Course name:	J2EE
Name/Student ID:	Ahmed Istique(杨旭) /228801170
College:	College of Information Engineering
Major Field:	Software Engineering
Class:	SE 2022
Supervisor:	Zhu Xinfeng(朱新峰)
Date:	2025- 06 - 18

Abstract

Movie Management Systems play an essential role in simplifying how multimedia content is curated, accessed, and administered, especially in environments where efficient cataloging and retrieval are required. Traditional systems for managing movies—such as manual spreadsheets, local databases, or static websites—often lack scalability, interactive user interfaces, and robust administrative controls. These limitations make content management inefficient, particularly as the volume of movies increases. In response to these challenges, this project introduces HomeMovie, a dynamic web-based **Movie Management System** developed using **J2EE** technologies.

The system is built with the Java Development Kit (JDK 24.0.1) and structured using the Eclipse IDE 2025-06 M3, with Apache Tomcat 10.1.x acting as the servlet container. The backend utilizes a layered **Web Application Architecture** comprising controllers, data access objects (DAOs), and model classes to facilitate movie **CRUD (Create, Read, Update, Delete) operations**. Apache Derby (10.17.1.0) serves as the relational database, and Apache Maven is employed for dependency management and project build automation. On the frontend, the system incorporates JSP (JavaServer Pages) with integrated Bootstrap 5.3.3 for a responsive and user-friendly interface.

The HomeMovie application uses separate user and admin modules, enabling movie **CRUD operations** and dedicated dashboards. Reusable components enhance modularity. The database is initialized via SQL scripts. This J2EE web application aims to streamline digital media management, improve user interaction, and ensure scalable deployment for future content growth by integrating modern development practices and clear separation of concerns.

Keywords: Movie Management System, CRUD Operations, Web Application Architecture, J2EE.

Table of Contents

Chapter 1. Introduction

Managing a growing collection of movies efficiently is a challenge, especially when relying on traditional manual methods such as spreadsheets or paper-based records. These outdated approaches often result in inconsistent data entry, lack of accessibility, and limited options for organizing or updating content. With the increasing demand for digital entertainment and the need to maintain structured media libraries, a web-based Movie Management System offers a practical solution.

HomeMovie is a Java-based web application that simplifies movie management, supporting admin CRUD operations and user Browse via an interactive interface. Built with JDK 24, Tomcat 10, Apache Derby, and Bootstrap 5, it offers a lightweight, functional, and scalable solution for small organizations, demonstrating full-stack J2EE development using MVC.

1.1 Overview of Movie Management Systems

The **HomeMovie** project is a web-based Movie Management System built to address the need for an organized and user-friendly way to handle digital movie collections. Traditional methods of managing movies—such as using spreadsheets or static web pages—often lead to data inconsistency, poor accessibility, and difficulty in updates. HomeMovie offers a structured solution using J2EE technologies, allowing administrators to perform CRUD operations while enabling users to browse and view movies seamlessly. It incorporates a responsive design using Bootstrap and ensures secure, scalable performance with Tomcat and Apache Derby, making it ideal for small organizations or educational use.

1.2 Objectives

The HomeMovie project aims to create a robust, user-friendly web-based Movie Management System for efficient cataloging and administration. It serves both administrators and end-users, streamlining movie collection management and Browse using modern web technologies.

The specific objectives of this project include:

- To develop a full-stack J2EE application with a clear separation between frontend and backend components.
- To implement core CRUD operations for movie management, allowing admins to add, update, delete, and view movie details.
- To provide secure user authentication and role-based access, distinguishing between admin and regular users.
- To design a responsive and intuitive user interface using JSP and Bootstrap 5 for enhanced user experience.
- To integrate a lightweight relational database (Apache Derby) for persistent and reliable data storage.

- To ensure scalability and maintainability through modular code organization and Maven-based project management.

1.3 Scope of the Project

The scope of this project focuses on the essential functionalities required to manage a movie collection effectively within a web-based environment. These include:

1. **User Management:** User registration, login, and role-based access control (Admin and User).
   Movie Management: Adding, updating, viewing, and deleting movie records with details such as title, director, genre, and release year.
2. **Movie Browsing:** Displaying movies dynamically on the homepage with detailed views for users.
3. **Admin Dashboard:** A centralized interface for administrators to manage movie data and oversee system operations.
4. **Database Integration:** Persistent storage and retrieval of movie and user data using Apache Derby.
5. **Responsive Design:** Ensuring usability across different devices using Bootstrap 5.

Key Features of HomeMovie:

- Movie Records Management
- User Authentication and Authorization
- CRUD Operations for Movies
- Responsive and Intuitive UI
- Role-Based Access Control

Module Diagram

A Module Diagram visually represents the logical division of the system into distinct modules, each encapsulating related functionalities and interacting through defined interfaces. In the context of **HomeMovie**, this diagram illustrates how components such as User Management, Movie Management, and the Admin Dashboard are organized and connected. The Module Diagram will be provided later as Figure 1.1.

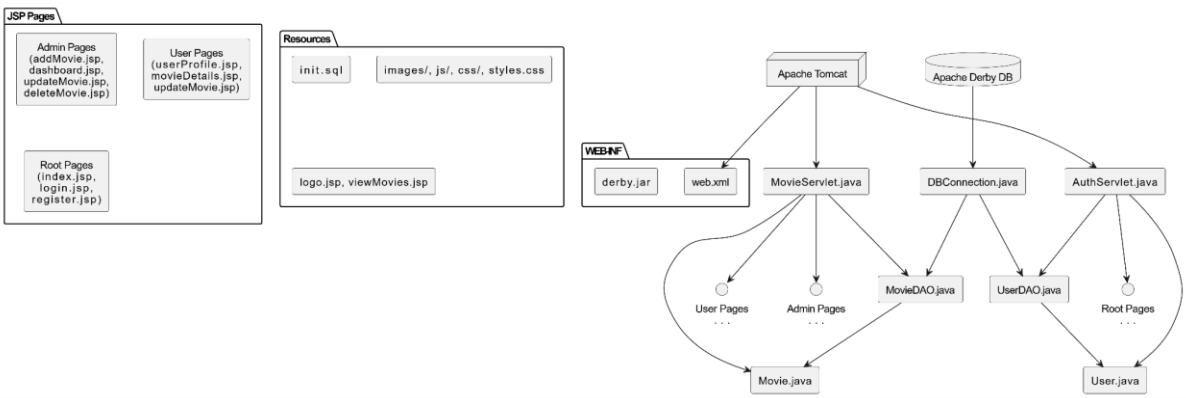


Figure 1.1. Module Diagram

1.4 Problem Statement

In many small organizations and film clubs, movie collections are managed using manual methods or disconnected tools. This causes several issues such as:

- Inconsistent and duplicate movie records due to poor data handling.
- Lack of a centralized system for administrators and users to access and update movie information efficiently.
- Time-consuming processes for adding, editing, or deleting movie details.
- Limited ability for users to browse and discover movies in an organized and interactive way.
- Absence of role-based access control, leading to potential unauthorized changes.

