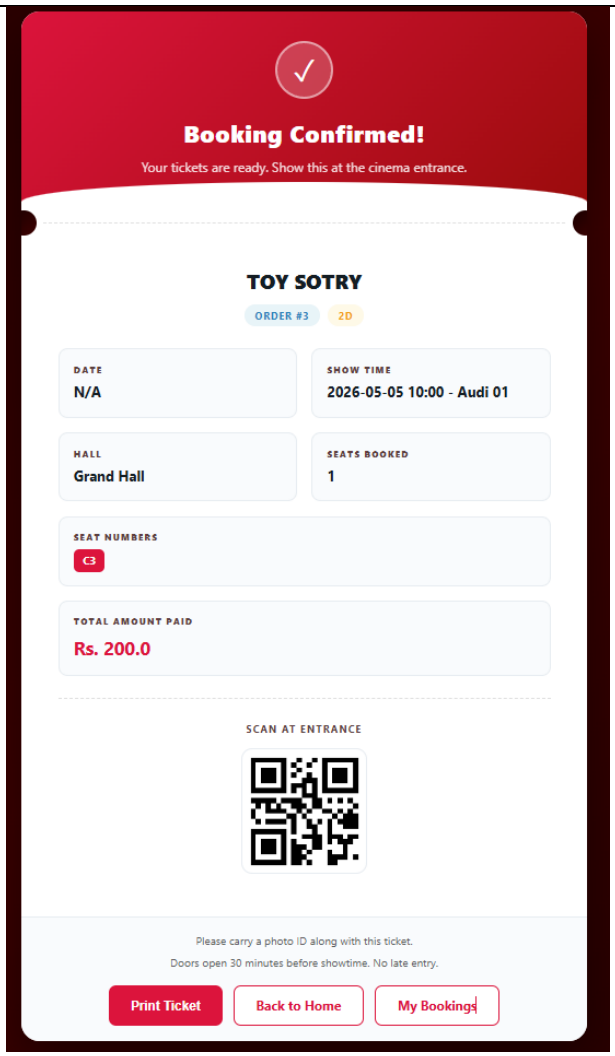
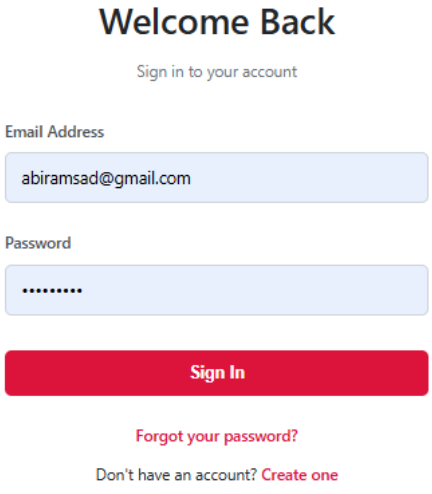
	
Test Result	Passed

Table 10: Functional Test - Concurrent Seat Booking Prevention

Field	Description
Test Name	Functional Test - Concurrent Seat Booking Prevention
Given Input	Two users attempt to book seat B4 for the same movie, date, time, and hall at nearly the same moment.
Expected Outcome	The first booking should be confirmed successfully. The second booking should be rejected after the system detects that the seat has already been reserved, preventing duplicate allocation of the same physical seat.
Actual Outcome	


Booking Confirmed!
Your tickets are ready. Show this at the cinema entrance.
TOY SOTRY
ORDER #3
2D
DATE
N/A
SHOW TIME
2026-05-05 10:00 - Audi 01
HALL
Grand Hall
SEATS BOOKED
1
SEAT NUMBERS
G3
TOTAL AMOUNT PAID
Rs. 200.0
SCAN AT ENTRANCE
Please carry a photo ID along with this ticket.
Doors open 30 minutes before showtime. No late entry.
Print Ticket
Back to Home
My Bookings


Welcome Back
Sign in to your account
Email Address
abiramsad@gmail.com
Password
Sign In
Forgot your password?
Don't have an account? Create one

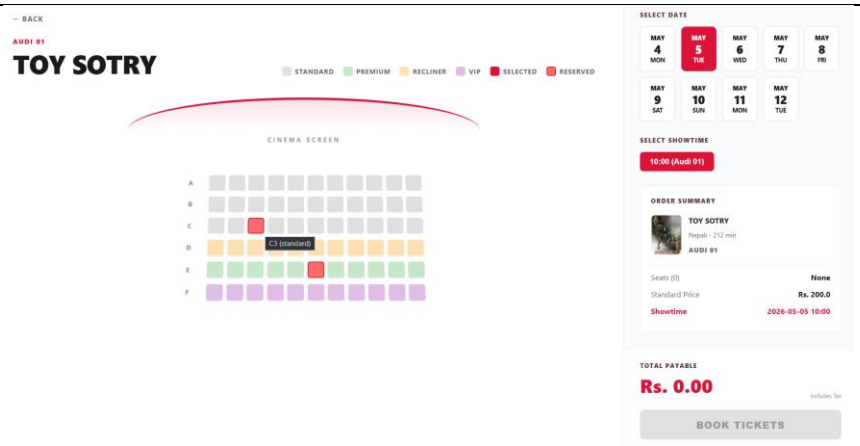
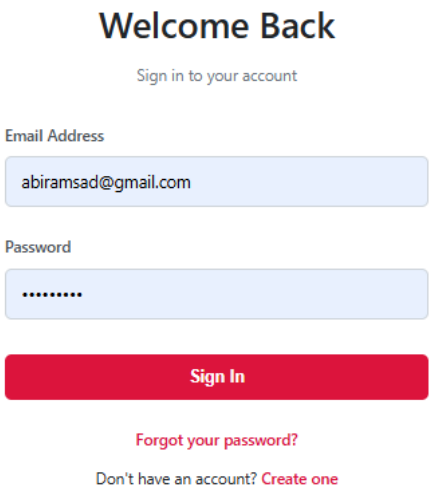
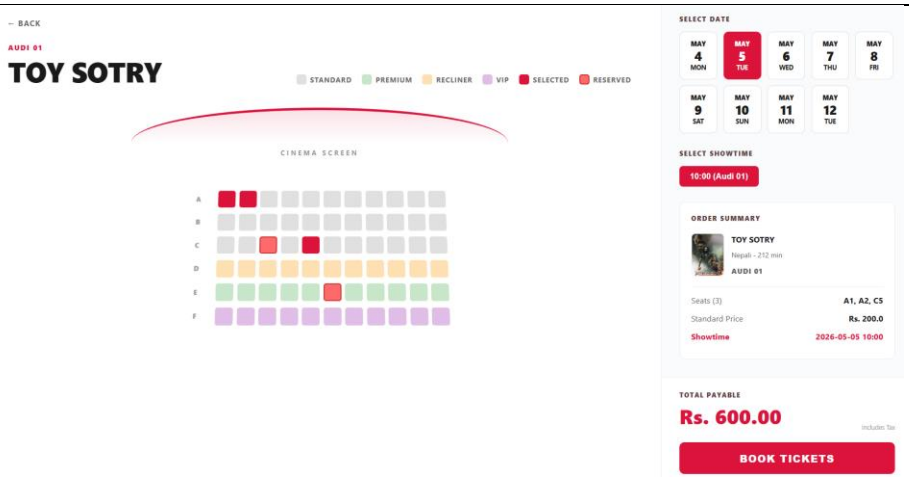
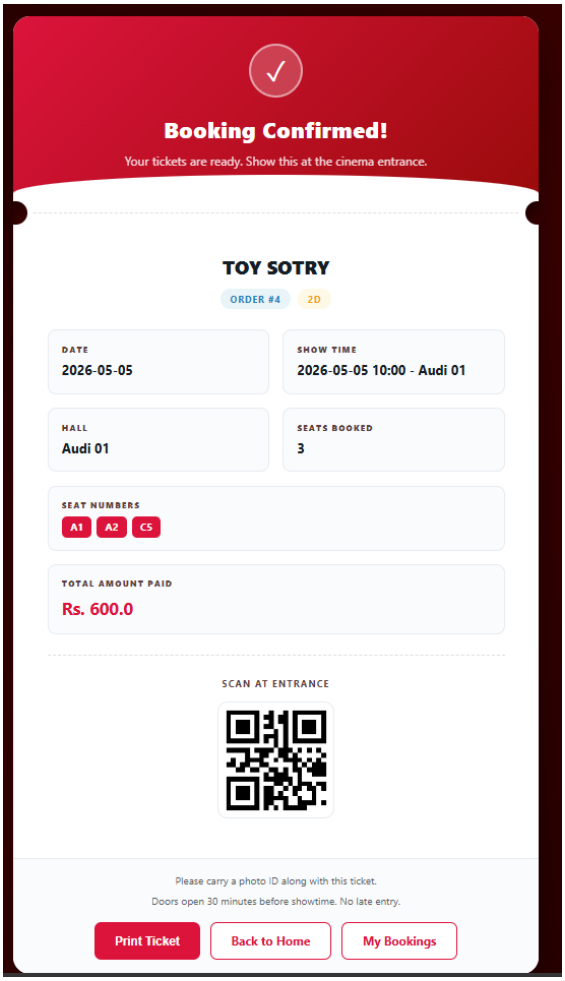
	
Test Result	Passed

Table 11: Functional Test - Dynamic Seat Availability Loading

Field	**Description**
Test Name	Functional Test - Dynamic Seat Availability Loading
Given Input	User opens the booking page for a showtime where seats A1, A2, and C5 have already been booked.
Expected Outcome	The backend should provide booked seat data to the JSP page, and the JavaScript seat map should render those seats as locked or unavailable.
Actual Outcome	





booking_id	user_id	movie_id	show_time	number_of_seats	seat_type	total_price	status	booking_date
1	3	1	2026-05-05 10:00 - Audi 01	1	E6	200	Confirmed	2026-05-04 00:34:36
2	5	1	2026-05-05 10:00 - Audi 01	1	E7	200	Cancelled	2026-05-04 11:19:42
3	5	1	2026-05-05 10:00 - Audi 01	1	C3	200	Confirmed	2026-05-04 15:00:14
4	3	1	2026-05-05 10:00 - Audi 01	3	A1, A2, C5	600	Confirmed	2026-05-04 15:08:36

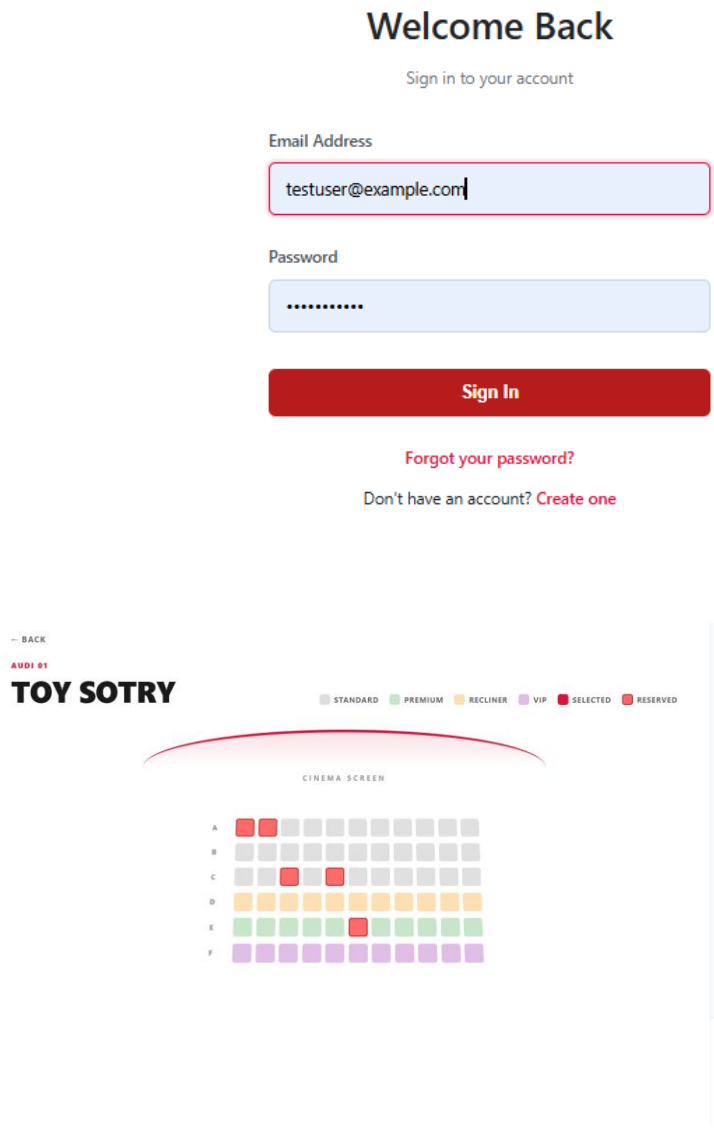
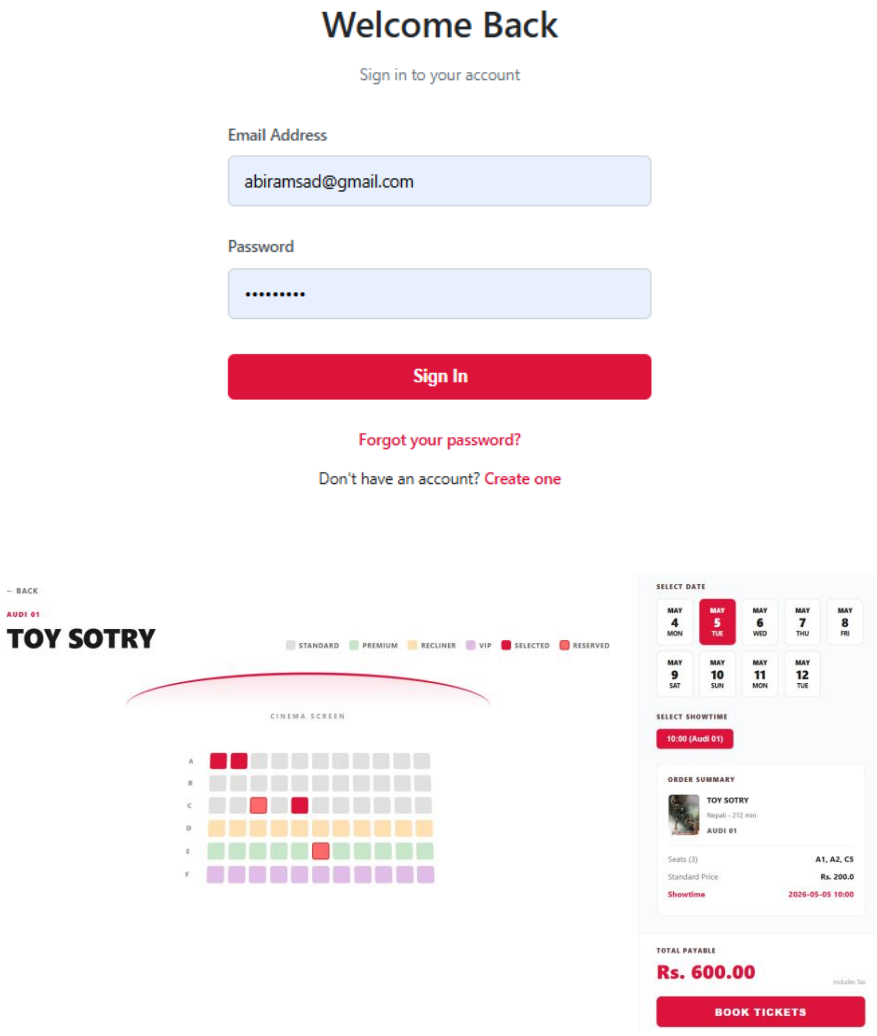
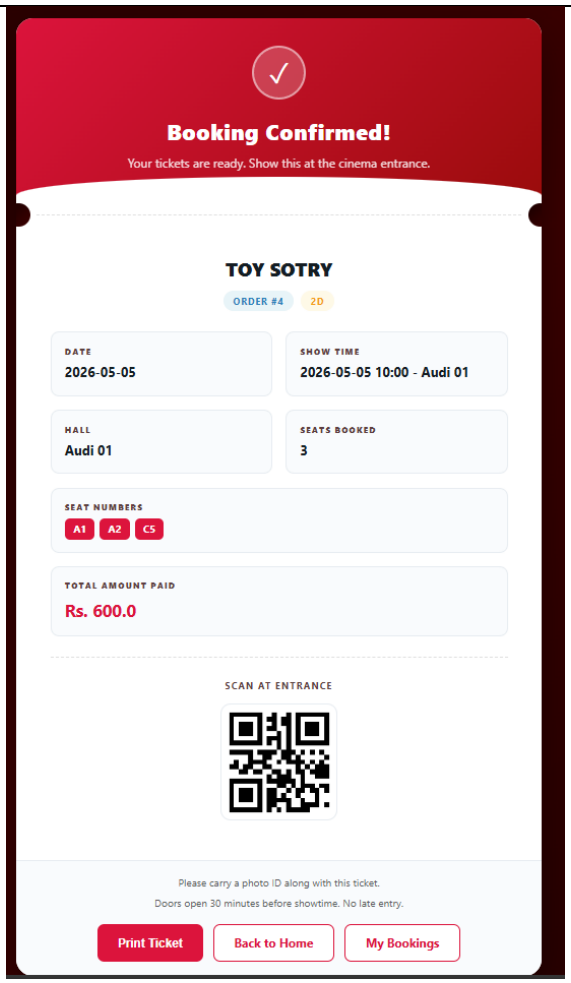
	
Test Result	Passed

Table 12: Functional Test - Ticket Confirmation Generation

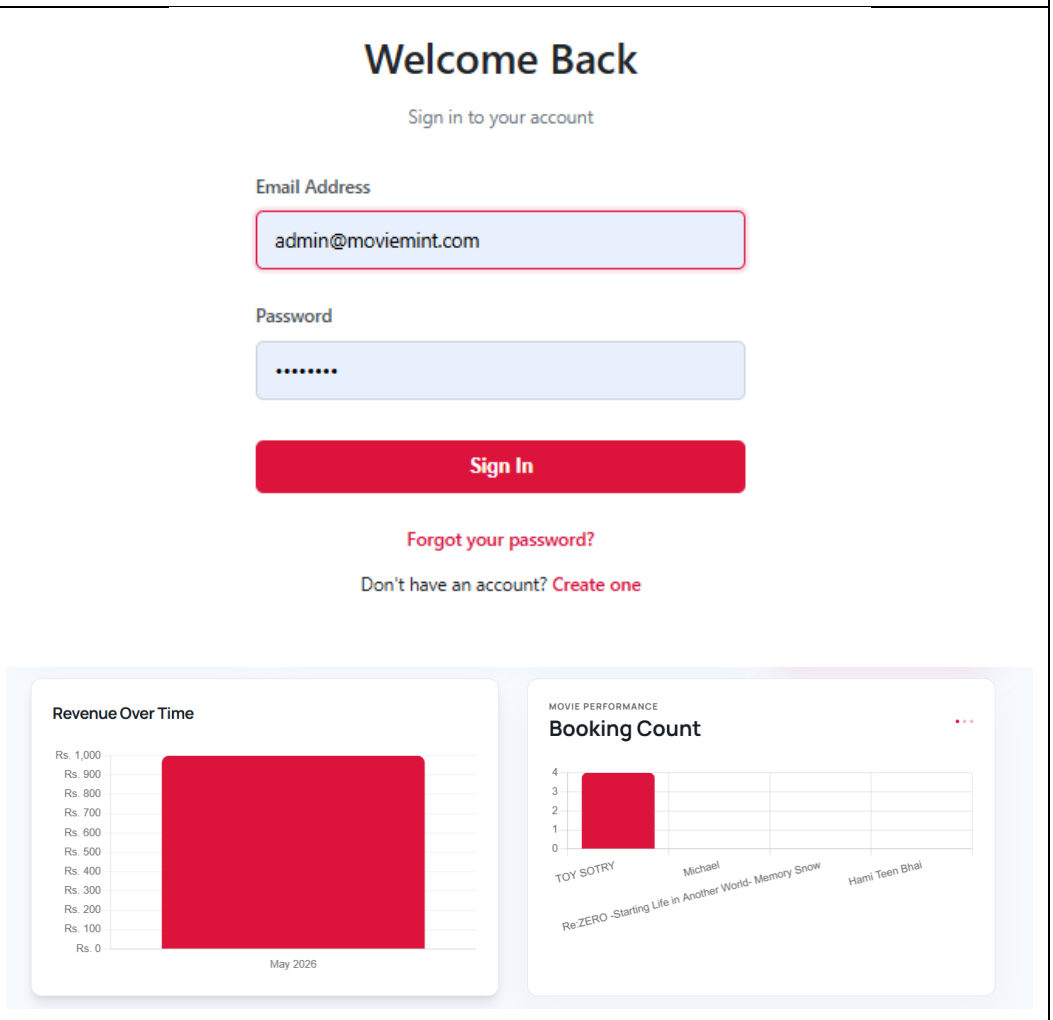
Field	Description
Test Name	Functional Test - Ticket Confirmation Generation
Given Input	User confirms a booking after selecting a valid movie, date, time, hall, and seat combination.
Expected Outcome	The system should create a booking record, calculate the final price, assign Confirmed status, and display a ticket confirmation containing movie title, date, time, hall, selected seats, and total price.
Actual Outcome	

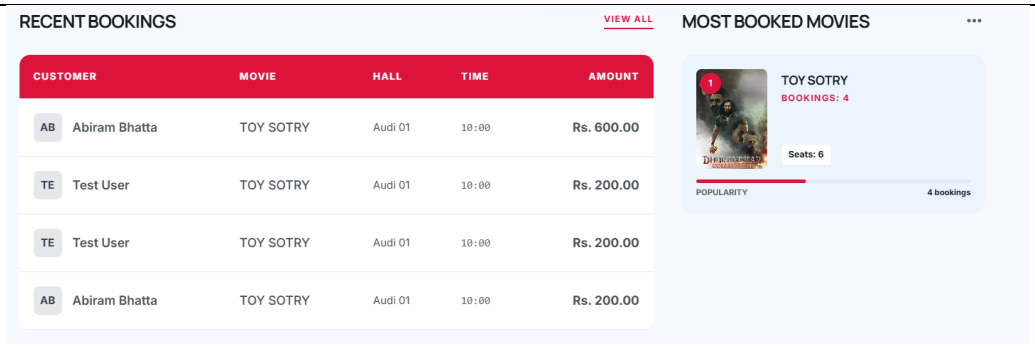


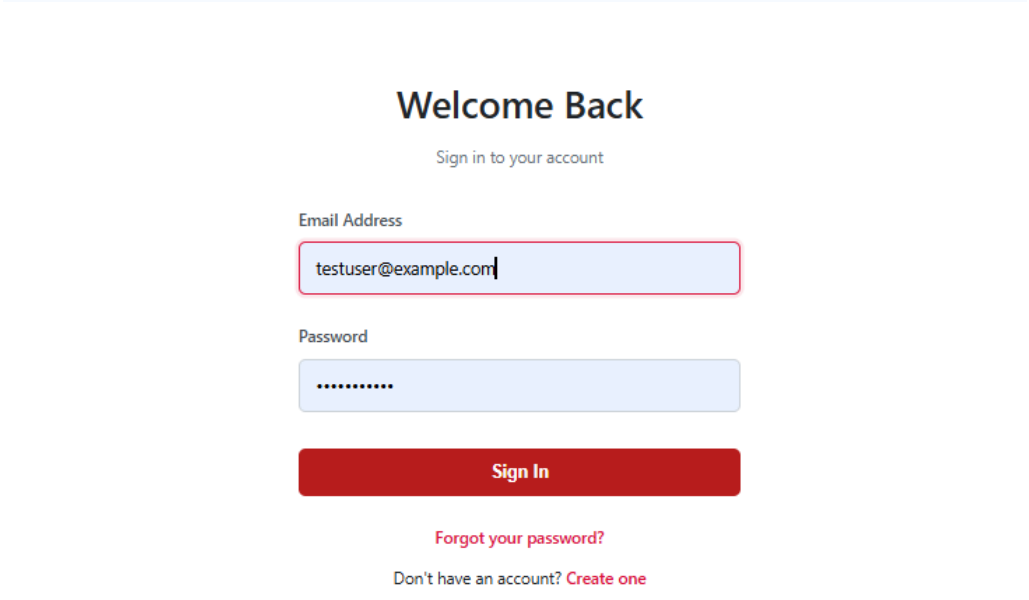
booking_id	user_id	movie_id	show_time	number_of_seats	seat_type	total_price	status	booking_date
1	3	1	2026-05-05 10:00 - Audi 01	1	E6	200	Confirmed	2026-05-04 00:34:36
2	5	1	2026-05-05 10:00 - Audi 01	1	E7	200	Cancelled	2026-05-04 11:19:42
3	5	1	2026-05-05 10:00 - Audi 01	1	C3	200	Confirmed	2026-05-04 15:00:14
4	3	1	2026-05-05 10:00 - Audi 01	3	A1, A2, C5	600	Confirmed	2026-05-04 15:08:36

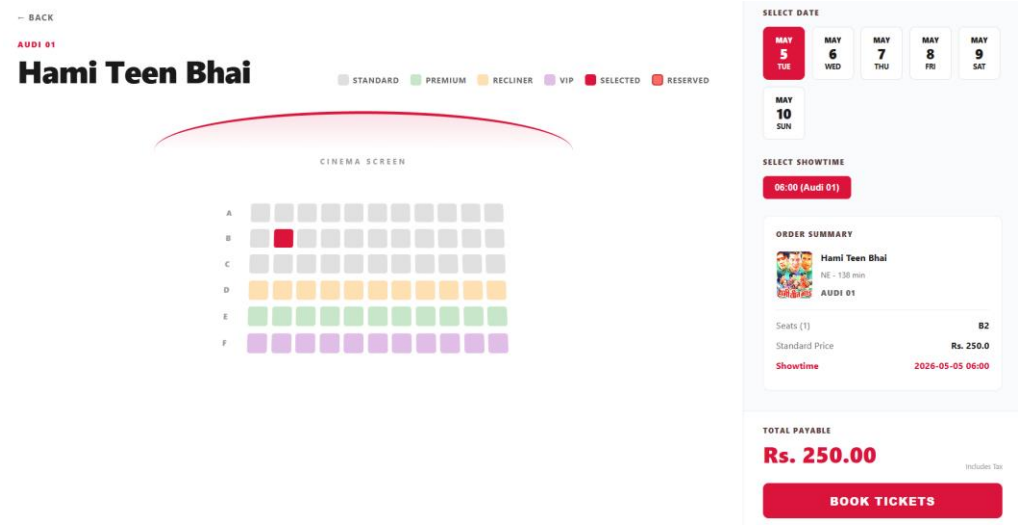
Test Result	Passed

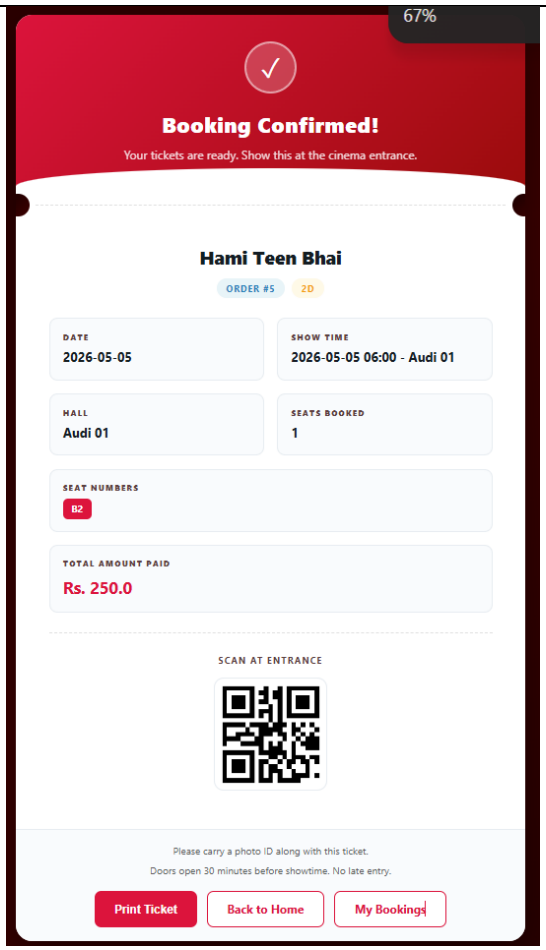
Table 13: Functional Test - Admin Dashboard Analytics Refresh

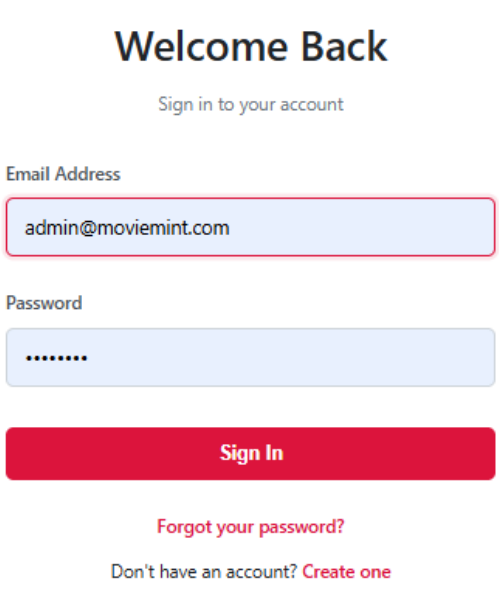
Field	Description
Test Name	Functional Test - Admin Dashboard Analytics Refresh
Given Input	A new confirmed booking is created and the administrator refreshes the dashboard.
Expected Outcome	Dashboard metrics such as total revenue, booking count, recent activity, top booked movies, and chart values should update using the latest DAO aggregation results.
Actual Outcome	

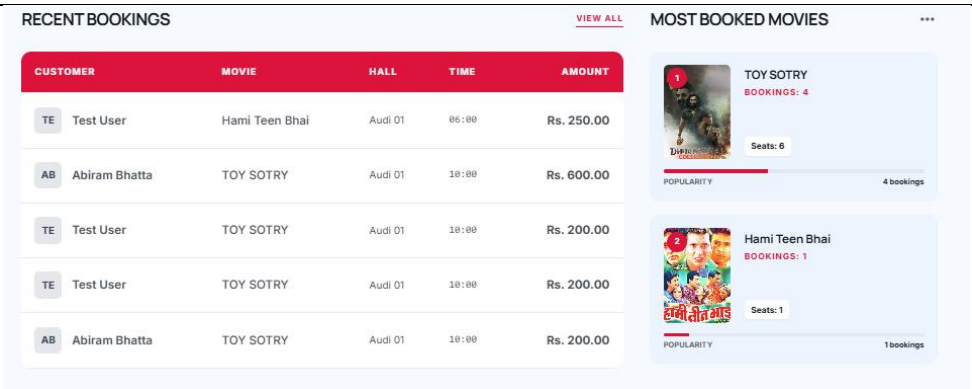
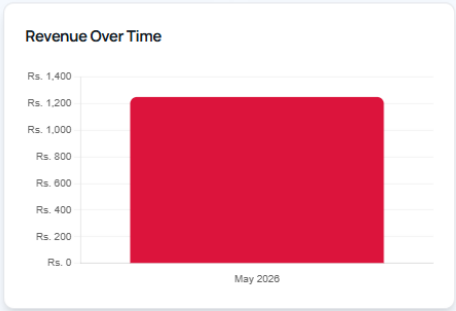
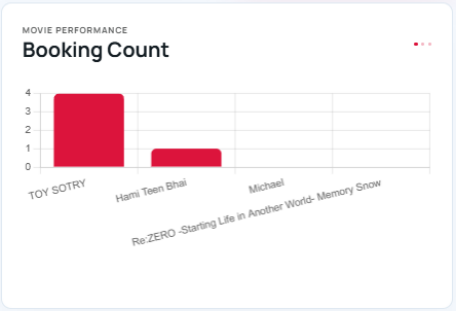

RECENT BOOKINGS
VIEW ALL
MOST BOOKED MOVIES
CUSTOMER
MOVIE
HALL
TIME
AMOUNT
AB Abiram Bhatta TOY SOTRY Audi 01 10:00 Rs. 600.00
TE Test User TOY SOTRY Audi 01 10:00 Rs. 200.00
TE Test User TOY SOTRY Audi 01 10:00 Rs. 200.00
AB Abiram Bhatta TOY SOTRY Audi 01 10:00 Rs. 200.00
TOY SOTRY
BOOKINGS: 4
Seats: 6
POPULARITY
4 bookings


Welcome Back
Sign in to your account
Email Address
testuser@example.com
Password
Sign In
Forgot your password?
Don't have an account? Create one


BACK
AUDI 01
Hami Teen Bhai
STANDARD
PREMIUM
RECLINER
VIP
SELECTED
RESERVED
CINEMA SCREEN
SELECT DATE
SELECT SHOWTIME
06:00 (Audi 01)
ORDER SUMMARY
Hami Teen Bhai
AUDI 01
Seats (1) B2
Standard Price Rs. 250.0
Showtime 2026-05-05 06:00
TOTAL PAYABLE
Rs. 250.00
BOOK TICKETS


67%
Booking Confirmed!
Your tickets are ready. Show this at the cinema entrance.
Hami Teen Bhai
ORDER #5
2D
DATE
2026-05-05
SHOW TIME
2026-05-05 06:00 - Audi 01
HALL
Audi 01
SEATS BOOKED
1
SEAT NUMBERS
B2
TOTAL AMOUNT PAID
Rs. 250.0
SCAN AT ENTRANCE
Please carry a photo ID along with this ticket.
Doors open 30 minutes before showtime. No late entry.
Print Ticket
Back to Home
My Bookings


Welcome Back
Sign in to your account
Email Address
admin@moviemint.com
Password
Sign In
Forgot your password?
Don't have an account? Create one

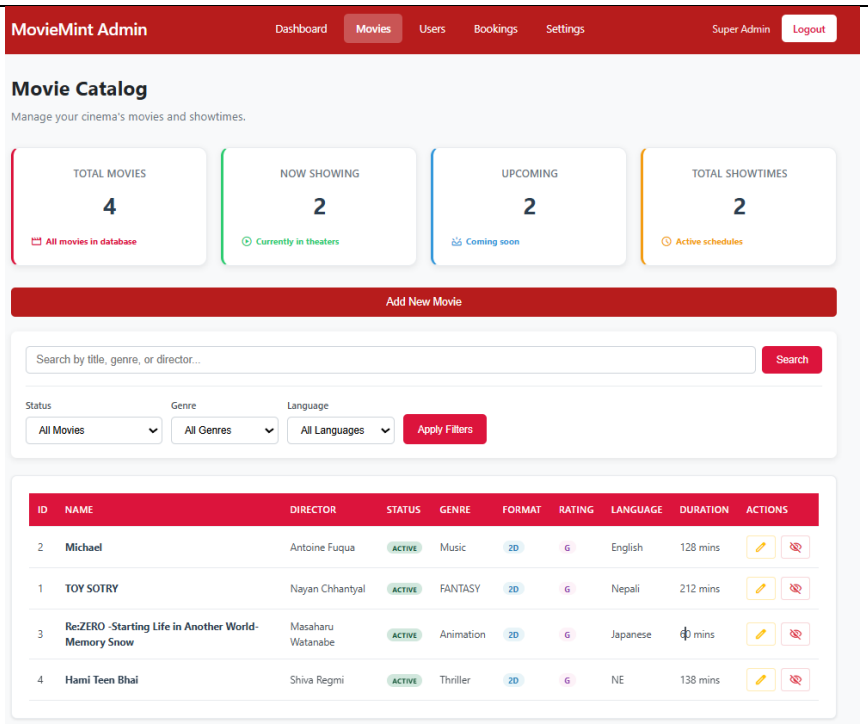
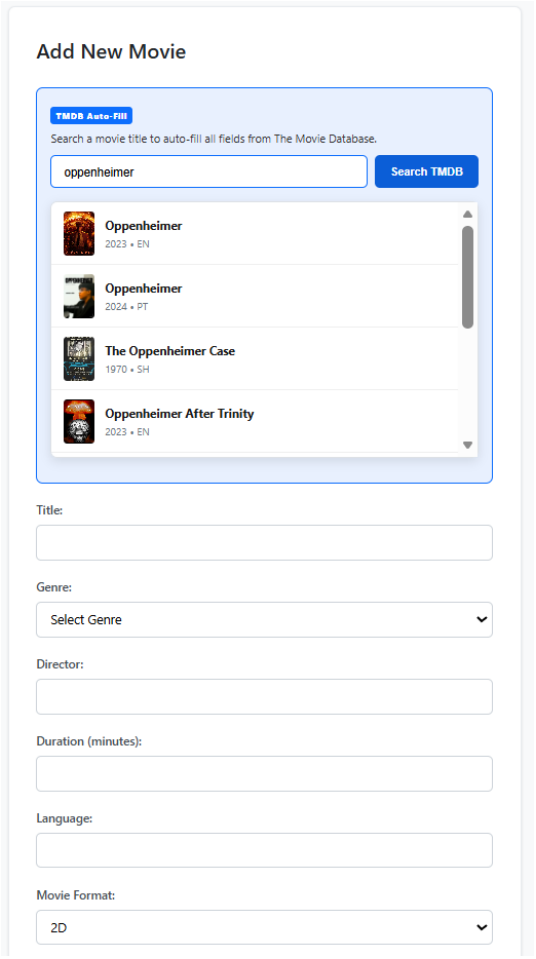
	  
Test Result	Passed

5.3 API & Integration Testing

Tables 14 to 18 assess the reliability of external integrations. These tests are important because MovieMint depends on TMDB for movie metadata and Gmail SMTP for OTP and ticket email delivery.

Table 14: API Test - TMDB Search JSON Parsing Success

Field	Description
Test Name	API Test - TMDB Search JSON Parsing Success
Given Input	Admin searches for a valid movie title through the TMDB auto-fill feature.
Expected Outcome	The system should receive a valid JSON response, parse the movie title, overview, poster path, release date, and related metadata, and populate the movie form fields correctly.