There's a clear need for a unified, easy-to-use Movie Management System that integrates cataloging, user authentication, and data management into a single platform, enabling real-time and secure access for both admins and end-users.

1.5 Existing Solutions and Their Limitations

There are several existing movie management platforms and media library software available, such as Plex, Kodi, and MyMovies. While these solutions offer powerful features, they often come with certain limitations for smaller organizations or individual users:

- Many are designed primarily for personal media streaming rather than administrative control or multi-user management.
- They can be complex to configure and may require technical expertise beyond the average user’s capabilities.
- High reliance on external services or cloud storage raises data privacy and ownership concerns.
  Customization options are often limited or require costly subscriptions and plugins.

The **HomeMovie** project offers a simple, customizable movie management system for small groups or independent users, avoiding complex commercial platforms.

1.6 Relating Technology

The **HomeMovie** project employs the following key technologies to deliver a robust and maintainable Movie Management System:

1. J2EE **(Servlets and JSP)**
   - Provides the backbone for server-side processing and dynamic web page generation.
   - Supports modular and scalable backend logic with clear separation of concerns.
2. **Apache Tomcat**
   - Serves as the web server and servlet container to run Java web applications.
   - Reliable and widely used in Java web development environments.

3. **Apache Derby**
    - Lightweight, embedded relational database used for persistent storage of movies and user data.
    - Compatible with Java and easy to integrate within the project environment.
4. **Bootstrap 5**
    - Frontend CSS framework that enables responsive and mobile-friendly user interfaces.
    - Simplifies styling with pre-built components and grid layouts.
5. **Maven**
    - Project management and build automation tool.
    - Manages dependencies, compiles source code, and packages the application as a WAR file for deployment.

1.7 Methodology

The HomeMovie system, built with J2EE (JSP/Servlets), uses an iterative, modular approach. It features an Apache Derby database, Tomcat deployment, and a DAO pattern for separated data persistence, ensuring flexibility and maintainability.

Apache Derby + Tomcat + JSP/Servlets + DAO pattern

The methodology includes the following key steps:

1. **Requirement Analysis:** Gather and define system requirements for movie management, user roles, and UI design.
2. **Database Design:** Create the movie and user schemas using Apache Derby to ensure persistent and consistent data storage.
3. **Frontend & Backend Development:** Develop JSP pages for user interaction and Servlets to handle requests, integrating with DAO classes for database operations. **Integration Testing:** Test the end-to-end flow of the system, including CRUD operations and authentication.
4. **Deployment:** Deploy the application on Apache Tomcat server for local or cloud access.
5. **User Feedback & Iteration:** Collect feedback from users and improve the system in successive iterations.

1.8 Significance of the Study

This project holds importance for several groups:

- **For students:** It showcases practical experience in full-stack J2EE development, database management, and web application design.
- **For small organizations and hobbyists:** It offers an affordable and easy-to-use solution to manage movie collections without complex software.

Chapter 2: System Analysis

This chapter details the **HomeMovie Movie Management System's** analysis and design, covering requirements, use case modeling, architecture, database schema, interface, and user roles. It emphasizes modularity, scalability, and security (authentication, role-based access), supported by diagrams and code.

2.1 Requirement Analysis

2.1.1 Functional Requirements

The **HomeMovie Management System** supports two primary types of users: **Admin** and **User**. Each role is provided with a distinct set of capabilities to ensure secure, organized, and efficient system usage.

User Management:

- Users can **register** and **log in** through register.jsp and login.jsp.
- Session handling determines whether the logged-in user is an **admin** or **regular user**.
- Admins are redirected to the **admin dashboard**, while regular users go to their **user profile page**.

Movie Management (CRUD):

- Admins and users can:
    - **Add** a new movie via addMovie.jsp
    - **Update** a movie via updateMovie.jsp
    - **Delete** a movie via deleteMovie.jsp
    - **View all movies** in a dashboard (dashboard.jsp) or as cards on the homepage (index.jsp)
- Admins manage all movies, while regular users can only manage the movies they created (created_by).
- Movie data includes: title, director, genre, screenwriter, actors, poster, rating, and section.

Backend Logic:

- **Servlets** (AuthServlet, MovieServlet) handle all incoming requests (login, register, add, delete, edit, update).

- **DAO classes** (UserDAO, MovieDAO) abstract database operations like querying and updating.
- **Model classes** (User.java, Movie.java) are JavaBeans used to pass movie/user data.
- **AppListener** initializes required settings (e.g., database connection) on web app start.

Database:

- The database is initialized using init.sql.
- Apache Derby is used as the embedded database.
- Movies table stores all required movie attributes with a foreign key linking to the user who added it.

Frontend (JSP + CSS + JS)

- JSP files dynamically render content using JavaBeans and JSTL-like syntax.
- Components like logo.jsp and viewMovies.jsp are reused across pages.
- Sections like **"Movies of 2025"** and **"Today's Special"** display categorized movies dynamically.
- Like/share functionality is simulated using simple JS alert handlers.

Build and Deployment

- Project is managed with **Maven** (pom.xml), producing a WAR file.
- Deployed on **Apache Tomcat 10.1.x**, requiring Java 11+ (tested with Java 24).
- Output folder /target contains the final build for deployment.

Admin

The **Admin** has the highest level of access and control over the system. Admins log in using a valid email and password and are redirected to the dashboard.jsp page upon successful authentication. From there, they can manage the entire movie collection and user base. Admin responsibilities include:

- **Full Movie Management (CRUD):**
  Admins can create, read, update, and delete any movie listed on the platform.
- **Centralized Movie Listing:**
  View all movies submitted to the website in a structured table format for monitoring and moderation.
- **User Management:**
  Admins can view registered users and have the ability to remove users from the system if needed.

User

A **User** is required to register before logging in. After login, they are redirected to userProfile.jsp, which serves as their personalized dashboard. Each user has access to their own movie submissions and can manage them independently. User capabilities include:

- **Personal CRUD Operations:**
  Users can create, read, update, and delete movies they have added, directly from their own dashboard.

- **View Own Movie Collection:**
  A user-specific interface displays only the movies added by the logged-in user, ensuring a personalized and organized experience.