Actual Outcome	 

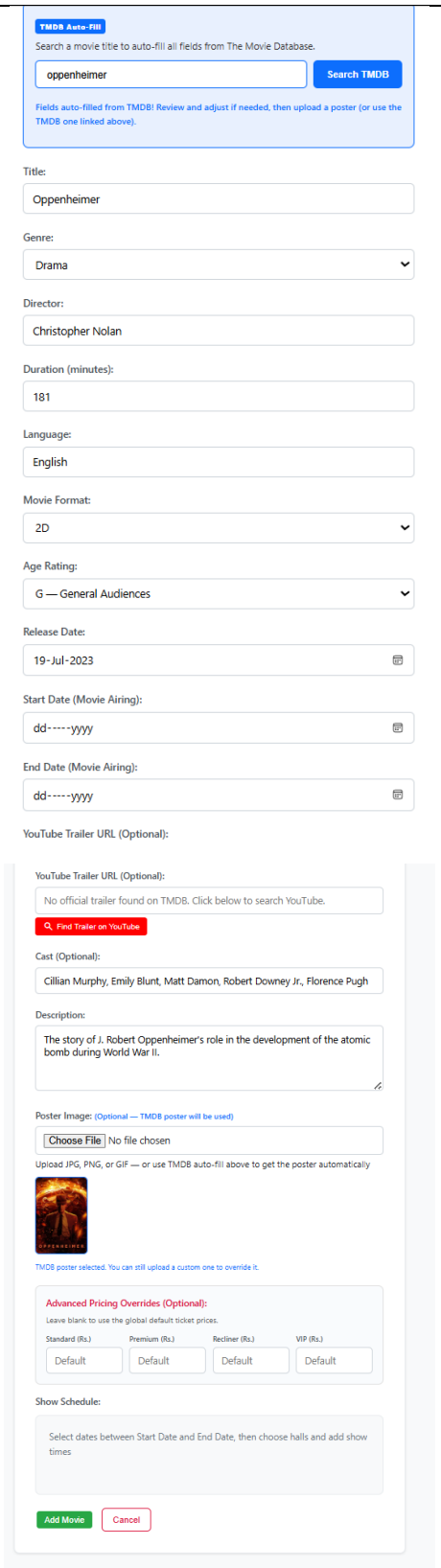
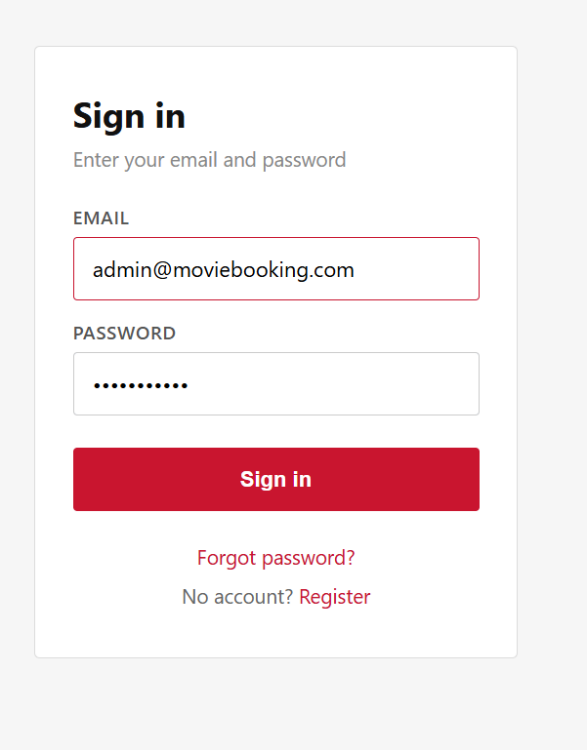
	
Test Result	Passed

Table 15: API Test - TMDB Poster Download and Local Storage

Field	Description
Test Name	API Test - TMDB Poster Download and Local Storage
Given Input	Admin selects a TMDB movie result that includes an available poster image.
Expected Outcome	The poster should be downloaded from TMDB image servers, saved locally in the application image directory, and linked to the movie record using the stored poster path.
Actual Outcome	

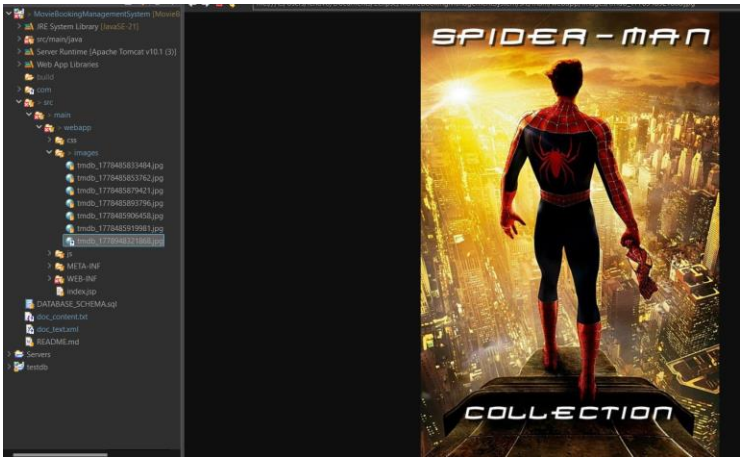
	Cast (Optional): Tobey Maguire, Willem Dafoe, Kirsten Dunst, James Franco, Cliff Robertsc Description: After being bitten by a genetically altered spider at Oscorp, nerdy but endearing high school student Peter Parker is endowed with amazing powers to become the superhero known as Spider-Man. Poster Image: (Optional — TMDB poster will be used) Choose File No file chosen Upload JPG, PNG, or GIF — or use TMDB auto-fill above to get the poster automatically  TMDB poster selected. You can still upload a custom one to override it. Advanced Pricing Overrides (Optional): Leave blank to use the global default ticket prices. Standard (Rs.) Premium (Rs.) Recliner (Rs.) VIP (Rs.) Default Default Default Default 
Test Result	Passed

Table 16: API Test - TMDB Timeout and Fallback Degradation

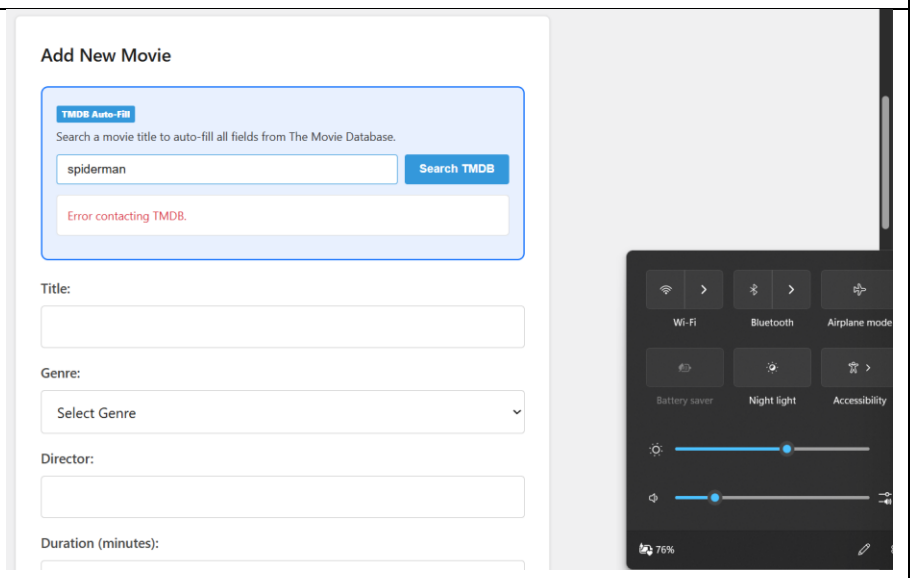
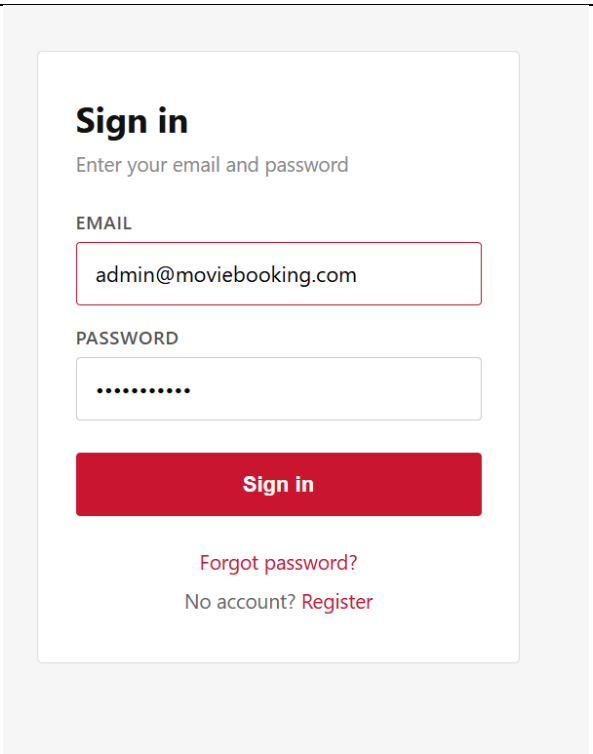
Field	Description
Test Name	API Test - TMDB Timeout and Fallback Degradation
Given Input	The TMDB request exceeds the expected response time or the API is temporarily unavailable.
Expected Outcome	The application should not crash. It should display a controlled error message and still allow the administrator to enter movie details manually.
Actual Outcome	
Test Result	Passed

Table 17: API Test - Missing Trailer Fallback Handling

Field	Description
Test Name	API Test - Missing Trailer Fallback Handling
Given Input	TMDB returns movie metadata but no official trailer URL.
Expected Outcome	The movie creation process should continue successfully. The trailer field should remain optional or be handled through the fallback trailer logic without blocking the catalogue workflow.
Actual Outcome	

Add New Movie

TMDB Auto-Fill

Search a movie title to auto-fill all fields from The Movie Database.

Avatar | Search TMDB

Fields auto-filled from TMDB! Review and adjust if needed, then upload a poster (or use the TMDB one linked above).

Title:

Avatar

Genre:

+ Custom Genre

Custom Genre:

Science Fiction

Director:

YouTube Trailer URL (Optional):

No official trailer found on TMDB. Click below to search YouTube.

search Find Trailer on YouTube

Cast (Optional):

Sam Worthington, Zoe Saldaña, Sigourney Weaver, Stephen Lang, Michel

Description:

In the 22nd century, a paraplegic Marine is dispatched to the moon Pandora on a unique mission, but becomes torn between following orders and protecting an alien civilization.

Poster Image: (Optional — TMDB poster will be used)

Choose File No file chosen

Upload JPG, PNG, or GIF — or use TMDB auto-fill above to get the poster automatically

TMDB poster selected. You can still upload a custom one to override it.

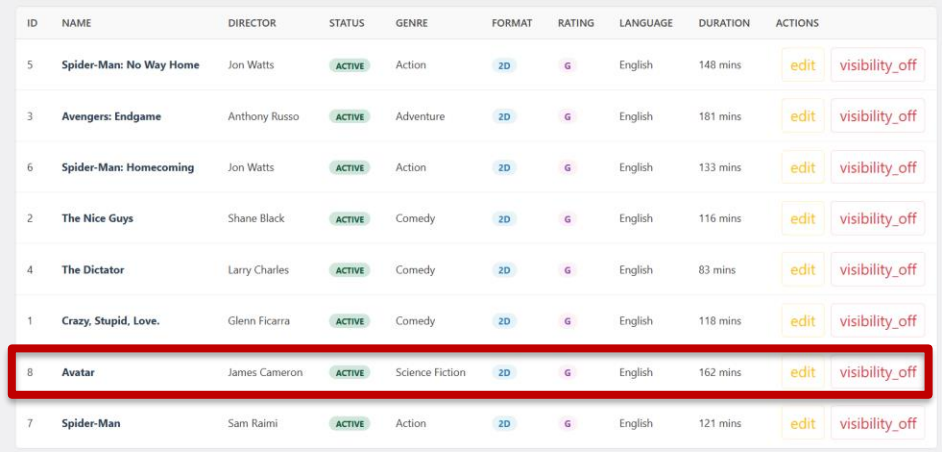
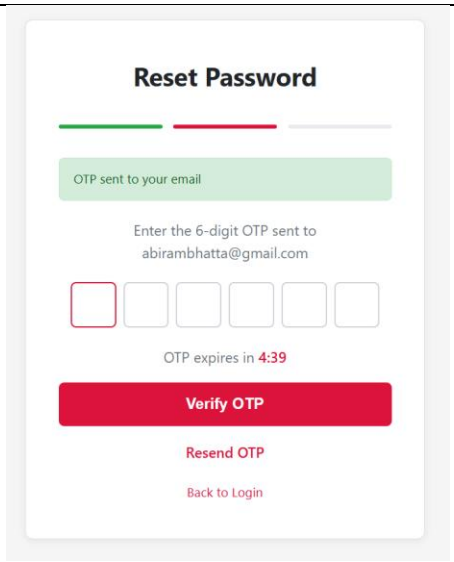
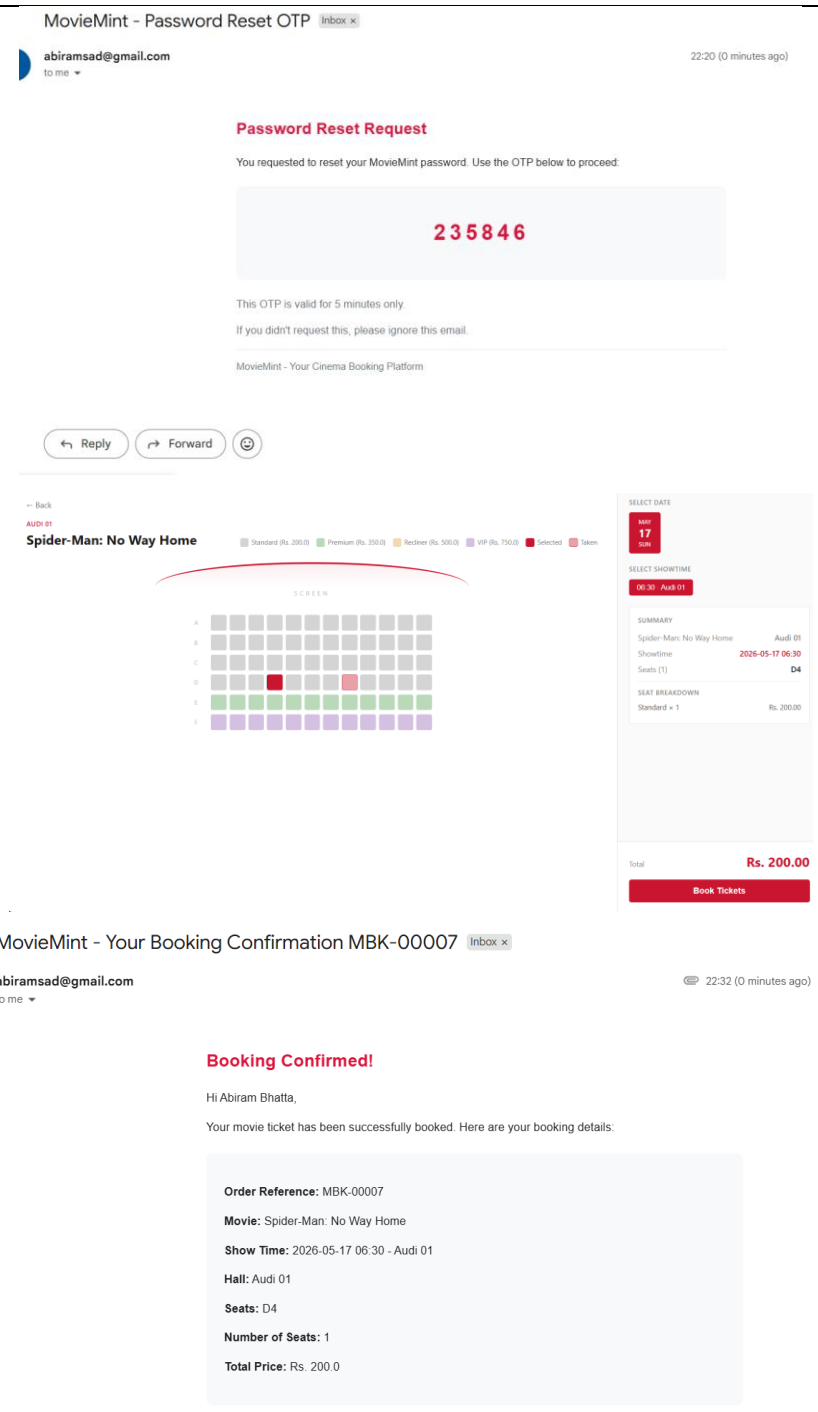
	
Test Result	Passed

Table 18: API Test - Gmail SMTP TLS Email Delivery

Field	**Description**
Test Name	API Test - Gmail SMTP TLS Email Delivery
Given Input	User requests a password reset OTP or receives a digital ticket after a successful booking.
Expected Outcome	The email service should connect to Gmail SMTP using TLS authentication and send the message successfully without exposing SMTP credentials in the user interface or response output.
Actual Outcome	

	
Test Result	Passed

5.4 Validation & Input Sanitization Testing

Table 19: Boundary Value Seat Selection

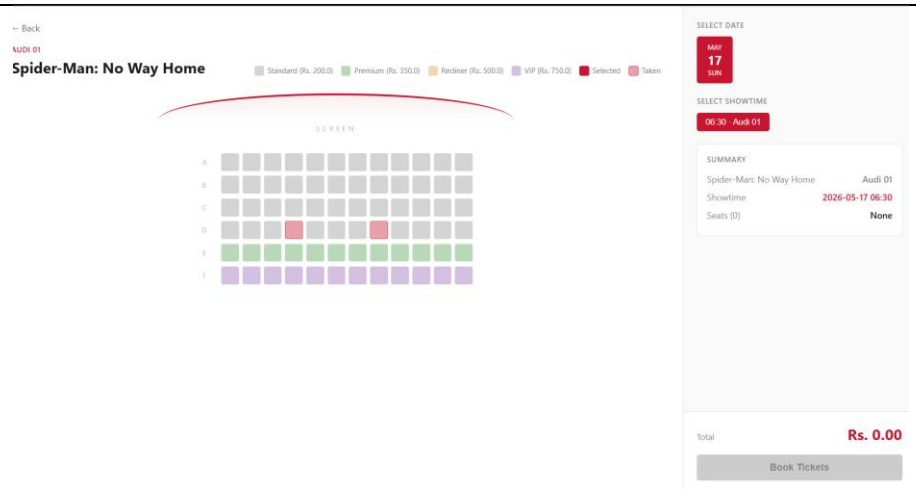
Field	Description
Test Name	Zero-Seat Booking Validation
Given Input	User navigates to the seat selection page but clicks "Book Tickets" without selecting any seats.
Expected Outcome	The system should block the transition and the Book Tickets Button shouldn't be clickable.
Actual Outcome	
Test Result	**Passed**

Table 20: Future-Date Enforcement

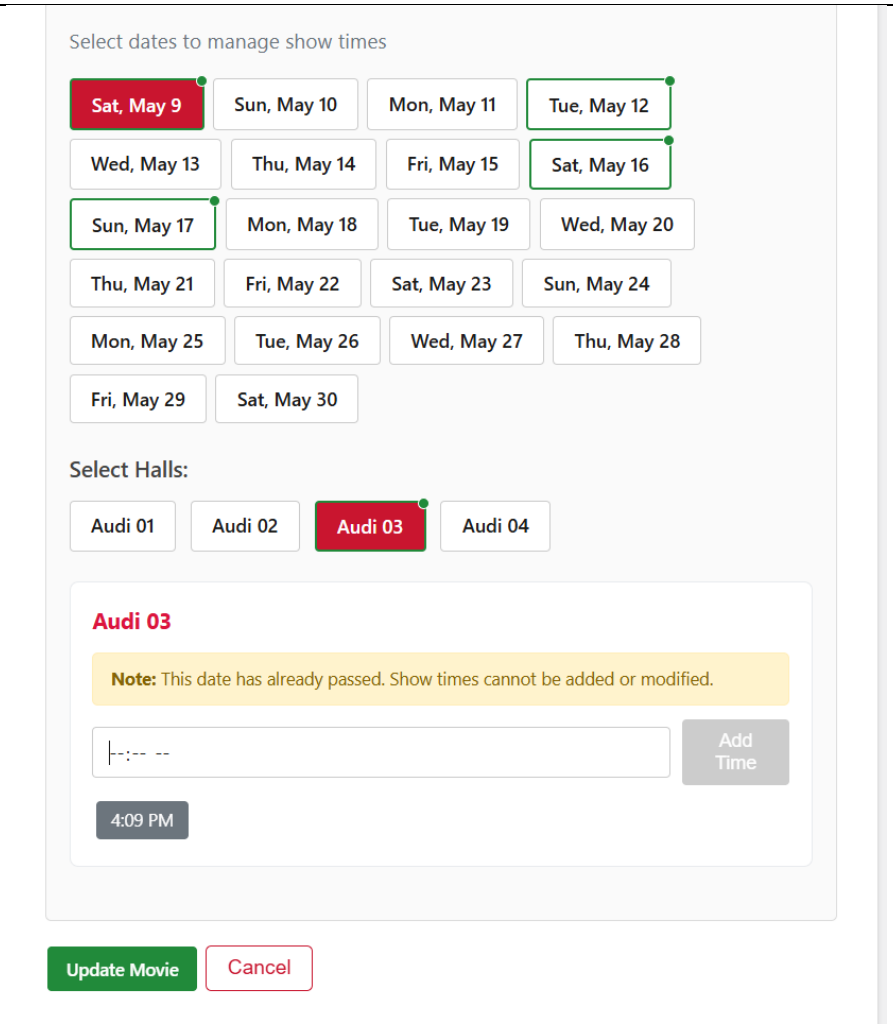
Field	Description
Test Name	Past-Date Showtime Restriction
Given Input	Admin attempts to create a new showtime for a date that has already passed.
Expected Outcome	The date-picker or backend validation should reject the entry to prevent "ghost" showtimes.
Actual Outcome	
Test Result	**Passed**