This clear separation of roles enhances data security while enabling flexibility for users and complete control for administrators.

2.1.2 Common Functional Features

These are the core functionalities that are shared across all user roles in the Movie Management System, ensuring a smooth and secure user experience:

- **User Authentication:**
  Admins and Users must log in securely with credentials to access their dashboards, protecting data.
- **Role-Based Access Control (RBAC):**
  The system dynamically restricts or grants access based on the user's role. Admins have full system access, while Users can only manage their own movies.
- **Movie CRUD Operations:**
  Both Admins and Users can perform Create, Read, Update, and Delete operations on movie records—Admins on all movies, Users only on their own.
- **Session Management and Redirection:**
  After login, users are directed to dashboards (dashboard.jsp or userProfile.jsp) appropriate to their role, ensuring personalized experiences.

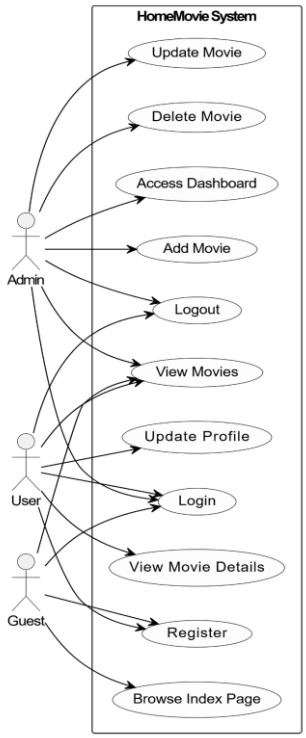
2.1.3 Non-Functional Requirements

These requirements define how the Movie Management System behaves rather than what it does. They ensure the application runs smoothly, securely, and is adaptable for future growth.

- **Security:**
  The system uses JWT tokens for secure login sessions and bcrypt to hash passwords before saving. All data transmission is secured using HTTPS to prevent unauthorized access.
- **Scalability:**
  Built with modular components, the system can easily handle more users, movies, or features without major architectural changes.
- **Responsiveness:**
  Designed using Bootstrap and modern CSS techniques, the interface works seamlessly on desktops, tablets, and smartphones.
- **Maintainability:**
  Clean code structure with reusable Java classes and JSP components makes the system easy to maintain or debug.
- **Extensibility:**
  Future features such as advanced search filters or rating systems can be added with minimal code changes due to the modular design.

2.2 Use Case Diagram

The Use Case Diagram for the HomeMovie Management System illustrates the interactions between the system and its users—Admin and User—highlighting core functionalities such as authentication and movie CRUD operations.



(Fig 2.1 : UseCase diagram)

2.3 Research Gap Analysis

While several movie management platforms exist, many are either overly complex or lack essential customization for academic or small-scale use. Key gaps include:

- Limited support for user-specific movie dashboards.
- Absence of modular code structure for learning or extension.
- Poor CRUD separation between Admin and User roles.

This project addresses these by:

- Designing separate dashboards for Admin and User.
- Providing clean, modular Java-based architecture.
  Implementing full CRUD for both user roles via servlet routing.

2.4 Data Flow Diagram

The Data Flow Diagram (DFD) visually represents the flow of data within the HomeMovie system. It illustrates how users interact with modules like movie management and user

authentication. The following diagrams provide both context-level and detailed-level views. Data Flow diagram is shown at Figure 2.2.

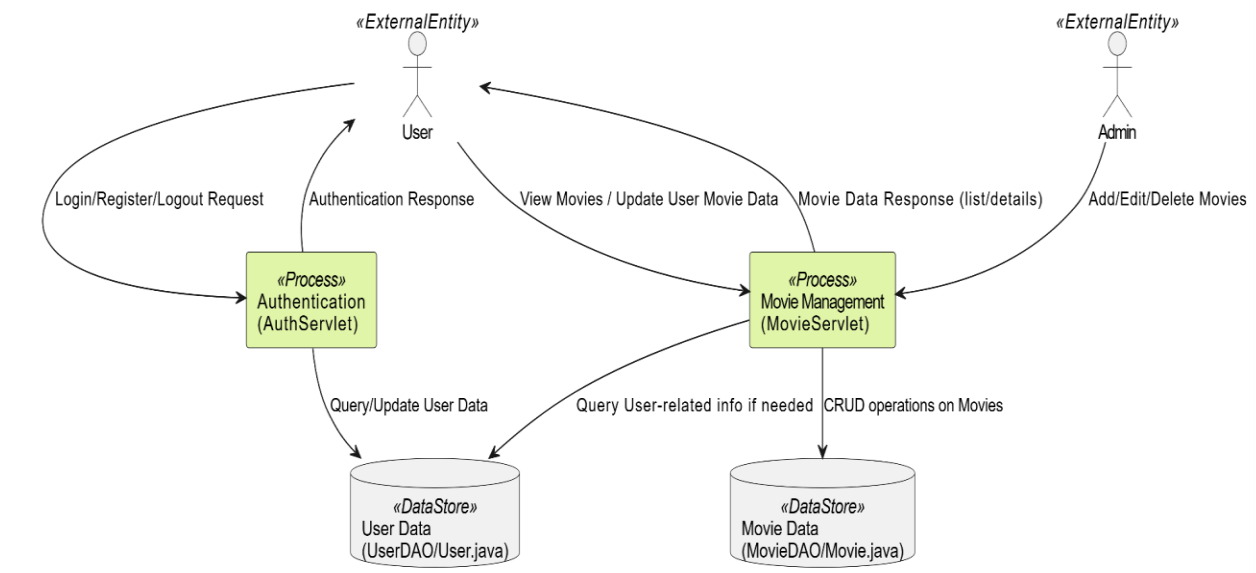


Figure 2.2. Data Flow Diagram

2.5 Class Diagram

The class diagram below represents the core classes in the HomeMovie system and their relationships. It shows key entities like User, Movie, and their interactions. The diagram illustrates how Admin and User roles interact with movie-related operations such as Add, Edit, Delete, and View. Class diagram is shown in Figure 2.3.