Table 21: Profile Update Field Length

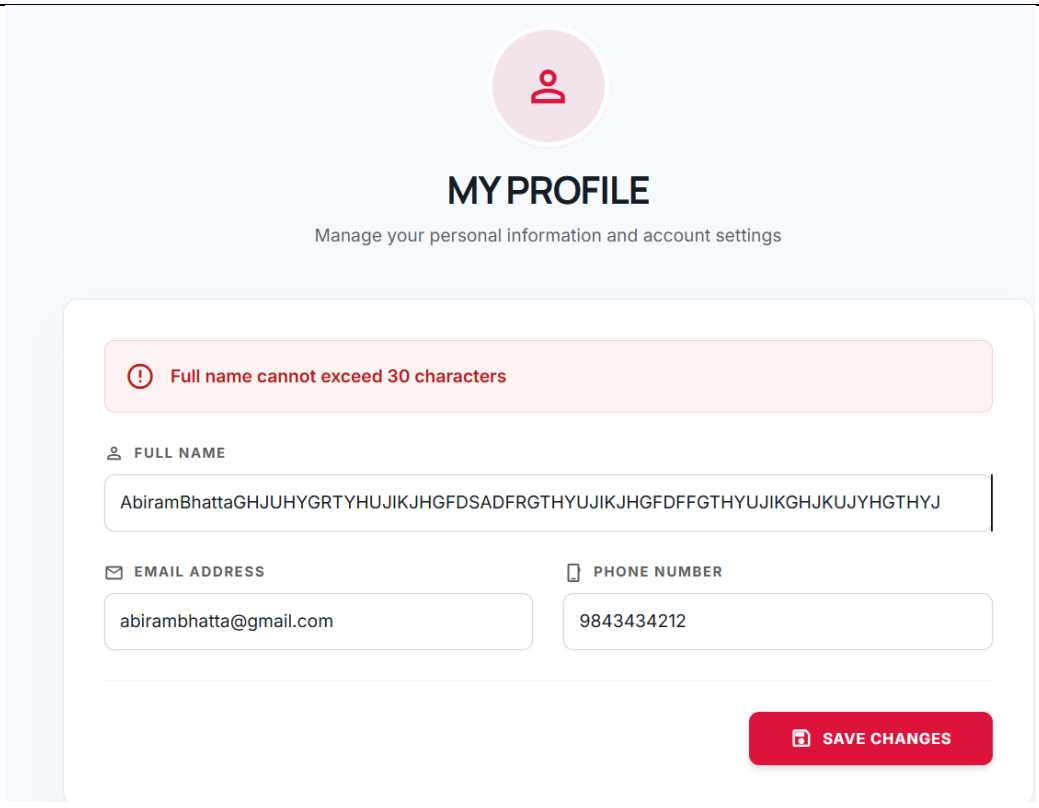
Field	Description
Test Name	Name Character Limit Validation
Given Input	User tries to change their name to a string of 50+ random characters in the 'Edit Profile' section.
Expected Outcome	The app should enforce a reasonable limit to prevent UI breakage or DB overflow.
Actual Outcome	
Test Result	**Passed**

Table 22: Complex Password Strength

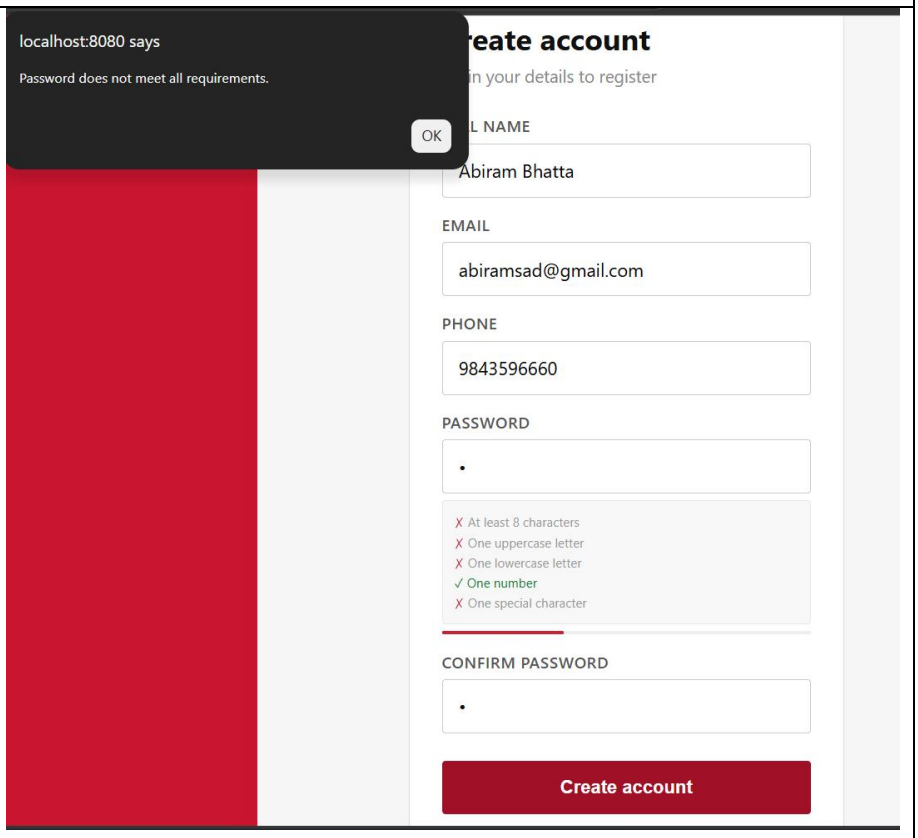
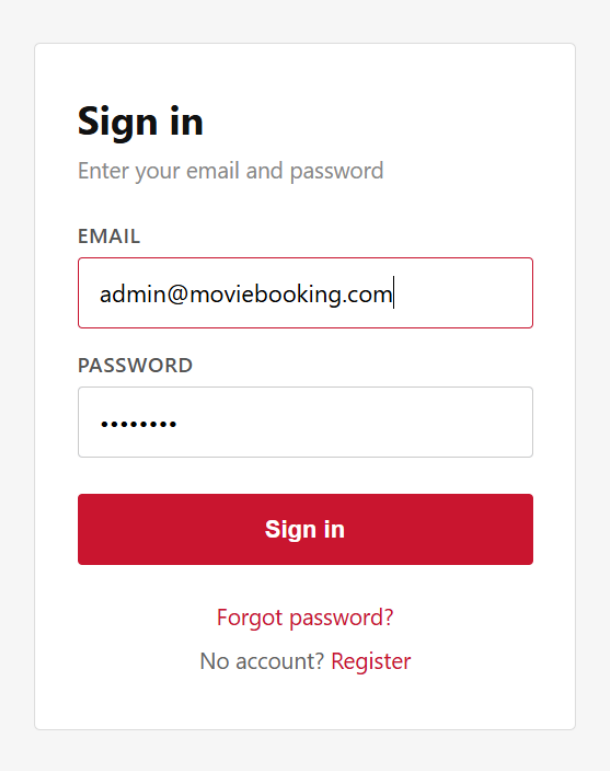
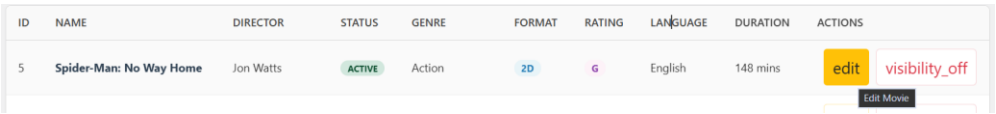
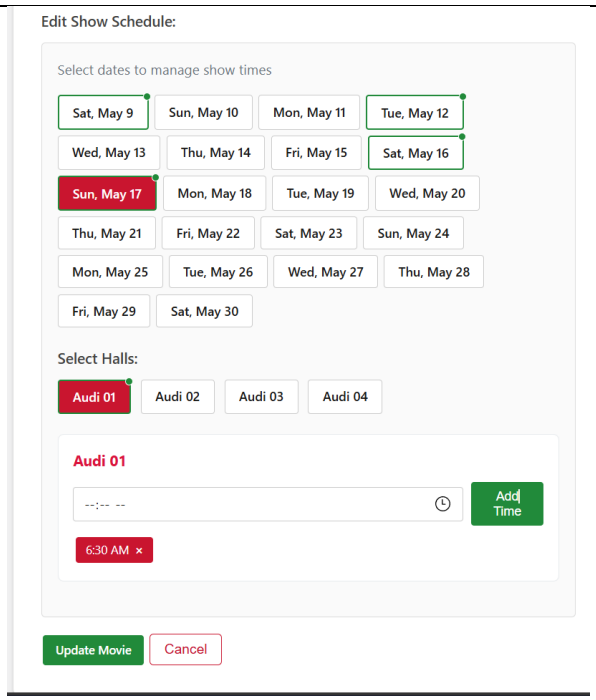
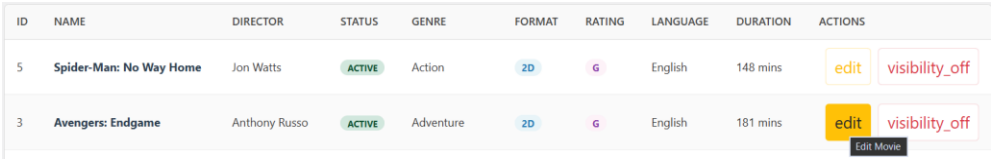
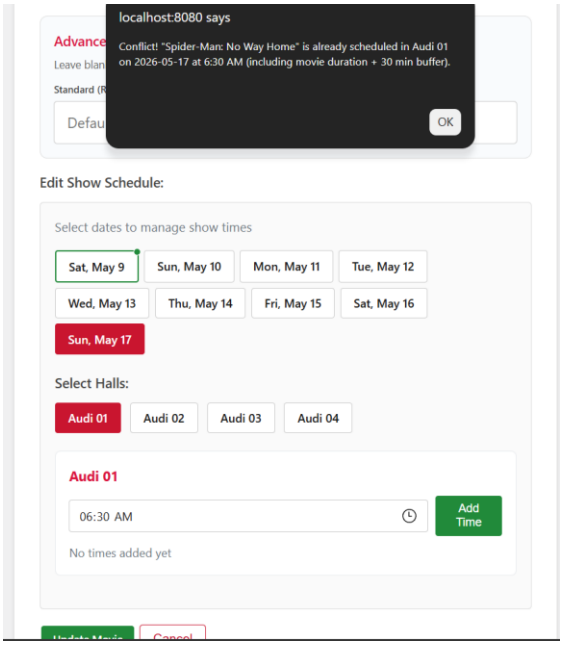
Field	Description
Test Name	Weak Password Rejection
Given Input	User registers with a single-character password like "1".
Expected Outcome	System should require a minimum length for security.
Actual Outcome	
Test Result	**Passed**

Table 23: Theater Hall Conflict Check

Field	Description
Test Name	Functional Test - Hall Overlap Prevention
Given Input	Admin tries to schedule a movie in a hall that is already booked for that time slot.
Expected Outcome	The system detects the time conflict and prevents the new showtime from being saved.
Actual Outcome	 

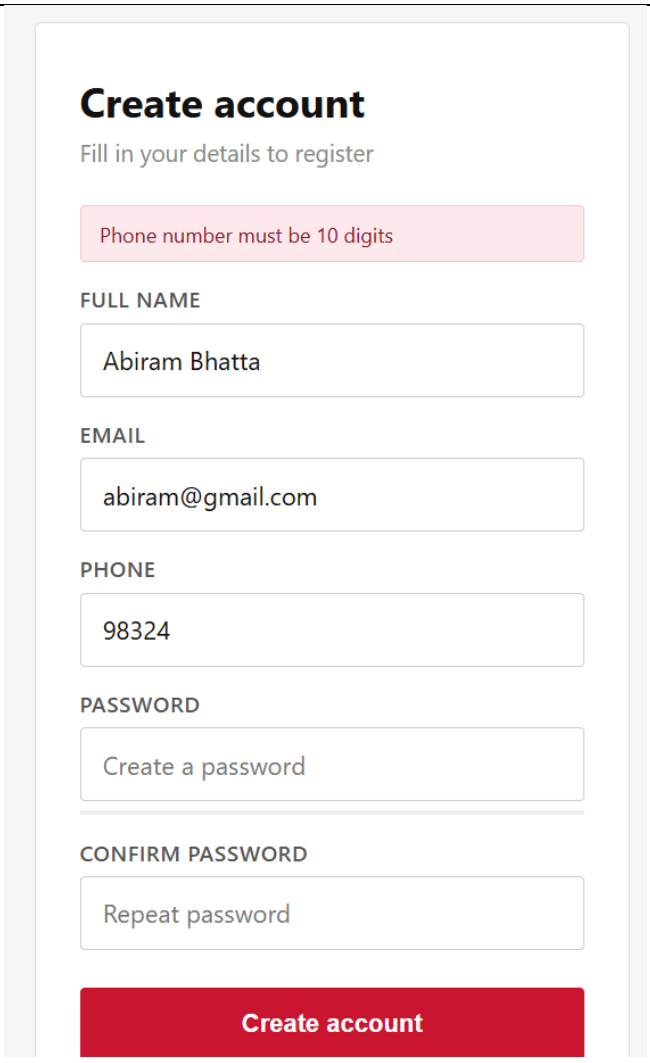






Test Result	**Passed**

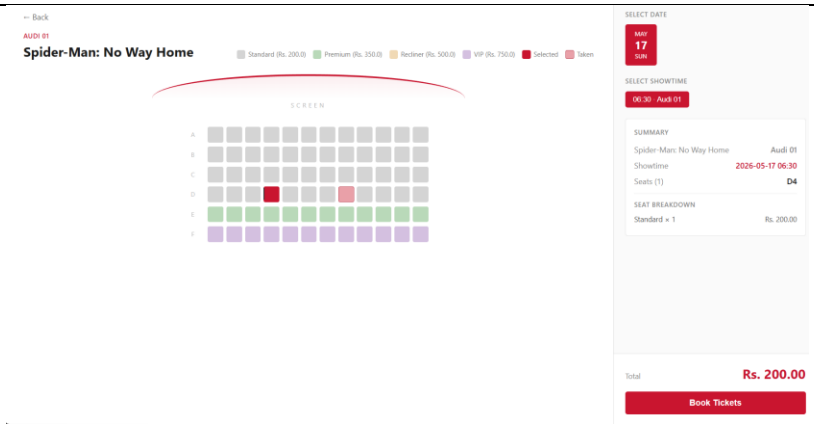
Table 24: Mobile Number Digit Count

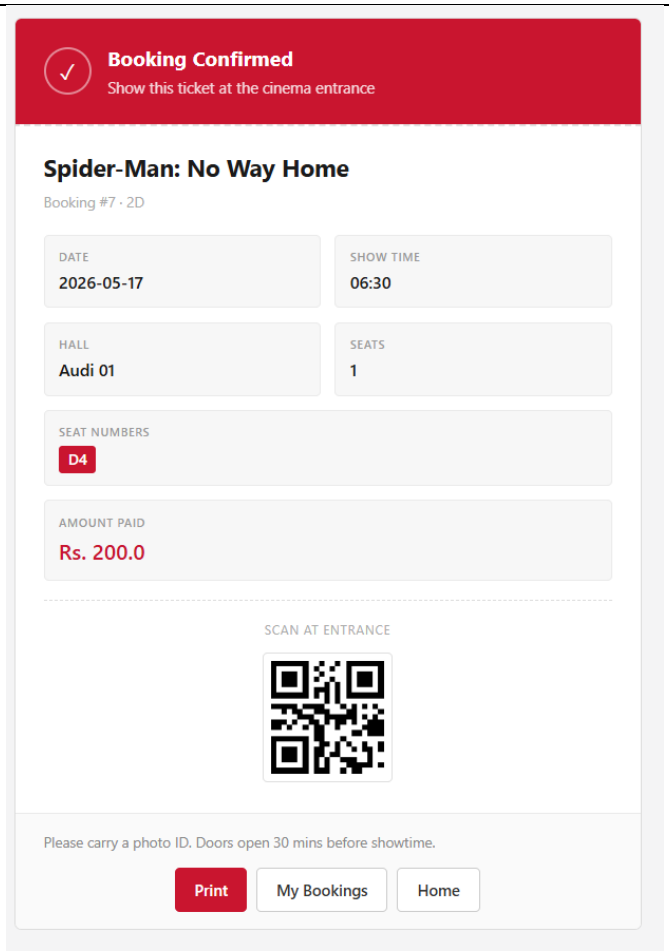
Field	Description
Test Name	Phone Number Length Validation
Given Input	User enters a 5-digit number in the phone field during registration.
Expected Outcome	The system should reject the input as it doesn't match standard 10-digit mobile formats.
Actual Outcome	
Test Result	**Passed**

5.5 Database & Persistence Testing

This section ensures your data stays safe, accurate, and survives technical hiccups.