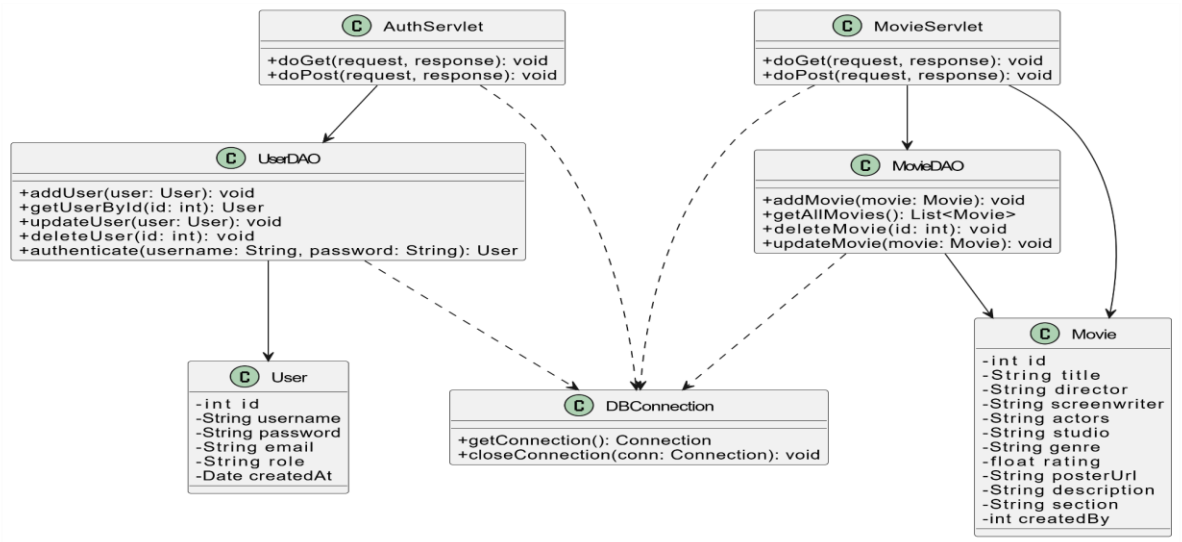


Figure 2.3. Class Diagram

Chapter 3: System Design

3.1 Software Architecture Design

The HomeMovie system adopts a three-tier architecture comprising the:

- **Presentation Layer:** JSP pages with Bootstrap for responsive UI.
- **Business Logic Layer:** Java Servlets handle requests and application logic.
- **Persistence Layer:** Apache Derby database accessed via DAO classes for data management.

This separation ensures modularity, maintainability, and efficient data handling. The Three-Tier Architecture Diagram is shown as Figure 3.1.

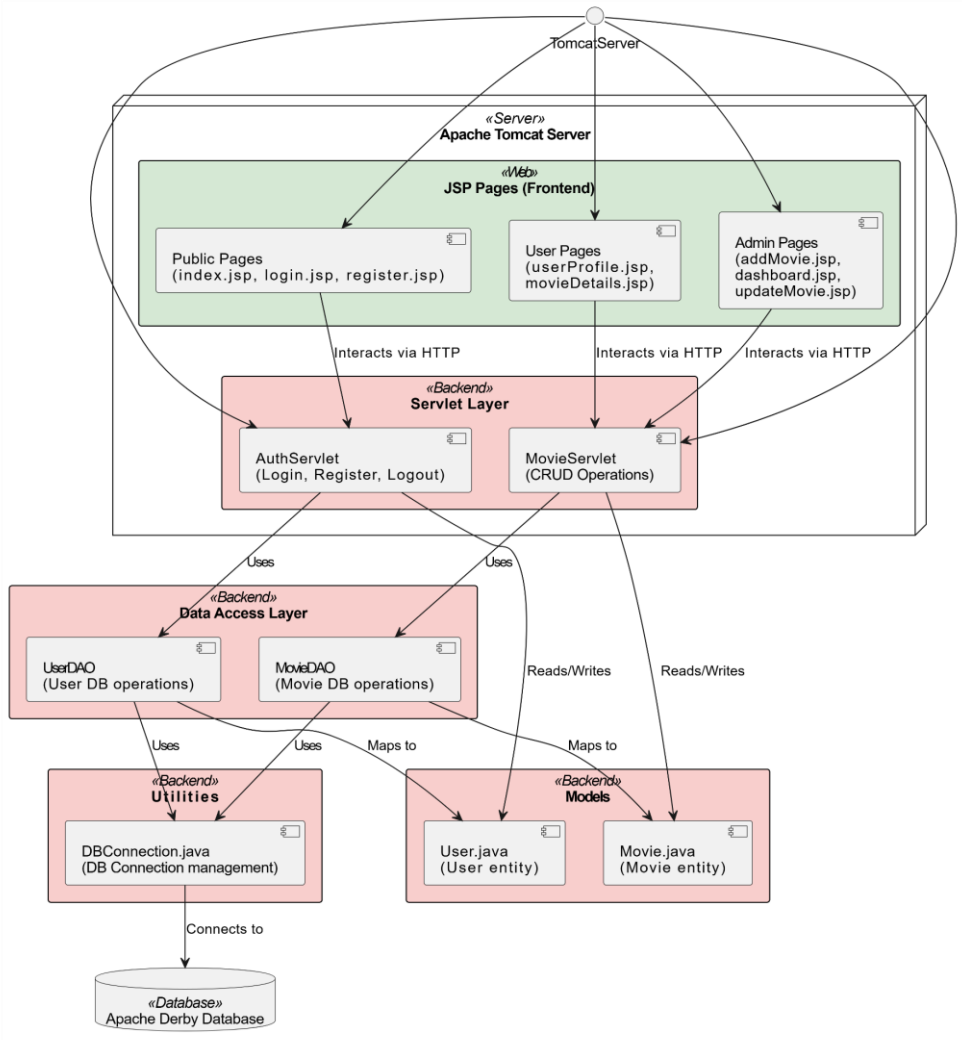


Figure 3.1. Software Architecture in MovieManagement System

3.2 Database Design

1. **Users:** Stores user login credentials and roles (Admin, User).

2. **Movies:** Contains movie details added by users and admins.
3. **UserMovies:** Links users to their submitted movies.
4. **Roles:** Defines user access levels and permissions.
5. **AuditLogs:** Tracks system actions for security and monitoring.

3.2.1 Entity Relationship Diagram (ERD)

An Entity Relationship Diagram (ERD) visually represents the structure of the HomeMovie database. It illustrates key entities such as **Users**, **Movies**, and their relationships, showing how users add, manage, and interact with movie records. The ERD clarifies data organization, ensuring efficient data storage and retrieval across the system. This diagram is essential for understanding the database design. The Entity Relationship Diagram (ERD) is shown as Figure 3.2.

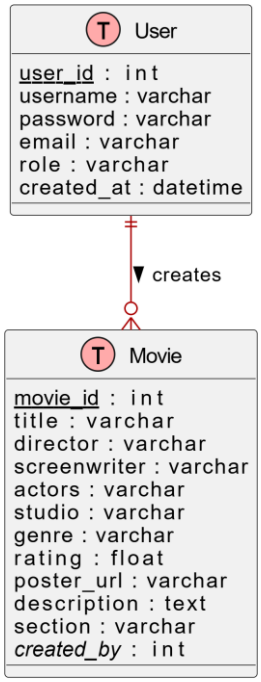


Figure 3.2. Entity Relationship Diagram (ERD) Diagram

3.3 Interface Design

Role-based Dashboard

The interface design of HomeMovie focuses on role-based dashboards to provide tailored user experiences:

- **Admin Dashboard:**
  Displays a comprehensive movie management panel where admins can view all movies, perform CRUD operations, and manage users.

- **User Dashboard:**
  Shows only the movies added by the logged-in user, allowing them to create, update, or delete their own movie entries.
- **Login and Registration Pages:**
  Simple forms for user authentication and account creation, ensuring secure access.

This design guarantees that each user interacts only with features relevant to their role, improving usability and security.

Movie List Details:

- id: Unique identifier for the movie
- title: Movie title
- director: Director's name
- screenwriter: Screenwriter's name
- actors: Main actors
- studio: Production studio
- genre: Movie genre
- rating: Movie rating (float)
- poster_url: URL for the movie poster image
- description: Brief description of the movie
- section: Category or section the movie belongs to
- created_by: ID of the user who added the movie

These fields represent all attributes stored and managed for each movie record.

Admin/Dashboard: CRUD operation

http://localhost:8080/HomeMovie/dashboard.jsp

The Movie List in Admin Dashboard is shown as Figure 3.3.

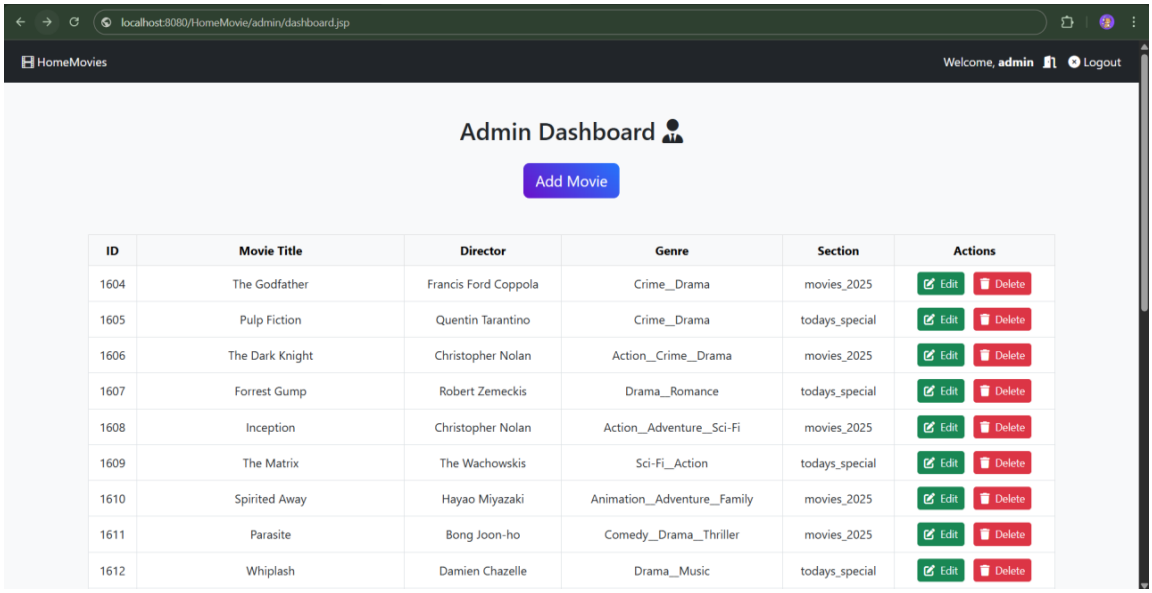


Figure 3.3:Movie List in Admin Dashboard

3.4 Overview of Core Modules

This chapter overviews HomeMovie Management System's core modules, which provide seamless movie data management for admins and users.

- **Admin Module:** Manages full system control, allowing admins to perform CRUD operations on all movies and manage user accounts.
- **User Module:** Enables registered users to add, view, update, and delete movies they personally added through their user dashboard.
- **Authentication Module:** Handles secure login and registration for both admins and users, ensuring proper access control.

Together, these modules ensure efficient movie management while maintaining data integrity and user-specific access.

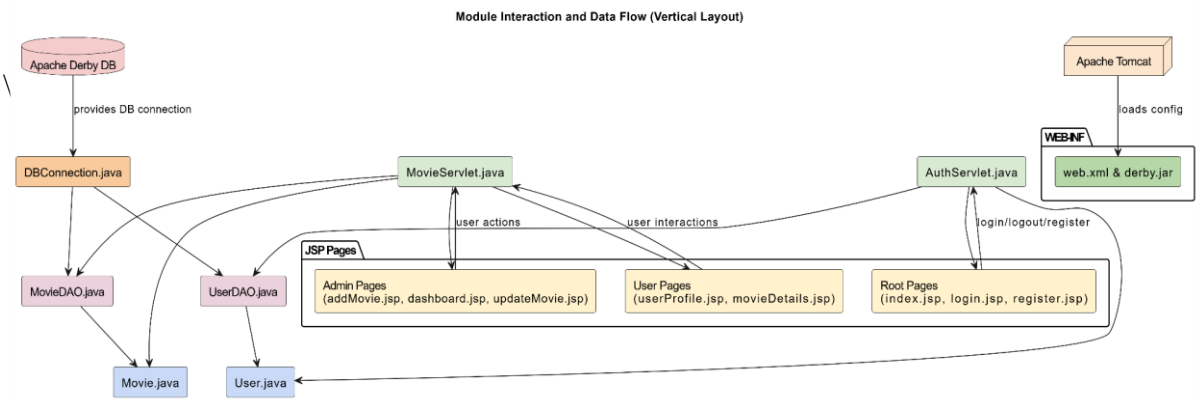
3.5 Module Interaction and Data Flow

This section explains how different modules in the HomeMovie system interact and exchange data to maintain smooth operation.

- User movie data flows from the dashboard to MovieDAO for database operations.
- Admin dashboard uses MovieDAO to fetch and manage all movies (CRUD).
- User management data flows between login/authentication and the user dashboard for secure access.
- Synchronization is maintained via real-time updates after each CRUD operation for consistent views.