Table 25: Booking Data Storage

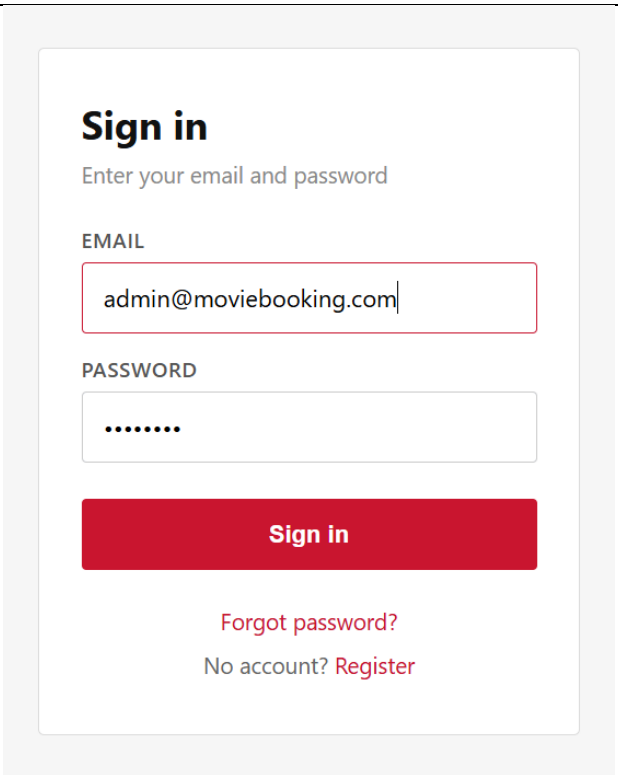
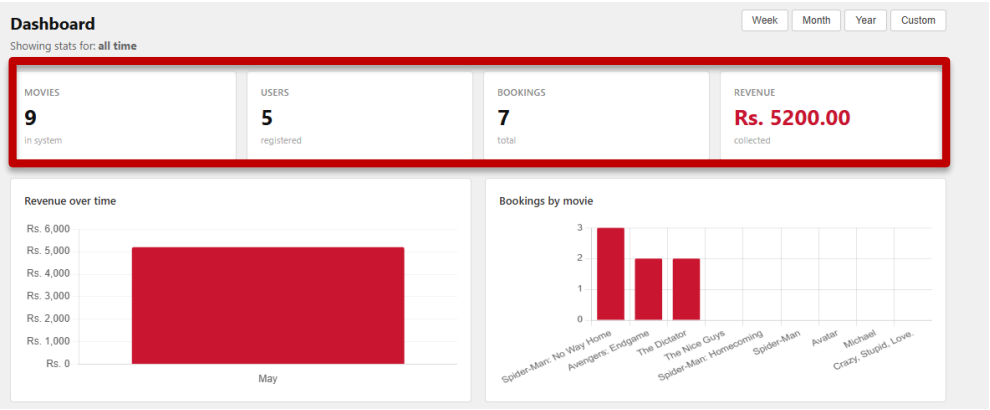
Field	Description
Test Name	Booking Record Insertion
Given Input	User books a seat for a movie and confirms ticket
Expected Outcome	Booking details should be stored in the database successfully
Actual Outcome	

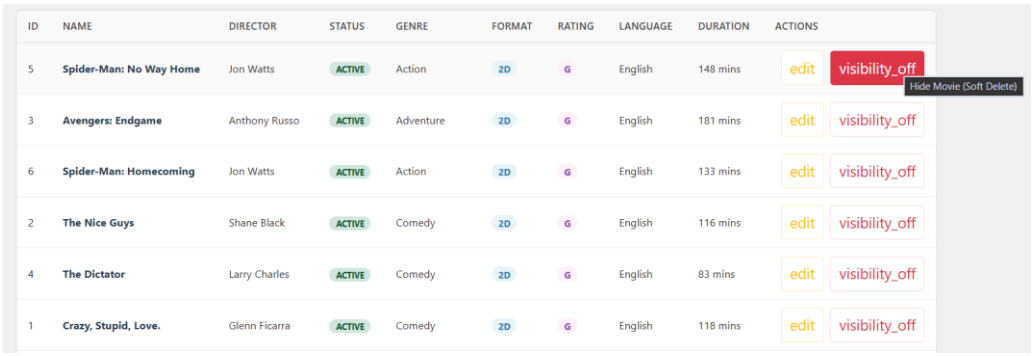


booking_id	user_id	movie_id	show_time	number_of_seats	seat_type	total_price	status	booking_date
1	2	3	2026-05-09 10:00 - Audi 01	1	E6	350	Completed	2026-05-08 08:47:13
2	2	3	2026-05-09 10:00 - Audi 01	9	E3, E5, E4, E2, D3, D4, D5, D6, D7	2400	Completed	2026-05-08 09:25:02
3	2	4	2026-05-11 00:25 - Audi 03	2	D5, E6	600	Completed	2026-05-10 00:19:50
4	4	4	2026-05-11 00:25 - Audi 03	2	D8, F10	1250	Completed	2026-05-10 08:33:24
5	2	5	2026-05-12 10:30 - Audi 01	1	D5	200	Completed	2026-05-11 13:37:40
6	5	5	2026-05-17 06:30 - Audi 01	1	D8	200	Completed	2026-05-16 22:26:19
7	5	5	2026-05-17 06:30 - Audi 01	1	D4	200	Completed	2026-05-16 22:32:39

Test Result	Passed

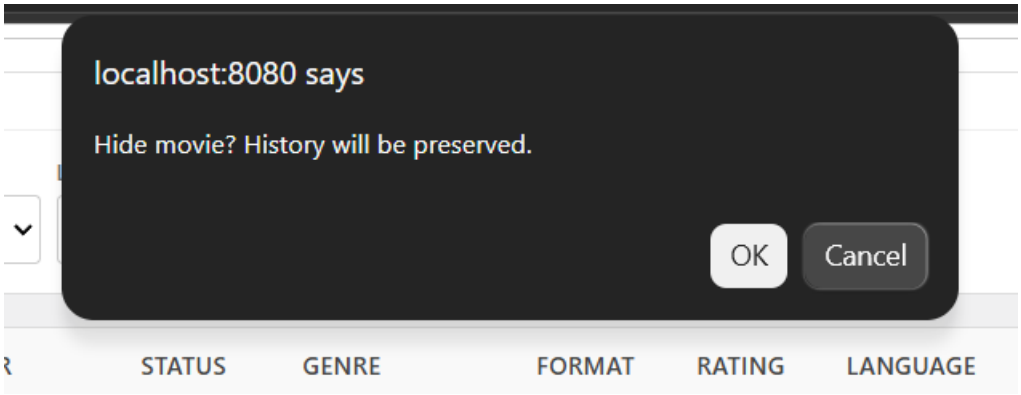
Table 26: Movie Deletion (Soft-Delete)

Field	Description
Test Name	Data Integrity During Movie Removal
Given Input	Admin deletes a movie that already has historical bookings.
Expected Outcome	The movie should be hidden from the site, but the bookings must NOT be deleted (for financial records).
Actual Outcome	 



ID	NAME	DIRECTOR	STATUS	GENRE	FORMAT	RATING	LANGUAGE	DURATION	ACTIONS
5	Spider-Man: No Way Home	Jon Watts	ACTIVE	Action	2D	G	English	148 mins	edit visibility_off
3	Avengers: Endgame	Anthony Russo	ACTIVE	Adventure	2D	G	English	181 mins	edit visibility_off
6	Spider-Man: Homecoming	Jon Watts	ACTIVE	Action	2D	G	English	133 mins	edit visibility_off
2	The Nice Guys	Shane Black	ACTIVE	Comedy	2D	G	English	116 mins	edit visibility_off
4	The Dictator	Larry Charles	ACTIVE	Comedy	2D	G	English	83 mins	edit visibility_off
1	Crazy, Stupid, Love.	Glenn Ficarra	ACTIVE	Comedy	2D	G	English	118 mins	edit visibility_off

Hide Movie (Soft Delete)



5	Spider-Man: No Way Home	Jon Watts	HIDDEN	Action	2D	G	English	148 mins	edit restore
3	Avengers: Endgame	Anthony Russo	ACTIVE	Adventure	2D	G	English	181 mins	edit visibility_off
6	Spider-Man: Homecoming	Jon Watts	ACTIVE	Action	2D	G	English	133 mins	edit visibility_off
2	The Nice Guys	Shane Black	ACTIVE	Comedy	2D	G	English	116 mins	edit visibility_off
4	The Dictator	Larry Charles	ACTIVE	Comedy	2D	G	English	83 mins	edit visibility_off

booking_id	user_id	movie_id	show_time	number_of_seats	seat_type	total_price	status	booking_date
1	2	3	2026-05-09 10:00 - Audi 01	1	E6	350	Completed	2026-05-08 08:47:13
2	2	3	2026-05-09 10:00 - Audi 01	9	E3, E5, E4, E2, D3, D4, D5, D6, D7	2400	Completed	2026-05-08 09:25:02
3	2	4	2026-05-11 00:25 - Audi 03	2	D5, E6	600	Completed	2026-05-10 00:19:50
4	4	4	2026-05-11 00:25 - Audi 03	2	D8, F10	1250	Completed	2026-05-10 08:33:24
5	2	5	2026-05-12 10:30 - Audi 01	1	D5	200	Completed	2026-05-11 13:37:40
6	5	5	2026-05-17 06:30 - Audi 01	1	D8	200	Completed	2026-05-16 22:26:19
7	5	5	2026-05-17 06:30 - Audi 01	1	D4	200	Completed	2026-05-16 22:32:39

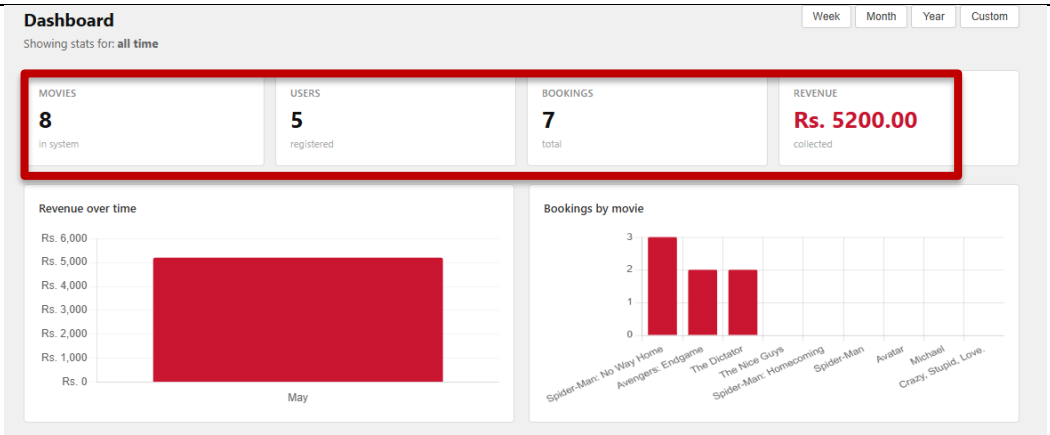
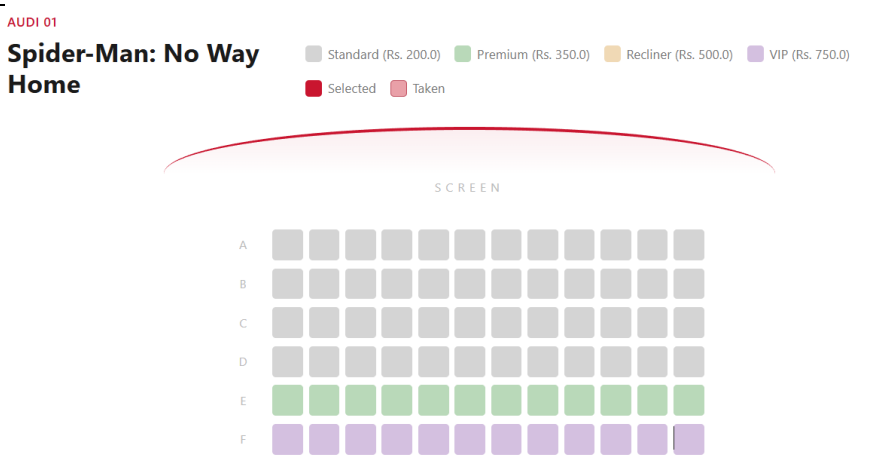
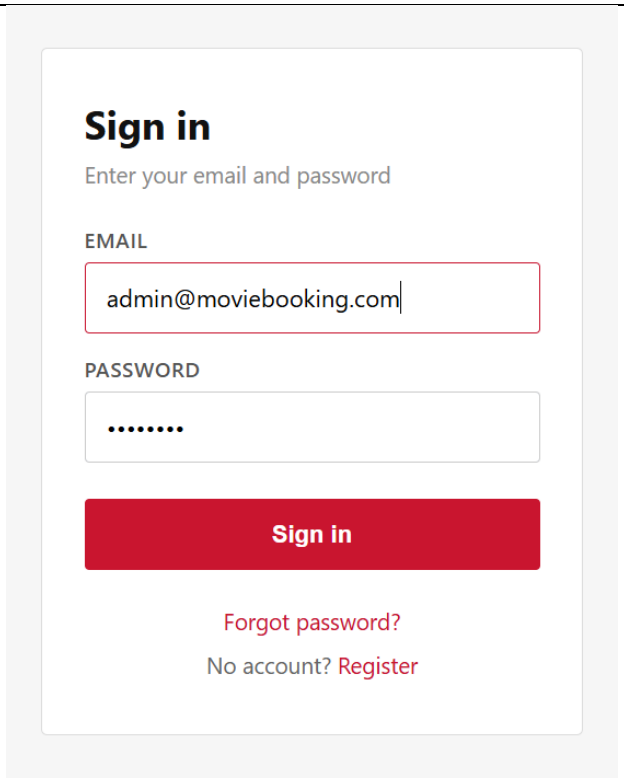
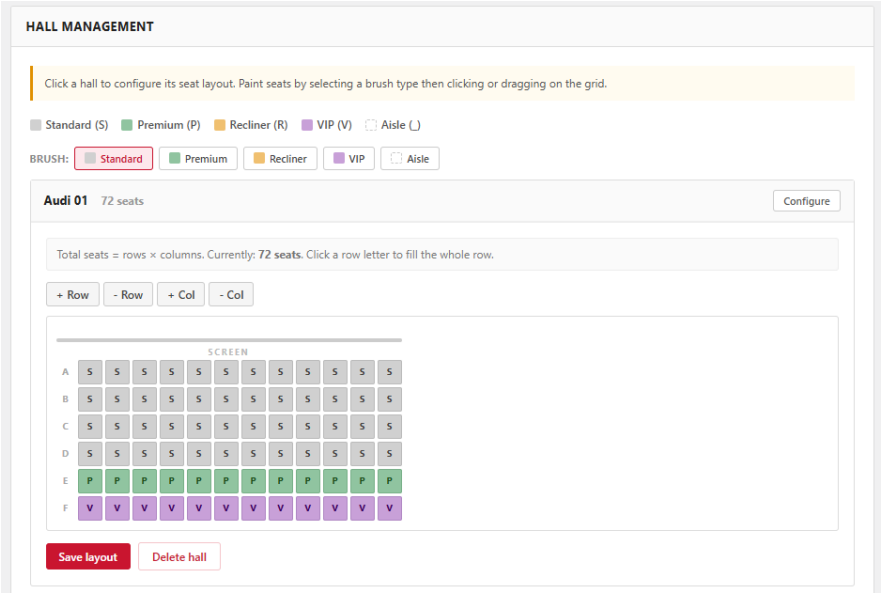
	
Test Result	**Passed**

Table 27: Dynamic Hall Configuration

Field	Description
Test Name	Hall Layout Persistence
Given Input	Admin changes Hall 1 from 50 seats to 60 seats in the dashboard.
Expected Outcome	The new layout should be immediately reflected for all future bookings of that hall.
Actual Outcome	


Sign in
Enter your email and password
EMAIL
admin@moviebooking.com
PASSWORD
••••••••
Sign in
Forgot password?
No account? Register


HALL MANAGEMENT
Click a hall to configure its seat layout. Paint seats by selecting a brush type then clicking or dragging on the grid.
Standard (S) Premium (P) Recliner (R) VIP (V) Aisle (_)
BRUSH: Standard Premium Recliner VIP Aisle
Audi 01 72 seats
Configure
Total seats = rows × columns. Currently: 72 seats. Click a row letter to fill the whole row.
+ Row - Row + Col - Col
SCREEN
Save layout
Delete hall

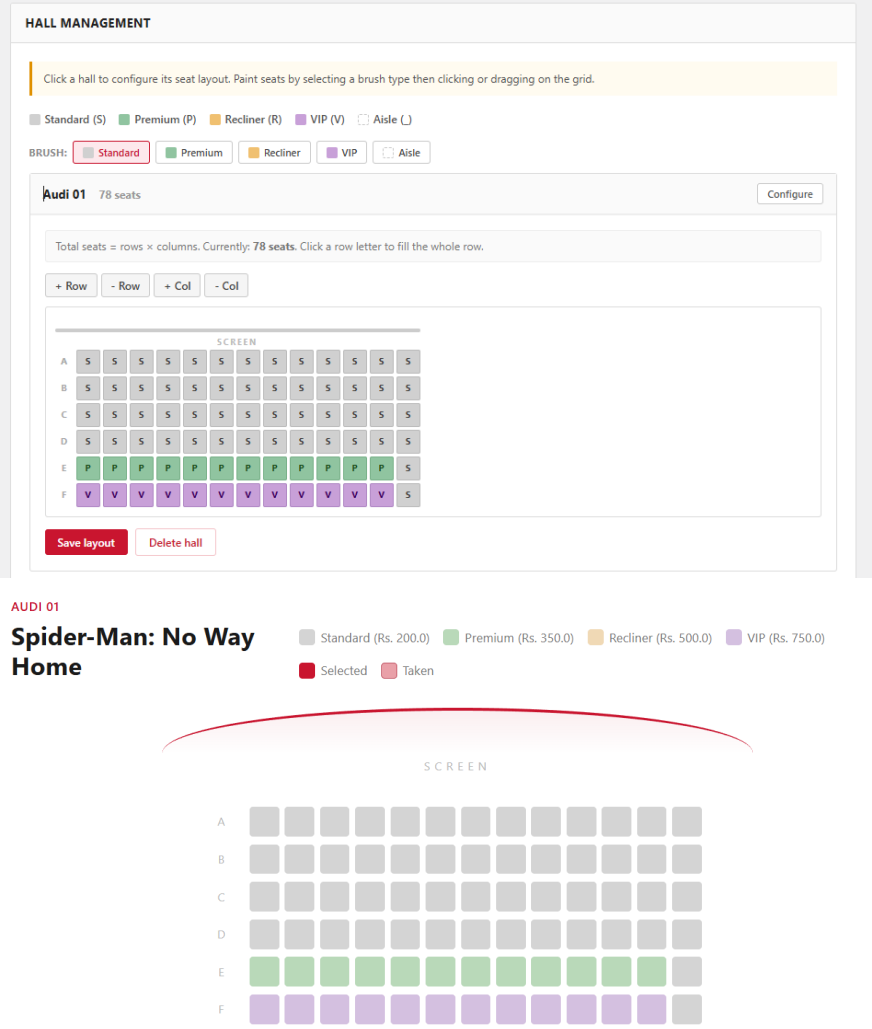
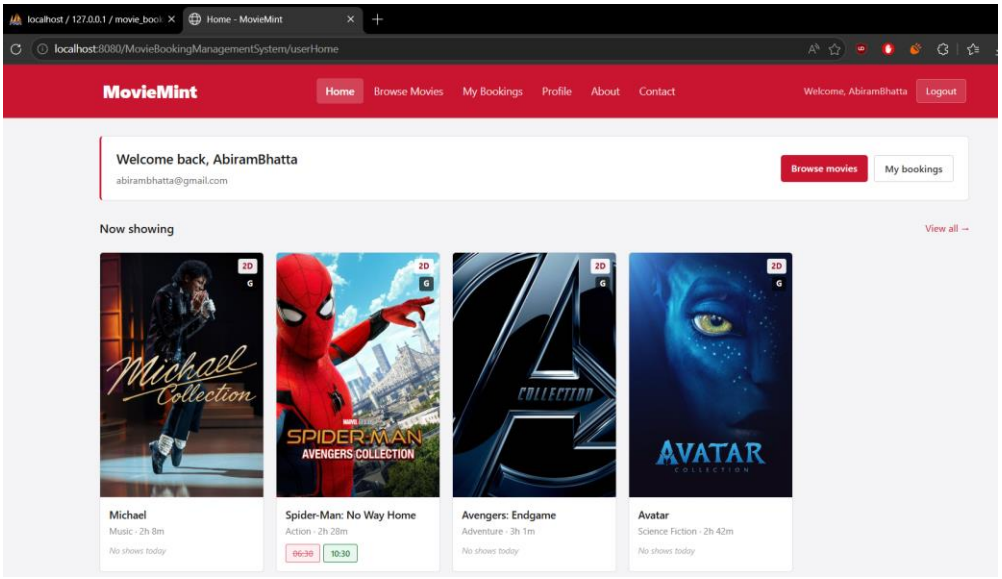
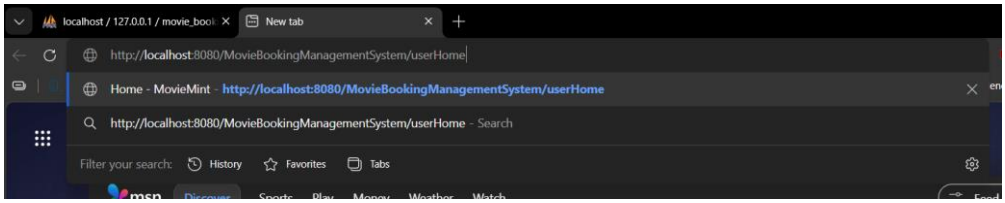
	
Test Result	**Passed**

Table 28: Session Persistence

Field	Description
Test Name	Login Session Survival
Given Input	User logs in, closes the browser tab, and re-opens the website.
Expected Outcome	If the session is still active, the user should remain logged in without needing to type the password again.
Actual Outcome	 

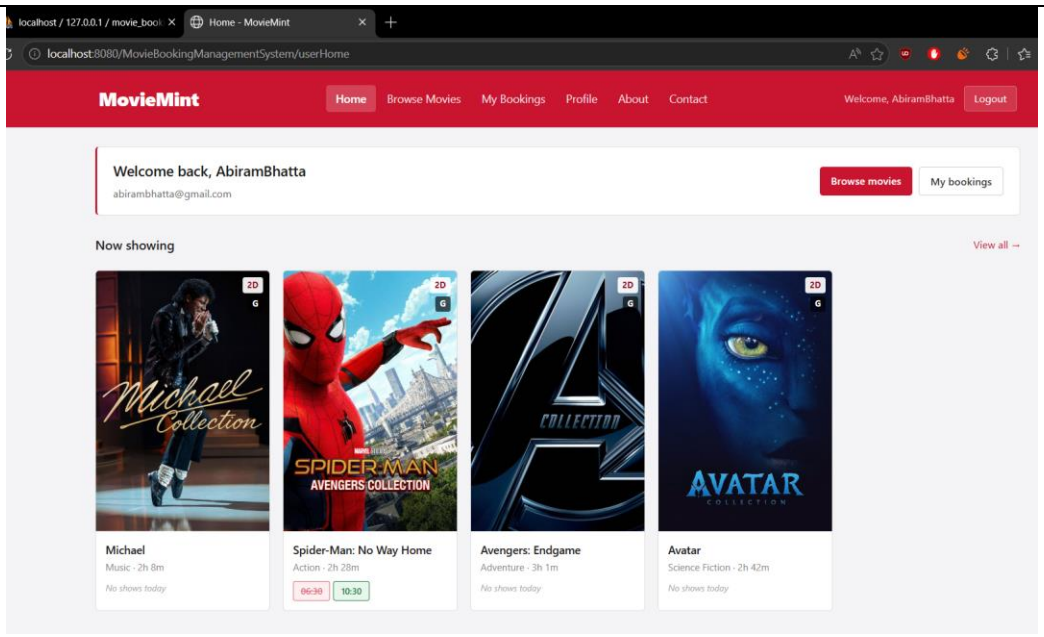
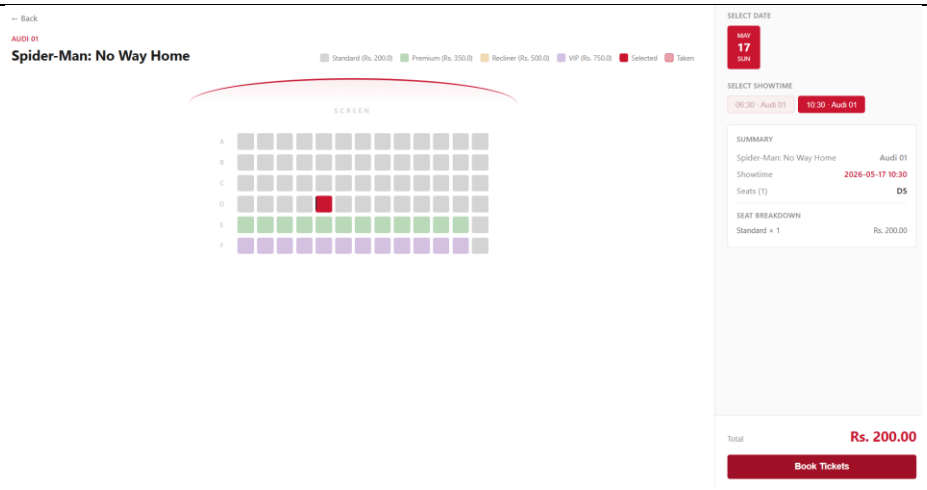
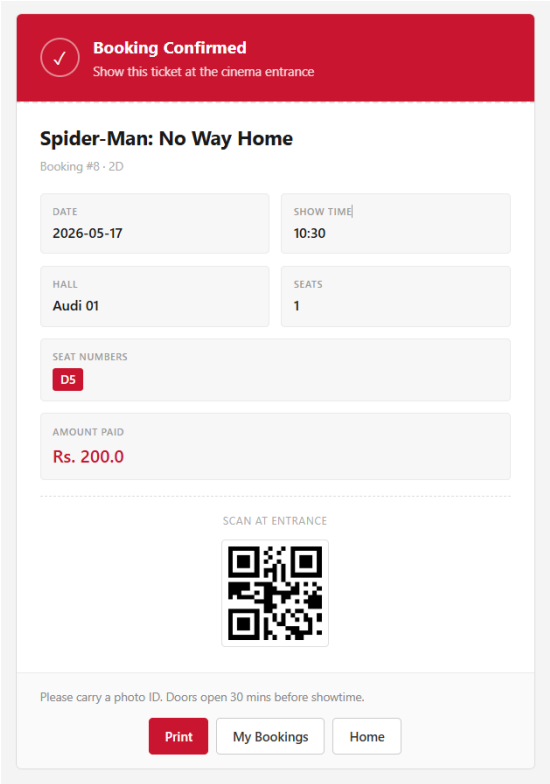
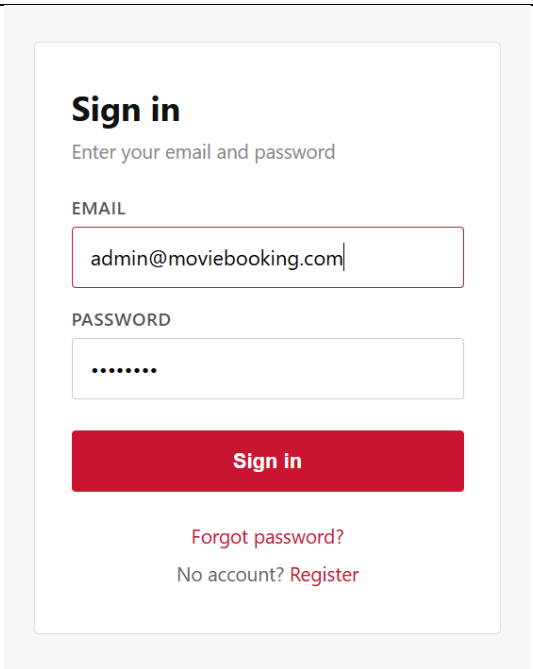
	
Test Result	**Passed**

Table 29: Booking Deletion

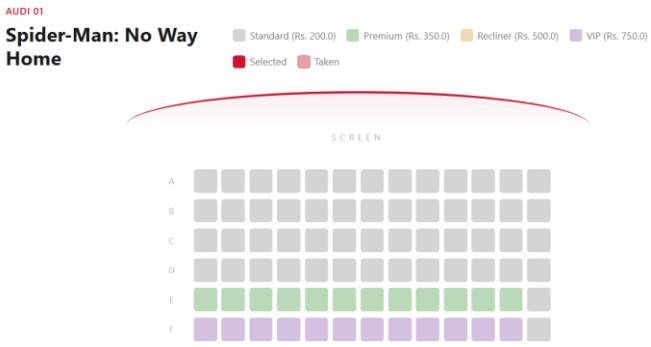
Field	Description
Test Name	Cancel Booking Record
Given Input	User cancels an existing booking
Expected Outcome	Booking record should be removed or marked cancelled

Actual Outcome	 



BOOKING ID	USER	MOVIE	BOOKING DATE	SHOW TIME	SEATS	TOTAL PRICE	STATUS	ACTIONS
8	AbiramBhatta	Spider-Man: No Way Home	17 May 2026 14:49	2026-05-17 10:30 - Audi 01	1	Rs. 200.00	CANCELLED	Cancelled Del

booking_id	user_id	movie_id	show_time	number_of_seats	seat_type	total_price	status	booking_date
1	2	3	2026-05-09 10:00 - Audi 01	1	E6	350	Completed	2026-05-08 08:47:13
2	2	3	2026-05-09 10:00 - Audi 01	9	E3, E5, E4, E2, D3, D4, D5, D6, D7	2400	Completed	2026-05-08 09:25:02
3	2	4	2026-05-11 00:25 - Audi 03	2	D5, E6	600	Completed	2026-05-10 00:19:50
4	4	4	2026-05-11 00:25 - Audi 03	2	D8, F10	1250	Completed	2026-05-10 08:33:24
5	2	5	2026-05-12 10:30 - Audi 01	1	D5	200	Completed	2026-05-11 13:37:40
6	5	5	2026-05-17 06:30 - Audi 01	1	D8	200	Completed	2026-05-16 22:26:19
7	5	5	2026-05-17 06:30 - Audi 01	1	D4	200	Completed	2026-05-16 22:32:39
8	5	5	2026-05-17 10:30 - Audi 01	1	D5	200	Cancelled	2026-05-17 09:04:50

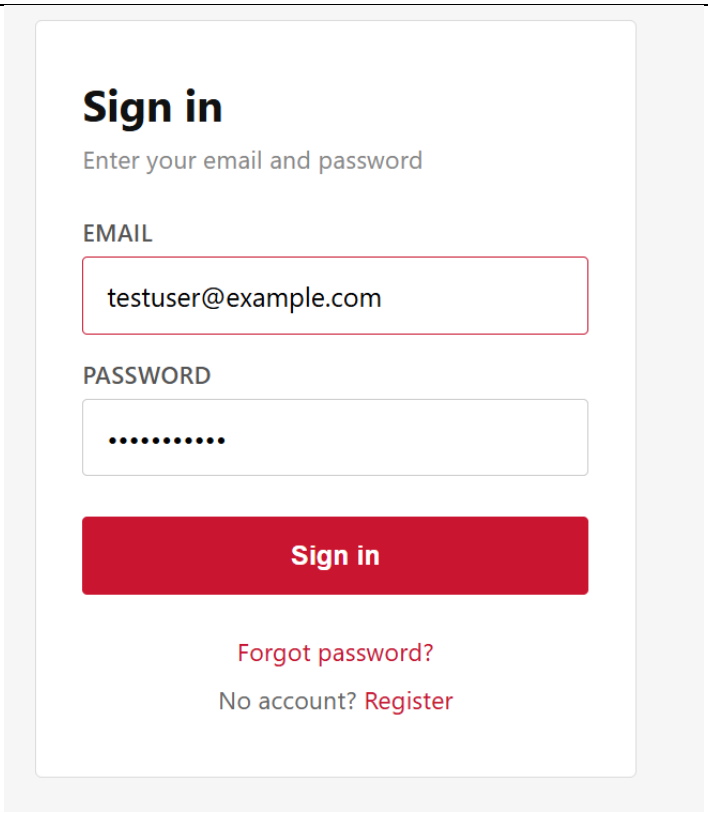


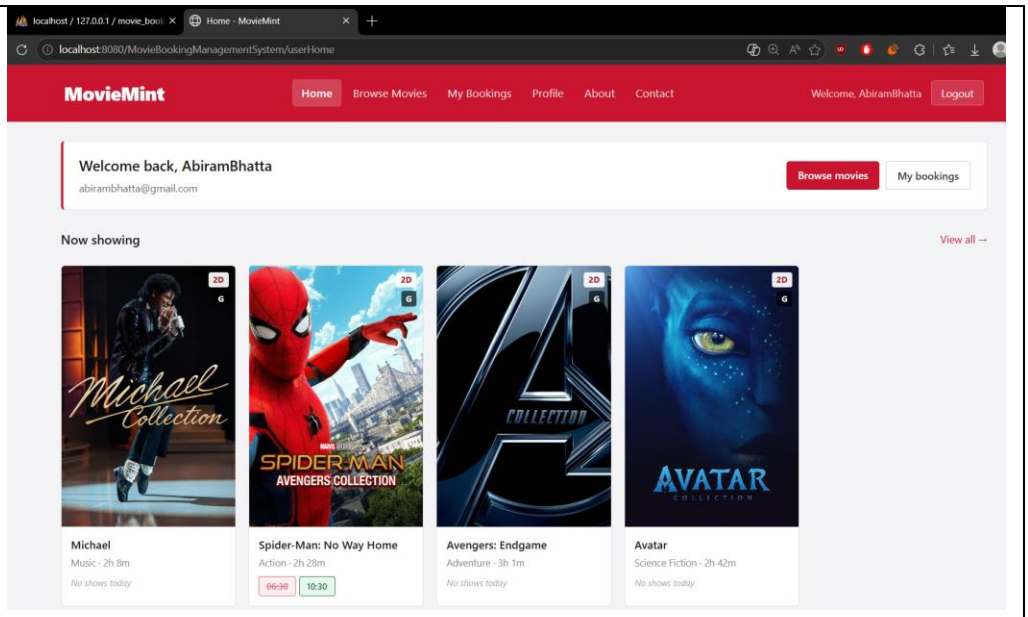
Test Result	Passed

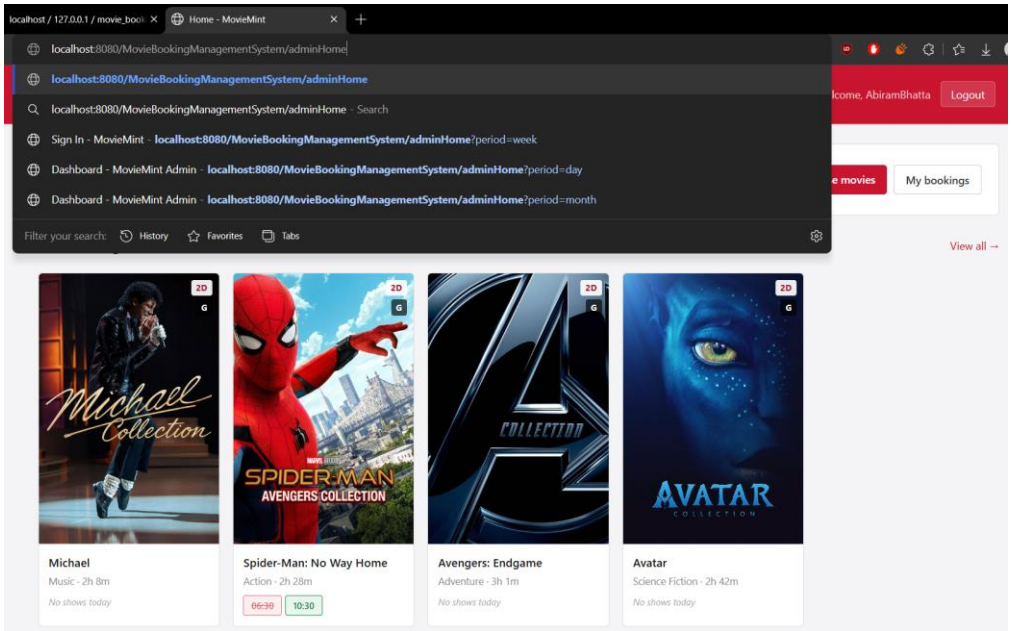
5.6 Security & Access Control Testing

Tables 30 to 34 examine the protection of restricted routes, authenticated sessions, password storage, OTP handling, and role-based authorisation.

Table 30: Security Test - Role-Based Access Control URL Forcing

Field	Description
Test Name	Security Test - Role-Based Access Control URL Forcing
Given Input	A normal user manually enters /adminHome, /manageMovies, or another admin route into the browser address bar.
Expected Outcome	The authentication filter should block access and redirect the user to the appropriate login or unauthorised access flow.
Actual Outcome	


localhost / 127.0.0.1 / movie_book
Home - MovieMint
localhost:8080/MovieBookingManagementSystem/userHome
MovieMint
Home
Browse Movies
My Bookings
Profile
About
Contact
Welcome, AbiramBhatta
Logout
Welcome back, AbiramBhatta
abirambhatta@gmail.com
Browse movies
My bookings
Now showing
View all →
Michael
Music · 2h 8m
No shows today
Spider-Man: No Way Home
Action · 2h 28m
06:30
10:30
Avengers: Endgame
Adventure · 3h 1m
No shows today
Avatar
Science Fiction · 2h 42m
No shows today


localhost:8080/MovieBookingManagementSystem/adminHome
localhost:8080/MovieBookingManagementSystem/adminHome
localhost:8080/MovieBookingManagementSystem/adminHome - Search
Sign In - MovieMint - localhost:8080/MovieBookingManagementSystem/adminHome?period=week
Dashboard - MovieMint Admin - localhost:8080/MovieBookingManagementSystem/adminHome?period=day
Dashboard - MovieMint Admin - localhost:8080/MovieBookingManagementSystem/adminHome?period=month
Filter your search
History
Favorites
Tabs
My bookings
View all →
Michael
Music · 2h 8m
No shows today
Spider-Man: No Way Home
Action · 2h 28m
06:30
10:30
Avengers: Endgame
Adventure · 3h 1m
No shows today
Avatar
Science Fiction · 2h 42m
No shows today

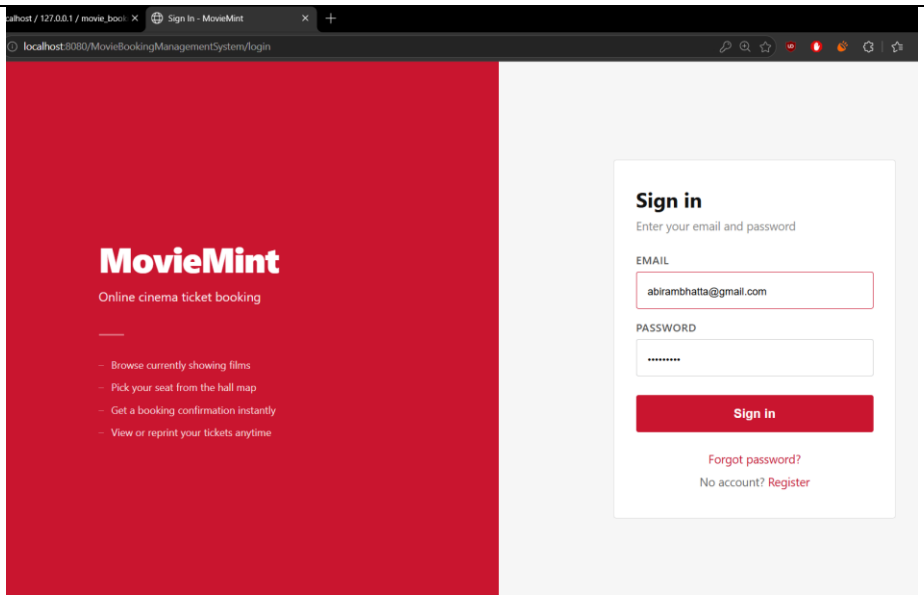
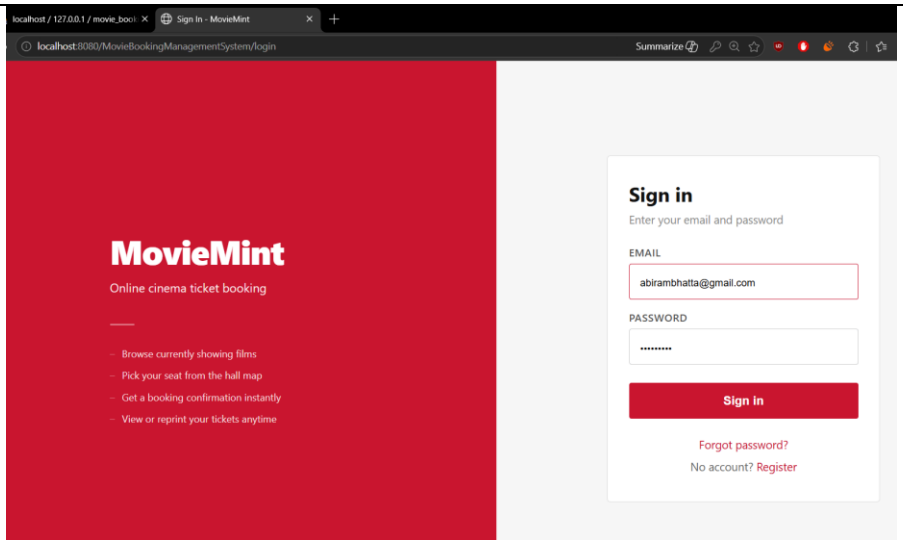
	
Test Result	Passed

Table 31: Security Test - Unauthenticated Protected Route Access

Field	**Description**
Test Name	Security Test - Unauthenticated Protected Route Access
Given Input	A logged-out visitor attempts to access /bookTicket, /myBookings, or /userProfile directly.
Expected Outcome	The system should detect the missing authenticated session and redirect the request to the login page.
Actual Outcome	