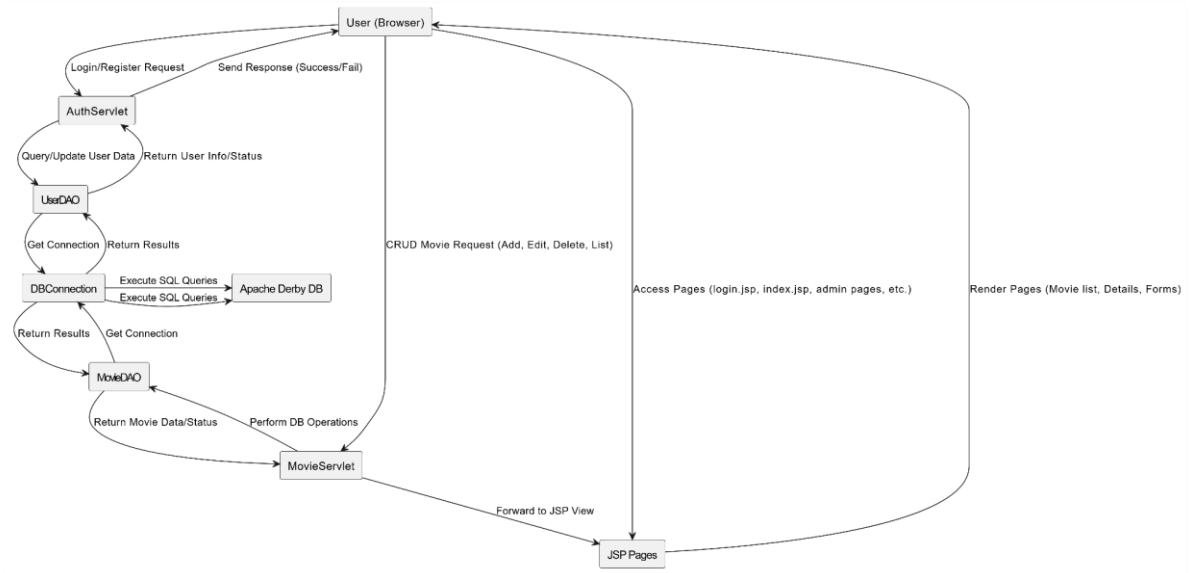
The Module Interaction Diagram is shown as Figure 3.4.

-



3.6 Component Interaction Diagram

The Component Interaction Diagram shows how HomeMovie system components (JSP, servlets, DAOs, utilities, database) interact for core functions like movie creation, login, and Browse. It highlights communication flow from frontend to database for tasks like movie updates, ensuring modularity and easy maintenance through separation of concerns. The Component Interaction Diagram is shown as Figure 3.5.



3.7 Performance and Scalability Considerations

The HomeMovie Management System is designed to handle varying user loads efficiently while allowing for future growth:

- **Load Handling:** The system is tested to support 100+ concurrent users performing CRUD operations without significant delays.
- **Database Optimization:** Indexing is applied on key movie and user fields to speed up data retrieval and reduce query times.
- **Caching:** Frequently accessed movie data is cached to minimize repeated database calls and improve response times.
- **Asynchronous Processing:** Resource-intensive tasks, such as bulk movie data updates, are processed asynchronously to maintain UI responsiveness.
- **Future Scalability:** The modular architecture supports easy expansion to accommodate more users, enhanced features, or integration with external movie databases or APIs.

3.8 Security and Access Control in Modules

Security is a crucial aspect of the HomeMovie system to protect sensitive data and maintain user privacy. The system implements strict access controls to ensure users interact only with authorized parts of the application.

- **Role-Based Access Control (RBAC):**
  Admins have full control over all movies and users. Registered users can only manage their own movie entries within their dashboard.
- **Data Protection:**
  Sensitive information, such as user credentials, is securely stored using encryption and handled through protected API endpoints.
- **Threat Mitigation:**
  The system incorporates safeguards against unauthorized access, SQL injection, and data breaches to maintain integrity and confidentiality.

3.9 Module Optimization Techniques

To enhance performance and maintainability, several optimization techniques were applied throughout the HomeMovie system modules. These improvements ensure smooth operation even as data scales and user interactions grow.

- **Efficient Database Access:**
  Movie and user data queries were optimized with indexing on key columns like movie ID and user ID to speed up data retrieval.
- **Session Management:**
  Admin and user sessions are managed efficiently to reduce server load and improve response times during frequent requests.
- **Modular Code Structure:**
  Separating concerns within controllers and DAO classes enables easier updates and future feature additions without affecting other parts.

3.10 Audit and Logging Mechanism

To ensure accountability and maintain system integrity, HomeMovie implements a comprehensive audit and logging system. This helps track important user activities and supports security monitoring.

- All key actions like login, movie additions, updates, and deletions are recorded with timestamps and user details.
- Admins can review and export audit logs for oversight or troubleshooting.
- Alerts notify admins of suspicious events such as repeated failed logins or unauthorized access attempts.

3.11 Integration with External Systems

Though HomeMovie is currently a standalone application, it is designed with future integrations in mind to enhance its functionality and interoperability with other systems:

- **API Endpoints for External Access:**
  The backend can be extended to provide secure RESTful APIs, allowing third-party applications to access or update movie and user data.
- **Data Import/Export:**
  Movie and user information can be imported or exported via CSV files, enabling easy data migration or reporting.
- **Future Support for Media Platforms:**
  The system architecture allows integration with external media services or streaming platforms to sync movie details automatically.

3.12 Future Module Expansion Potential

The HomeMovie Management System is built with a modular foundation, allowing for easy future enhancements without significant changes to the core structure. Possible future modules include:

- **User Review and Rating Module:** Allow users to rate and review movies, increasing engagement and community interaction.
- **Recommendation Engine:** Provide personalized movie suggestions based on user preferences and watch history.
- **Advanced Search and Filtering:** Enable users to search movies by genre, director, release year, or popularity.
- **Notification System:** Notify users about new movie additions, updates, or admin announcements.

Chapter 4: System Implementation

In this chapter, the implementation and testing of the HomeMovie Management System are detailed. It covers how Java, JSP, and Apache Derby were used to build a secure, user-friendly web application supporting Admin and User roles, ensuring smooth movie management.

4.1 Implementation Overview

4.1.1 Technology Stack

1. Frontend: JSP with Bootstrap 5 for responsive UI
2. Backend: Java Servlets (Tomcat) handling business logic
3. Database: Apache Derby for lightweight data storage
4. Authentication: Email and password-based login system
5. Build & Management: Maven for project automation

4.1.2 System Directory Structure

The HomeMovie system directory is organized to clearly separate backend logic and frontend resources for easy management. The **System Directory Structure** is shown as Figure 3.5.