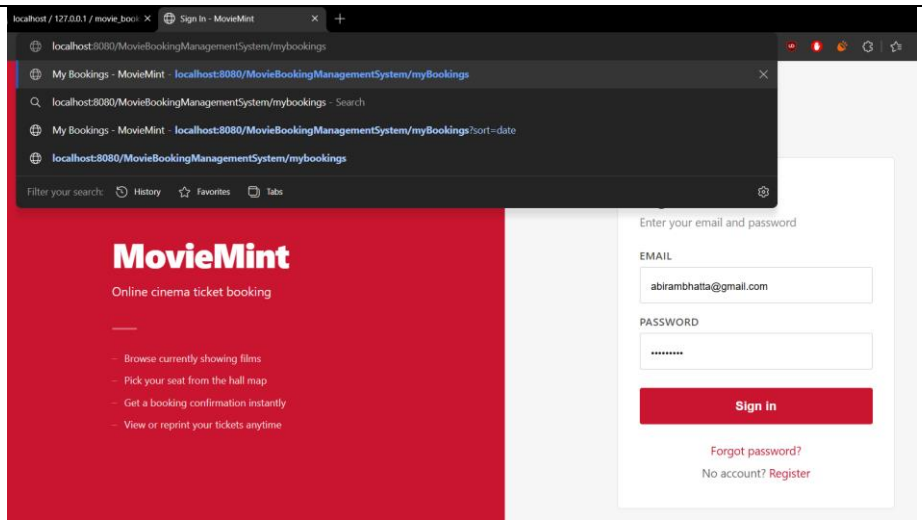
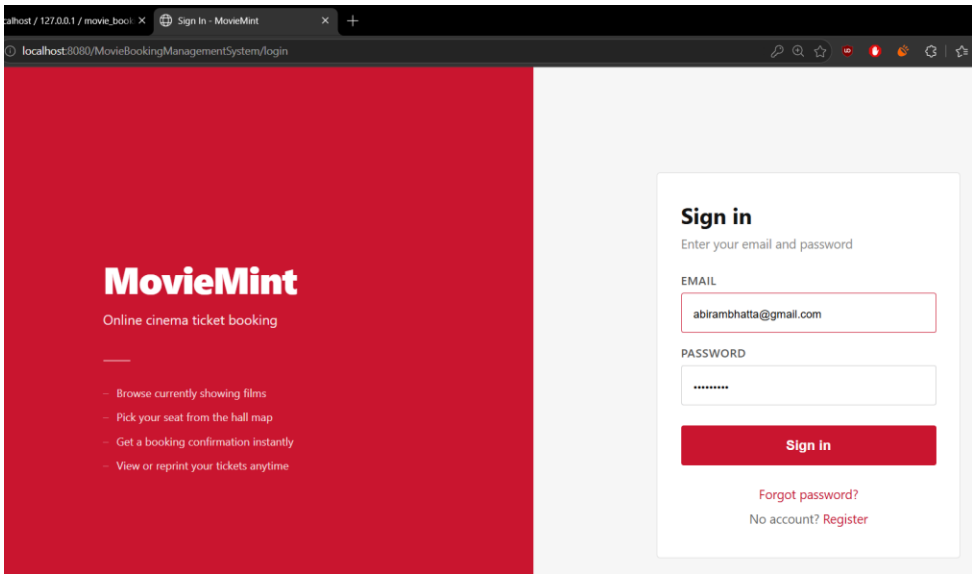
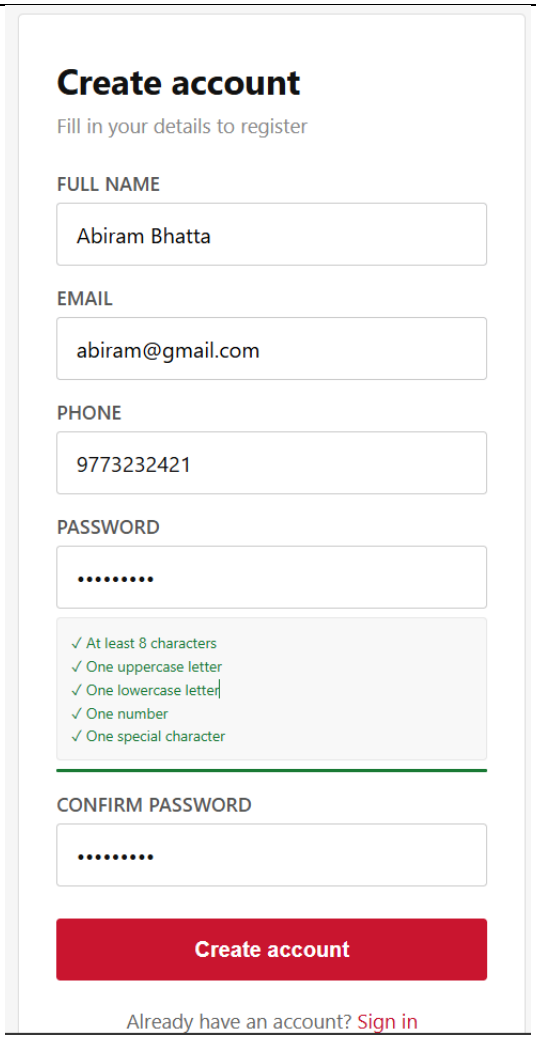
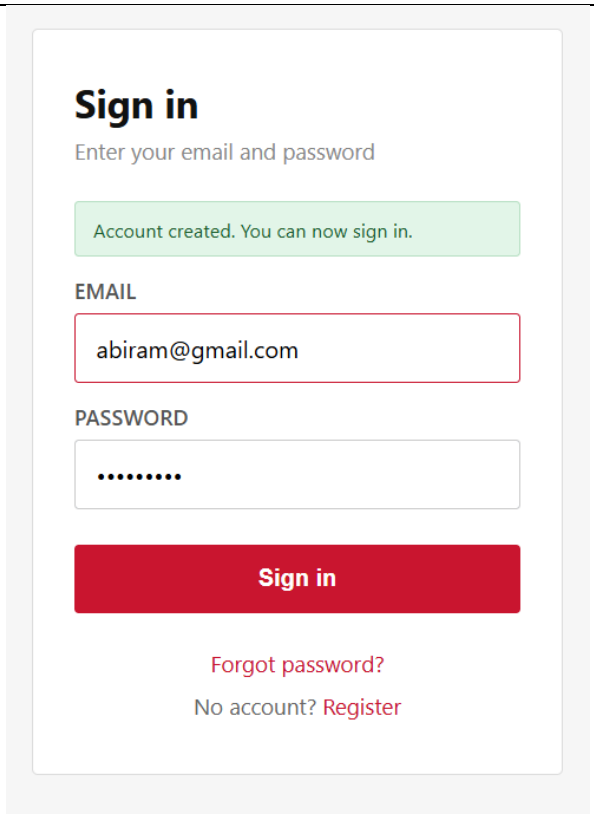
	 
Test Result	Passed

Table 32: Security Test - Secure Password Hashing Validation

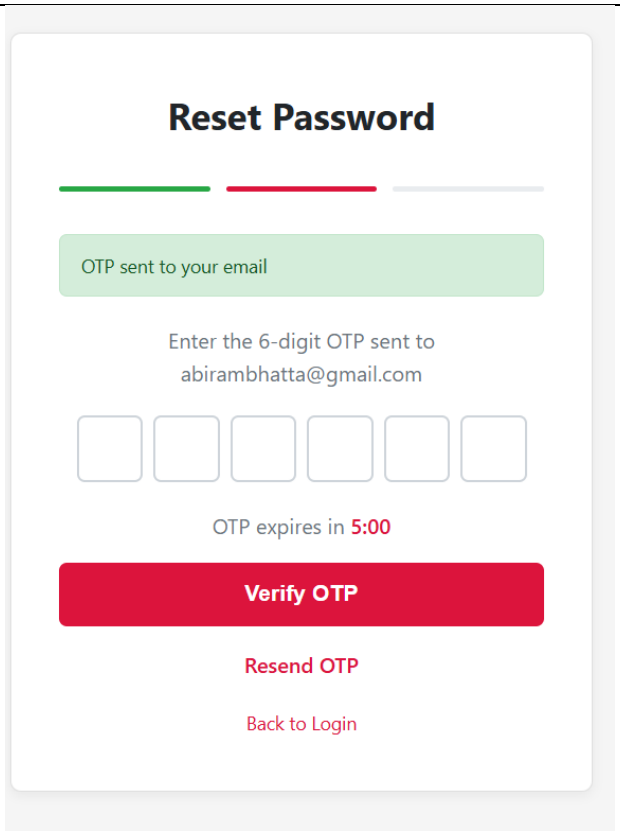
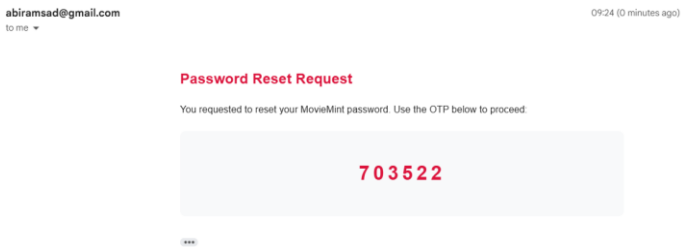
Field	Description
Test Name	Security Test - Secure Password Hashing Validation
Given Input	A new user registers with a valid password and the database record is inspected after registration.
Expected Outcome	The password should not be stored in plain text. The stored password value should be hashed or securely transformed before persistence.
Actual Outcome	

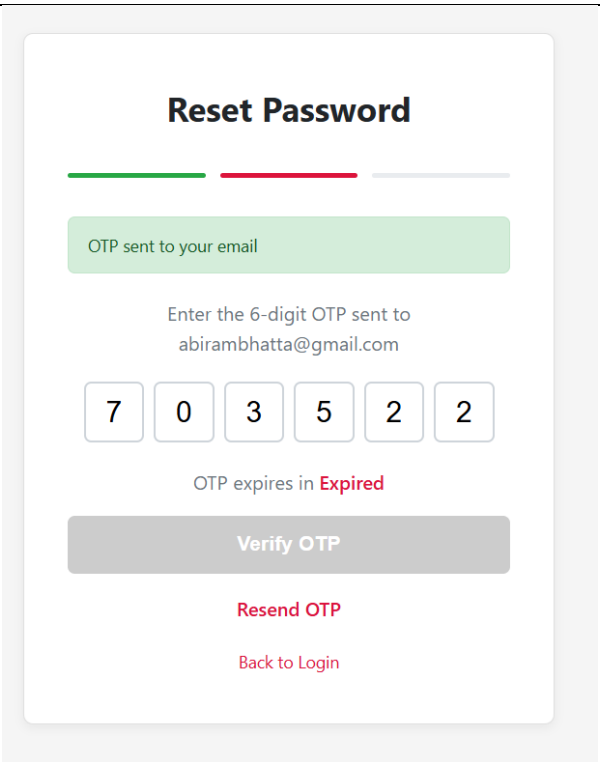


user_id	full_name	email	phone	password	role	created_at	reset_otp	otp_expiry
1	Super Admin	admin@moviebooking.com	0000000000	Gcdku5TGKHuw5nytUa5s3X2Ei7S4VWRLQUwKXMzMXcMObzqu2K...	admin	2026-05-07 19:15:32	NULL	NULL
2	Test User	testuser@example.com	9763690279	CKAOMkcvmCv3za8A7tLqyukVcJ/IxuXo61RVhCCpmzNiM9ADBw...	user	2026-05-07 20:00:20	NULL	NULL
3	Super Admin	admin@moviemint.com	9999999999	4jjQDcDmhsAiYRelAuCwLSUH4oinOuuwAcSQd1CK9P0iwcAEjz...	admin	2026-05-08 09:39:58	NULL	NULL
4	Abiram Bhatta	abiramsad@gmail.com	9843596660	VKidmnP9Meg86WkvEKSU6MuPL6ajKL3dnpXmlDZJzBlVeHJ5oE...	user	2026-05-10 08:32:52	193465	2026-05-11 10:10:25
5	AbiramBhatta	abirambhatta@gmail.com	9843434212	Sulw/DqLcdrRZ1eFt/F7CqUAf+aNmGL5R2PnxlAQirOgoThQCv...	user	2026-05-16 22:20:17	235846	2026-05-16 22:25:44
6	Abiram Bhatta	abiram@gmail.com	9773232421	sgx52vn40lWdE6J+90s7Oldx22dEX+E1J28uUsoy2xlylel9V2...	user	2026-05-17 09:20:58	NULL	NULL

Test Result	Passed

Table 33: Security Test - OTP Expiry Enforcement

Field	Description
Test Name	Security Test - OTP Expiry Enforcement
Given Input	User requests a password reset OTP and attempts to use it after the configured expiry time has passed.
Expected Outcome	The expired OTP should be rejected, and the user should be required to request a new code before resetting the password.
Actual Outcome	 

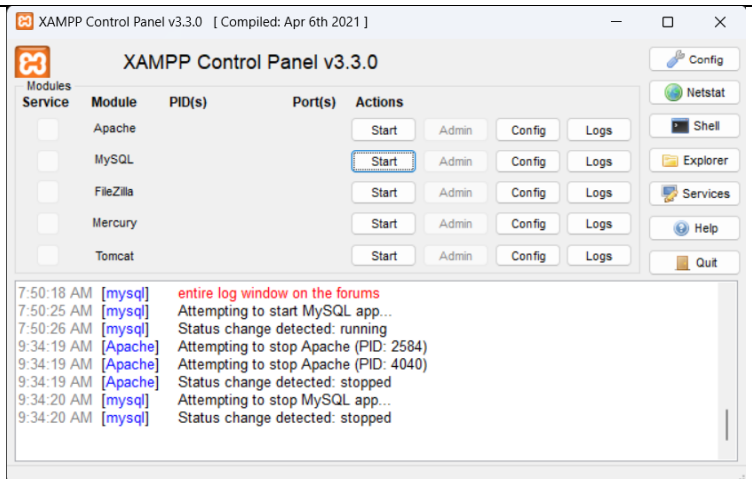
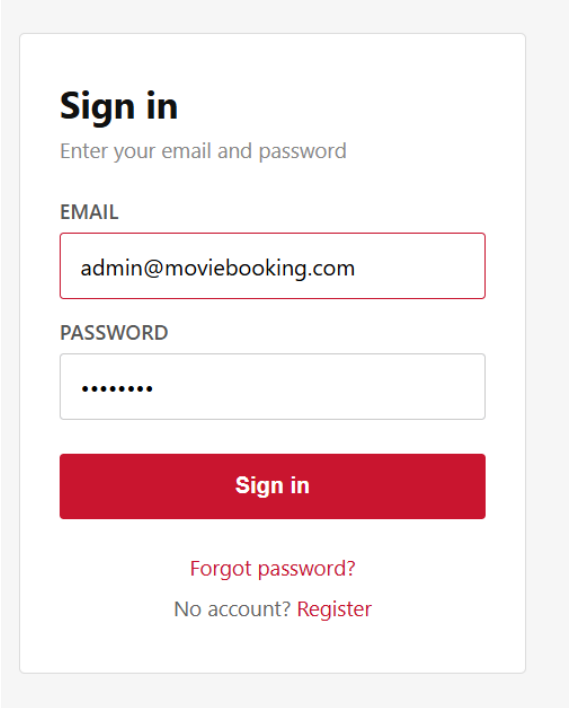
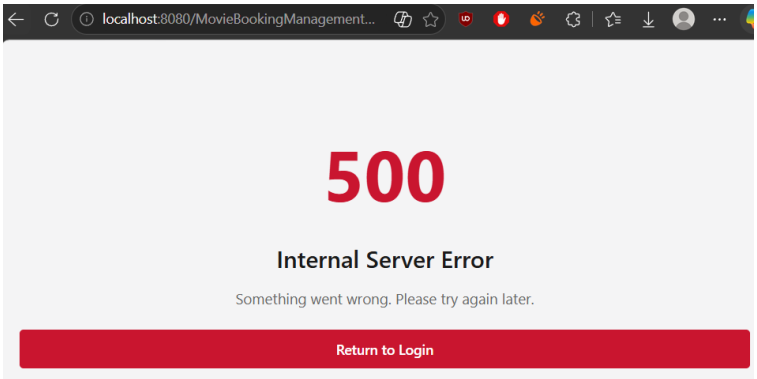
	
Test Result	Passed

5.7 Exception & Error Handling Testing

Tables 34 verify that unexpected faults are handled in a controlled manner without exposing stack traces, internal SQL errors, or unstable user interface behaviour.

Table 34: Exception Test - Graceful Database Disconnection Handling

Field	Description
Test Name	Exception Test - Graceful Database Disconnection Handling
Given Input	The MySQL server is stopped or database connection credentials are temporarily invalid.
Expected Outcome	The application should not expose raw stack traces to the user. A controlled error message or error page should be displayed while the exception is handled internally.

Actual Outcome	  
Test Result	Passed

6. Coursework Development

6.1 Development Tools

- **Java 21:** Programming platform used to write model classes, DAO classes, servlet controllers, filters, services, and utility classes. The repository README identifies Java 21 or higher as the required runtime.
- **Jakarta EE Servlets:** Server-side web technology used for handling HTTP requests and responses. It is used for login, registration, browsing, booking, password reset, dashboard, management, and support routes.
- **JSP and JSTL:** Dynamic view technologies used to render server-provided data inside customer and administrator pages. JSP files are kept inside WEB-INF/pages so they are served through controllers rather than direct browser access.
- **MySQL 8.0+:** Relational database management system used to store users, movies, show times, bookings, hall configuration, and system settings.
- **JDBC:** Java database connectivity API used by DAO classes to execute SQL queries, inserts, updates, joins, aggregations, and status changes.
- **Apache Tomcat 10.1+:** Java web application server used to deploy and run the servlet/JSP application.
- **HTML, CSS, and Vanilla JavaScript:** Frontend technologies used to design responsive forms, navigation, movie cards, dashboards, tables, filters, ticket views, and interactive seat selection.
- **Chart.js:** Browser-based charting library used for admin revenue charts, booking charts, movie performance charts, and seat distribution analytics.
- **TMDB API:** External movie metadata source used to auto-fill movie details, reduce manual data entry, and support poster/trailer data during admin movie entry.
- **JavaMail with Gmail SMTP:** Email support used for OTP messages and booking confirmation communication. The EmailService class uses SMTP configuration and TLS settings.

- **iTextPDF and XMLWorker:** PDF-related libraries used with the HTML-to-PDF ticket conversion workflow, supporting digital ticket output and attachment handling.
- **Git and GitHub:** Version control and repository hosting tools used to store source code, SQL schema, documentation, setup instructions, and submission evidence.
- **Environment configuration:** The repository includes .env.example, and EnvLoader reads database, email, and TMDB credentials from .env or environment variables. This improves deployment safety because credentials are kept outside the Java source code.
- **Project file organisation:** The implemented project uses a standard Maven-style web application layout. Java source files are placed under src/main/java/com/moviebooking and are grouped into config, controllers, dao, filter, model, service, and util packages. Web resources are placed under src/main/webapp, including WEB-INF/pages for JSP views, CSS, JavaScript, images, index.jsp, DATABASE_SCHEMA.sql, README.md, and .env.example.

6.2 Database Design

The database is designed around six main tables: users, movies, show_times, bookings, hall_config, and system_settings. Users and movies are core master records. Bookings are transactional records. Show times represent scheduled movie slots. Hall configuration stores seat layout and category mapping. System settings store global values such as ticket prices and available halls.

6.2.1 Data Dictionary

The data dictionary below documents the database fields used by the MovieMint system. Each table identifies the field name, data type, key role, nullability, and practical purpose within the application.

The users table stores customer and administrator account data, including authentication, authorization, and password-reset fields.