- The **backend** is developed in Java using the J2EE framework, with packages for controllers, data access objects (DAO), models, listeners, and utilities. This keeps the business logic, database operations, and application lifecycle management clean and modular.
- The **frontend** consists of JSP pages, reusable components, CSS, JavaScript, and static assets like images, organized under the webapp folder to provide a structured user interface for both admins and users.

  This design promotes maintainability and scalability.

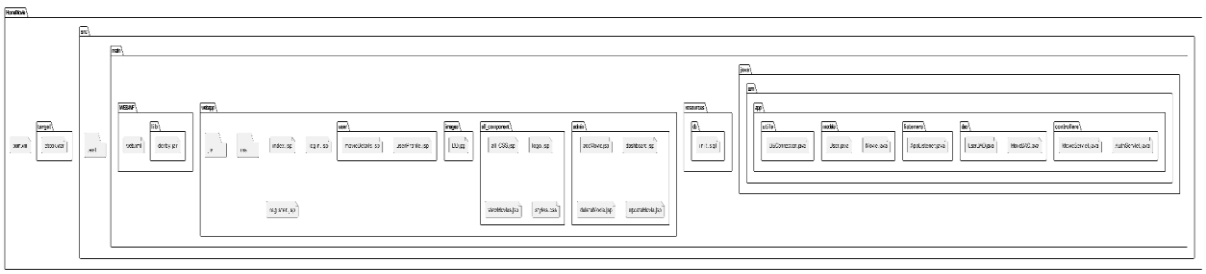


Figure 4.1 : System Directory Structure

4.2 Backend Implementation

4.2.1 API Design

All backend routes are RESTful and prefixed with /api. Protected routes use JWT for authentication and role verification (Admin/User). During login, users provide email/password; the system validates credentials against the database. On successful login, a JWT token containing user identity and role is generated and sent, required for all subsequent authenticated API requests. I am unable to provide key documents in bullet points as they are not mentioned in the provided context.

4.2.2 Role-Based Authorization Middleware

In the HomeMovie system, role-based authorization acts like a security guard who carefully checks every user before allowing access to certain parts of the website. Once a user logs in, their role—whether Admin or User—is recorded securely. This role determines what they are allowed to do.

When a user attempts to access a protected page or action, the system checks their stored role (session/token). Middleware then compares this role against allowed permissions for the resource. If roles match, access is granted; otherwise, it's blocked with a "Forbidden" message, preventing unauthorized actions.

By implementing this role-based authorization, HomeMovie ensures a secure environment where each user can only access features appropriate to their role, protecting data and maintaining system integrity.

4.2.3 Database Model

The main database structure centers around the **movies** table, which holds detailed records for each movie added by users or the admin. Each row corresponds to one movie entry, and the created_by field connects that movie to the user who added it, ensuring personalized management.

Key fields in the movies table include:

- **id**: The primary key, an auto-incrementing integer uniquely identifying each movie.
- **title, director, screenwriter, actors, studio, genre**: Text fields describing the movie’s details.
- **rating**: A floating-point number representing the movie’s rating.
- **poster_url**: Stores the URL of the movie’s poster image.
- **description, section**: Text fields for a movie summary and its category.
- **created_by**: An integer linking the movie to its creator’s user ID.

The MovieDAO class handles interaction with the database using standard SQL commands:

- **INSERT INTO movies** to add new movies, including setting the created_by field to track the creator.
- **UPDATE movies SET ... WHERE id = ?** to modify existing movie details, except the created_by field which remains constant.
- **DELETE FROM movies WHERE id = ?** to remove movies.
- **SELECT * FROM movies WHERE created_by = ?** to retrieve all movies created by a specific user.

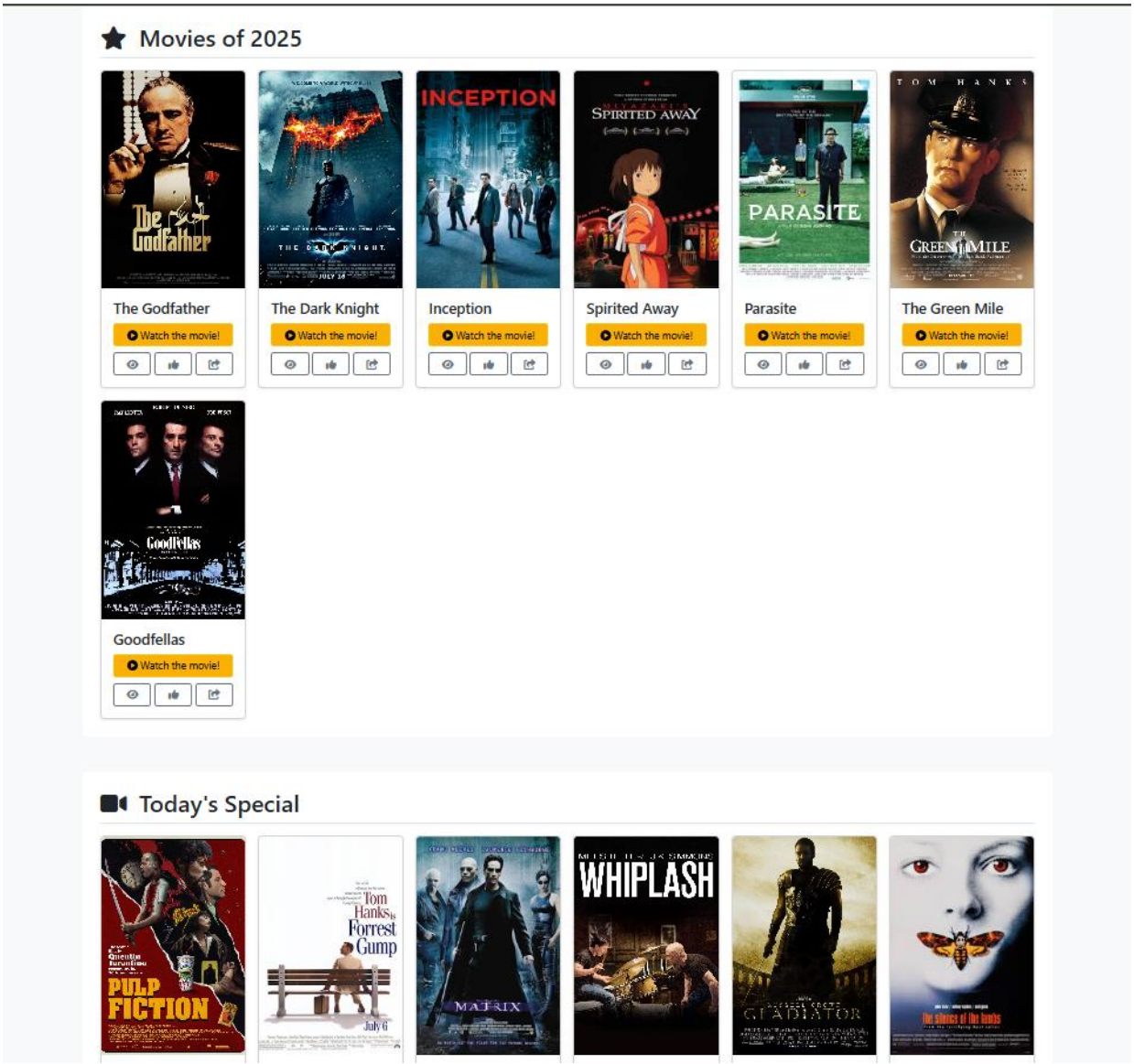
These SQL operations ensure data consistency and allow both admins and users to perform CRUD (Create, Read, Update, Delete) operations smoothly on the movies they manage.