Table 35: Data Dictionary - users table

Field Name	Data Type	Key	Null	Description
user_id	INT	PK	No	Unique identifier for each user account.
full_name	VARCHAR(100)	-	No	Full name of the customer or administrator.
email	VARCHAR(100)	Unique	No	Email address used for login and communication.
phone	VARCHAR(20)	Unique	No	Contact number linked to the account.
password	VARCHAR(255)	-	No	Hashed account password.
role	VARCHAR(20)	-	Yes	Defines user or admin access level.
created_at	TIMESTAMP	-	Yes	Date and time when the

				account was created.
reset_otp	VARCHAR(6)	-	Yes	Temporary OTP used for password reset.
otp_expiry	TIMESTAMP	-	Yes	Expiry time for the password reset OTP.

The movies table stores catalogue information, showing windows, media references, pricing overrides, and the soft-delete status.

Table 36: Data Dictionary - movies table

Field Name	Data Type	Key	Null	Description
movie_id	INT	PK	No	Unique identifier for each movie.
title	VARCHAR(255)	-	No	Movie title displayed in the catalogue.
genre	VARCHAR(100)	-	Yes	Movie genre used for browsing and filtering.
director	VARCHAR(100)	-	Yes	Director name.
duration	INT	-	No	Movie duration in minutes.
language	VARCHAR(50)	-	Yes	Primary movie language.
release_date	DATE	-	Yes	Official release date.

start_date	DATE	-	No	First date the movie is available in the cinema.
end_date	DATE	-	No	Final date the movie is available in the cinema.
description	TEXT	-	Yes	Movie synopsis or description.
poster_image	VARCHAR(255)	-	Yes	Local or external poster image path.
format	VARCHAR(20)	-	Yes	Movie format such as 2D, 3D, or IMAX.
age_rating	VARCHAR(10)	-	Yes	Age classification for the movie.
trailer_url	VARCHAR(255)	-	Yes	Trailer embed or YouTube URL.
cast_list	TEXT	-	Yes	Cast information for the movie.
price_standard	DOUBLE	-	Yes	Optional standard seat price override.
price_premium	DOUBLE	-	Yes	Optional premium seat price override.

price_recliner	DOUBLE	-	Yes	Optional recliner seat price override.
price_vip	DOUBLE	-	Yes	Optional VIP seat price override.
is_active	BOOLEAN	-	Yes	Soft-delete flag showing whether the movie is visible.

The show_times table connects movies with scheduled dates, times, and halls.

Table 37: Data Dictionary - show_times table

Field Name	Data Type	Key	Null	Description
show_id	INT	PK	No	Unique identifier for each scheduled show.
movie_id	INT	FK	No	References movies.movie_id.
show_date	DATE	-	No	Date of the scheduled show.
show_time	TIME	-	No	Start time of the scheduled show.
hall	VARCHAR(50)	-	Yes	Cinema hall assigned to the show.

The bookings table stores transactional ticket records and links each booking to the relevant user and movie.

Table 38: Data Dictionary - bookings table

Field Name	Data Type	Key	Null	Description
booking_id	INT	PK	No	Unique identifier for each booking.
user_id	INT	FK	No	References users.user_id.
movie_id	INT	FK	No	References movies.movie_id.
show_time	VARCHAR(100)	-	No	Readable show details including date, time, and hall.
number_of_seats	INT	-	No	Total number of seats booked.
seat_type	TEXT	-	No	Selected seat identifiers such as A1, A2, or B4.
total_price	DOUBLE	-	No	Total booking price.
status	VARCHAR(50)	-	Yes	Booking status such as Confirmed or Cancelled.
booking_date	TIMESTAMP	-	Yes	Date and time when the booking was made.

The hall_config table stores hall layout information and seat category mappings used by the dynamic seat-selection interface.

Table 39: Data Dictionary - hall_config table

Field Name	Data Type	Key	Null	Description
hall_name	VARCHAR(100)	PK	No	Unique name of the cinema hall.
seats_per_row	INT	-	No	Number of seats available in each row.
standard_rows	VARCHAR(100)	-	Yes	Rows mapped to standard pricing.
premium_rows	VARCHAR(100)	-	Yes	Rows mapped to premium pricing.
recliner_rows	VARCHAR(100)	-	Yes	Rows mapped to recliner pricing.
vip_rows	VARCHAR(100)	-	Yes	Rows mapped to VIP pricing.
layout_map	TEXT	-	Yes	Visual seat category map used for dynamic seat rendering.

The system_settings table stores global configuration values such as default ticket prices and available halls.

Table 40: Data Dictionary - system_settings table

Field Name	Data Type	Key	Null	Description
setting_key	VARCHAR(100)	PK	No	Unique setting name.
setting_value	TEXT	-	Yes	Stored configuration value for the setting.

- **users:** Purpose: Stores customer and administrator accounts. Important fields: user_id, full_name, email, phone, password, role, reset_otp, otp_expiry Relationship: One user can have many bookings.
- **movies:** Purpose: Stores movie catalogue data and pricing overrides. Important fields: movie_id, title, genre, director, duration, language, dates, poster, trailer_url, prices, is_active Relationship: One movie can have many show times and bookings.
- **show_times:** Purpose: Stores schedule slots for movies. Important fields: show_id, movie_id, show_date, show_time, hall Relationship: Each show time belongs to one movie and one hall name.
- **bookings:** Purpose: Stores ticket purchase records. Important fields: booking_id, user_id, movie_id, show_time, number_of_seats, seat_type, total_price, status Relationship: Each booking belongs to one user and one movie.
- **hall_config:** Purpose: Stores physical hall and row category configuration. Important fields: hall_name, seats_per_row, standard_rows, premium_rows, recliner_rows, vip_rows, layout_map Relationship: Linked logically to scheduled halls and seat map generation.
- **system_settings:** Purpose: Stores global application settings. Important fields: setting_key, setting_value Relationship: Used for ticket pricing and shared configuration.

6.3 Data Structures and Algorithms Used

- **ArrayList / List:** Used to store movie lists, booking lists, show time lists, and search results returned from DAO methods. This was selected because Useful for displaying repeating data on JSP pages.
- HashMap / Map: Used to prepare dashboard values, price settings, environment variables, seat categories, and JSON-like key-value data. This was selected because it provides fast lookup by keys such as setting name, seat ID, or environment variable name.
- **Two-dimensional seat layout representation:** The hall layout is treated as rows and seats so each seat can be generated dynamically. This was selected because Supports visual seat maps and row-based pricing.
- **String parsing:** Seat identifiers such as A1, B3, and F12 are parsed to identify row, number, and pricing tier. This was selected because Required for seat category calculation and booking analytics.
- **Interval conflict checking:** Movie start time, duration, and cleaning buffer are compared with existing show intervals. This was selected because Prevents overlapping show times in the same hall.
- **SQL aggregation:** SUM, COUNT, JOIN, GROUP BY, and COALESCE are used for dashboard statistics. This was selected because Supports revenue totals, top movies, and recent activity.

6.4 Core Feature Development

- **Authentication and authorization:** The system validates login credentials, stores user details in session, and uses AuthFilter to protect routes. Admin pages are separated from normal user pages through role checks.
- **Movie catalogue management:** Administrators can add, edit, search, filter, and hide movies. Soft delete is used so previous booking records remain meaningful even when a movie is removed from public display.
- **TMDB auto-fill:** The management form supports searching an external movie source to reduce manual data entry for title, description, cast, poster, duration,

format, age rating, and trailer information. Manual fields remain available when external metadata is incomplete.

- **Dynamic booking engine:** The booking page reads hall configuration and booked-seat data, creates JSON-backed seat information for the JSP, renders the visual seat map, and updates selected seats and total price as the customer interacts with the page.
- **Show scheduling:** The schedule builder links movies to dates, times, and halls. Conflict checking protects each hall from overlapping shows and includes a cleaning buffer after a movie ends.
- **Dashboard analytics:** The admin dashboard uses database aggregation to show operational information such as revenue, bookings, users, recent activity, and most booked movies.
- **System settings:** Administrators can configure ticket prices and hall layouts without changing source code. This makes the system more flexible for different cinema setups.

6.5 Validation and Exception Handling

Validation is applied before records are saved to the database. Registration requires a valid name, email, phone number, and password. Movie forms require valid dates, duration, title, and pricing values. Booking requires an authenticated user, a valid show, and seats that have not already been booked. Environment loading and database connection errors are handled so configuration problems can be diagnosed during deployment.

Exception handling is essential because the application depends on external systems such as MySQL, TMDB, and email services. When one component fails, the user should receive a controlled and readable message, while the system avoids exposing internal stack traces, credentials, or implementation details.

7. Critical Analysis

7.1 System Breakdown and Development Challenges

- **Dynamic seat map complexity:** Impact: The seat map must combine hall layout, seat tier rules, selected seats, and booked seats. A small mismatch in row naming or JSON formatting can affect booking accuracy. Handling method: The design separates hall configuration from booking records and builds a consistent seat JSON structure before rendering.
- **Show time conflicts:** Impact: Scheduling becomes difficult when two movies use the same hall near the same time. Handling method: The system uses duration and a cleaning buffer to compare show intervals before saving a new schedule.
- **Data consistency:** Impact: Movies, users, show times, and bookings depend on one another. Handling method: Foreign keys and DAO-level checks reduce invalid records and protect historical booking data.
- **Role separation:** Impact: Normal users and administrators need different access permissions. Handling method: AuthFilter centralizes route protection so role checks are not duplicated across every JSP page.
- **External API dependency:** Impact: TMDB data may be unavailable, incomplete, or delayed. Handling method: Manual entry fields remain available so the administrator can still add movies without relying fully on the API.
- **Responsive design:** Impact: Tables, seat maps, and dashboard cards can become crowded on small screens. Handling method: CSS layout planning, flexible grids, and simplified wireframes support a more adaptable interface.

7.2 SWOT Analysis

- **Strengths:** Clear MVC structure; dynamic seat selection; role-based access; centralized MySQL database; admin dashboard; configurable pricing; soft delete for movies; password reset support.

- **Weaknesses:** The system depends on server availability and database connectivity. Payment processing is represented as a booking confirmation flow rather than a real payment gateway. Some analytics depend on the quality of stored booking data.
- **Opportunities:** The system can be expanded with real online payment, QR verification at the cinema entrance, staff roles, refund workflow, mobile-first interface, recommendation logic, and notification templates.
- **Threats:** Security risks may appear if password hashing, session timeout, input validation, or SQL parameterization are not maintained. External API changes may affect TMDB auto-fill. High traffic may require optimization and caching.

7.3 Comparison with Existing Systems

- **Manual counter booking:** Problem: Requires physical presence, manual seat checking, and staff-driven communication. MovieMint improvement: MovieMint provides self-service browsing and booking through a web interface.
- **Spreadsheet-based management:** Problem: Data is scattered and reporting is manual. MovieMint improvement: MovieMint stores operational data in a structured relational database.
- **Basic booking forms:** Problem: May collect booking details but do not show real-time seat states. MovieMint improvement: MovieMint provides a dynamic seat map with available, selected, and booked states.
- **Unprotected admin pages:** Problem: Sensitive operations may be accessible to the wrong users. MovieMint improvement: MovieMint uses session validation and role-based filtering.
- **Static pricing systems:** Problem: Changing ticket prices may require code changes or manual adjustment. MovieMint improvement: MovieMint stores global and movie-specific price settings in the database.

7.4 Performance Evaluation

The system is designed for responsive data access by using targeted SQL queries and separating database operations into DAO methods. Search and filter operations are handled through database conditions rather than loading all records into the interface. Dashboard values are produced through aggregation queries, which reduces repeated manual calculations in Java code.

- **Movie search:** Search keyword, genre, language, and status filters are applied at query level. Evaluation: Fast enough for small and medium cinema catalogues; indexing title, genre, language, and active status can improve larger deployments.
- **Seat map loading:** The page loads hall layout and already booked seats for one selected show. Evaluation: Efficient because only the relevant show data is required.
- **Booking confirmation:** The system validates seat availability before insertion. Evaluation: Accuracy is prioritized over speed to avoid duplicate booking.
- **Dashboard analytics:** Revenue and booking summaries use SQL aggregation. Evaluation: Suitable for administrative reporting; very large booking tables may require date indexes or cached summaries.
- **Route security:** AuthFilter checks sessions before controllers render protected pages. Evaluation: Centralized checking reduces repeated logic and improves reliability.

7.5 Ethical and Security Considerations

The system stores personal user information such as names, emails, phone numbers, and passwords. For that reason, passwords should be hashed before storage, session data should be limited to necessary values, and administrative access should remain restricted. The application should also avoid exposing internal errors to customers. Password reset OTP values should expire and should not be reused after successful reset.

The system also follows an ethical project topic because it supports legitimate cinema operations and customer convenience. It does not promote harmful activity and can be evaluated through standard software engineering principles such as usability, reliability, maintainability, and data protection.

8. Conclusion and Future Recommendations

8.1 Conclusion

MovieMint successfully represents a dynamic movie booking management system with both customer-facing and administrator-facing functionality. The system demonstrates how a Java web application can combine JSP views, servlet controllers, DAO classes, MySQL storage, validation, authentication, and dynamic user interface behaviour to solve a practical cinema management problem.

The main strength of the project is that it goes beyond a basic booking form. It includes a structured database, role-based access, password reset, dynamic seat mapping, show scheduling, seat pricing, soft delete, admin analytics, and configurable hall settings. These features show practical use of data structures, object-oriented design, SQL relationships, and MVC architecture.

8.2 Lessons Learned

- A clear MVC structure reduces confusion and makes the system easier to maintain.
- Database design decisions directly affect feature quality, especially bookings, schedules, and reporting.
- Dynamic seat booking requires careful coordination between server-side data and client-side rendering.
- Validation and exception handling are essential for protecting data integrity and improving user experience.
- Wireframes make development easier because page structure, navigation, and workflows are decided before implementation.

8.3 Limitations

- The current system does not include a real payment gateway integration.
- The ticket confirmation includes a QR placeholder rather than a fully verified scanner workflow.
- Email services depend on external configuration and may fail if credentials or network access are not available.
- The application is monolithic, which is suitable for coursework and small deployments but may require modularization for very large traffic.
- Advanced staff roles such as cashier, manager, and support agent are not separated in detail.

8.4 Future Recommendations

- Integrate a secure online payment gateway and store payment status separately from booking status.
- Generate real QR codes for each ticket and create a staff scanning page for entry verification.
- Add email and SMS notification templates for booking confirmation, cancellation, and schedule changes.
- Add refund and cancellation policies with time-based rules.
- Introduce audit logs for administrative actions such as price changes, movie deletion, and booking status updates.
- Add advanced seat layout editing with drag-and-drop controls.
- Create REST API endpoints for a future mobile application.
- Add database indexes for frequently searched columns such as movie title, show date, status, and user email.
- Improve reporting with export options such as CSV and PDF summaries.
- Add accessibility improvements such as keyboard-friendly seat selection, ARIA labels, and stronger colour contrast.

References

Apache Software Foundation (2023) *Apache Tomcat 10.1 Documentation*. [Online]. Available at: https://tomcat.apache.org/tomcat-10.1-doc/ (Accessed: 18 April 2024).

Chart.js Contributors (2023) *Chart.js Documentation*. [Online]. Available at: https://www.chartjs.org/docs/latest/ (Accessed: 02 May 2024).

Jakarta EE (2022) *Jakarta Mail Specification (JavaMail API)*. Eclipse Foundation. [Online]. Available at: https://jakarta.ee/specifications/mail/ (Accessed: 29 April 2024).

Jakarta EE (2022) *Jakarta Servlet Specification, Version 6.0*. Eclipse Foundation. [Online]. Available at: https://jakarta.ee/specifications/servlet/6.0/ (Accessed: 22 April 2024).

Mozilla Developer Network (2024) *MDN Web Docs: Web technology for developers*. [Online]. Available at: https://developer.mozilla.org/ (Accessed: 25 April 2024).

Oracle Corporation (2023) *Java Platform, Standard Edition (Java SE) 21*. [Online]. Available at: https://docs.oracle.com/en/java/javase/21/ (Accessed: 15 April 2024).

Oracle Corporation (2023) *MySQL 8.0 Reference Manual*. [Online]. Available at: https://dev.mysql.com/doc/refman/8.0/en/ (Accessed: 20 April 2024).

The Movie Database (TMDB) (2024) *TMDB API Documentation*. [Online]. Available at: https://developer.themoviedb.org/docs (Accessed: 05 May 2024).

Appendices

Appendix A: Project Folder Structure

Folder or File	Purpose
`src/main/java/com/moviebooking/config`	Contains configuration classes such as database connection handling and environment-based setup.
`src/main/java/com/moviebooking/controllers`	Contains Servlet controllers responsible for public routes, user routes, admin routes, booking actions, movie management, authentication, and API-related request handling.
`src/main/java/com/moviebooking/dao`	Contains DAO classes that perform database operations, SQL queries, CRUD processing, row mapping, dashboard aggregation, booking lookup, and persistence logic.
`src/main/java/com/moviebooking/filter`	Contains authentication and authorization filters used to protect user and admin routes from unauthorized access.
`src/main/java/com/moviebooking/model`	Contains Java model/entity classes such as `User`, `Movie`, `Booking`, `ShowTime`, and `HallConfig`, which represent the main data objects of the system.
`src/main/java/com/moviebooking/service`	Contains service-level classes such as email delivery and business-support logic, including Gmail SMTP ticket or OTP delivery.
`src/main/java/com/moviebooking/util`	Contains reusable utility classes for

	validation, OTP generation, password handling, environment loading, PDF generation, and helper operations.
src/main/webapp/WEB-INF/pages	Contains protected JSP pages that cannot be accessed directly through the browser and must be served through Servlets.
src/main/webapp/css	Contains shared CSS files used to style the public pages, user pages, admin pages, forms, cards, tables, and responsive layouts.
src/main/webapp/js	Contains JavaScript files used for dynamic behaviour such as seat map rendering, form enhancement, filtering, and interactive interface actions.
src/main/webapp/images	Stores image assets such as movie posters, downloaded TMDB posters, icons, and other visual resources used in the application.
src/main/webapp/index.jsp	Entry page of the web application, usually used to redirect users to the correct public route or landing page.
DATABASE_SCHEMA.sql	Database setup script used to create the MovieMint MySQL database, including users, movies, bookings, show times, hall configuration, and system settings tables.
.env.example	Example environment configuration file showing required variables such as database credentials, TMDB API

	key, and Gmail SMTP settings.
README.md	Project documentation file containing setup instructions, project overview, technologies used, and repository information.

Appendix B: Route Map

Route	Page or Behaviour	Access
/login	Displays login page and handles user/admin authentication.	Public
/register	Displays registration page and handles new user account creation.	Public
/forgotPassword	Handles password reset process, including email entry, OTP verification, and new password submission.	Public
/about	Displays information about MovieMint and the purpose of the system.	User
/contact	Displays contact page and handles contact/support form interaction.	User
/userHome	Displays the user dashboard with recently added or available movies.	User
/browseMovies	Displays available movies with search and filtering options by genre, language, and show availability.	User
/movieDetails	Displays detailed movie information, including description, cast, trailer, format, language, and booking option.	User
/bookTicket	Handles ticket booking workflow, including showtime selection, dynamic seat map loading, seat selection, and price calculation.	User
/ticketConfirmation	Displays booking confirmation details after a	User

	successful ticket booking.	
/myBookings	Displays the logged-in user's booking history and ticket records.	User
/userProfile	Allows users to view and update their profile information.	User
/adminHome	Displays the admin dashboard with revenue, bookings, users, movies, charts, and recent activity.	Admin
/manageMovies	Allows admin to add, update, search, hide, and manage movies, including show schedules and TMDB auto-fill.	Admin
/manageUsers	Allows admin to view, search, manage, and update user information or roles.	Admin
/manageBookings	Allows admin to view booking records, update booking status, and manage ticket reservations.	Admin
/manageSettings	Allows admin to configure system settings such as ticket prices, hall layouts, seat categories, and global cinema settings.	Admin
/tmdbSearch	Handles TMDB API search requests and returns movie metadata for the admin movie form.	Admin
/logout	Ends the current user/admin session and redirects to the login page.	User/Admin
/error404	Displays a not-found page when an invalid route is accessed.	Public
/error500	Displays a controlled server error page when an internal error occurs.	Public