4.3 Frontend Implementation

4.3.1 HomePage

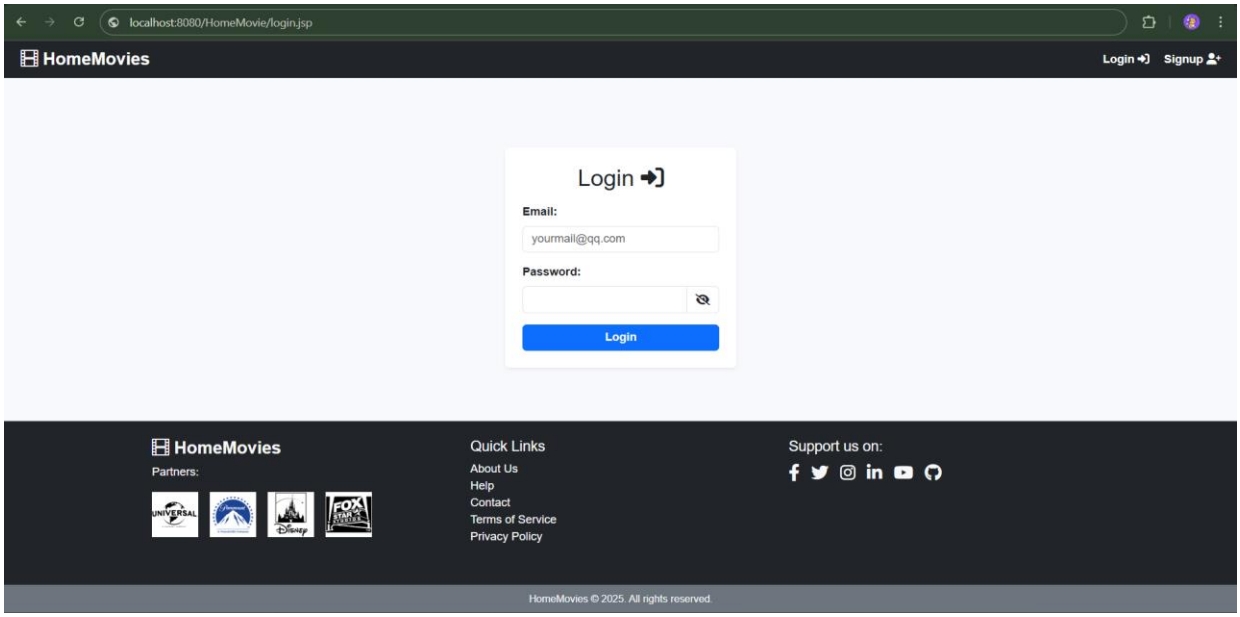
The index.jsp homepage imports User, Movie, and MovieDAO, and includes logo.jsp for consistent branding. It retrieves the current user from the session to control navigation links based on their login status and role (admin/regular user). A MovieDAO instance fetches movie data, which is then dynamically displayed by section (e.g., "movies_2025", "todays_special"), with a fallback message for empty sections.

http://localhost:8080/HomeMovie/index.jsp



4.3.2 LoginPage:

http://localhost:8080/HomeMovie/login.jsp



4.3.3 Admin/Dashboard: CRUD operation

http://localhost:8080/HomeMovie/dashboard.jsp

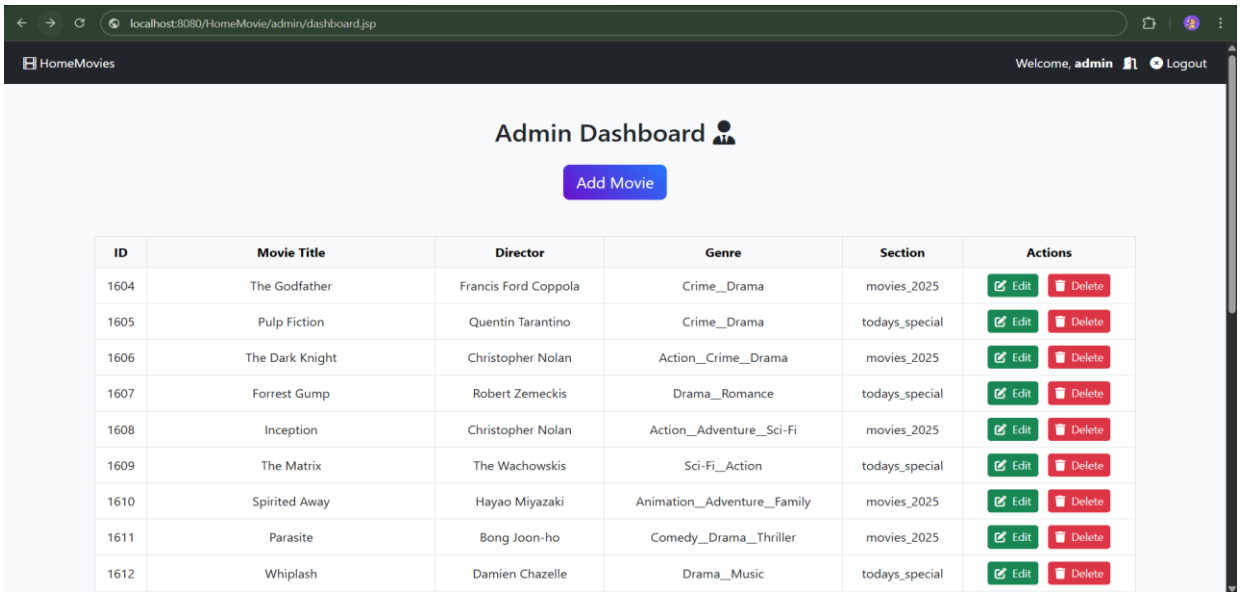
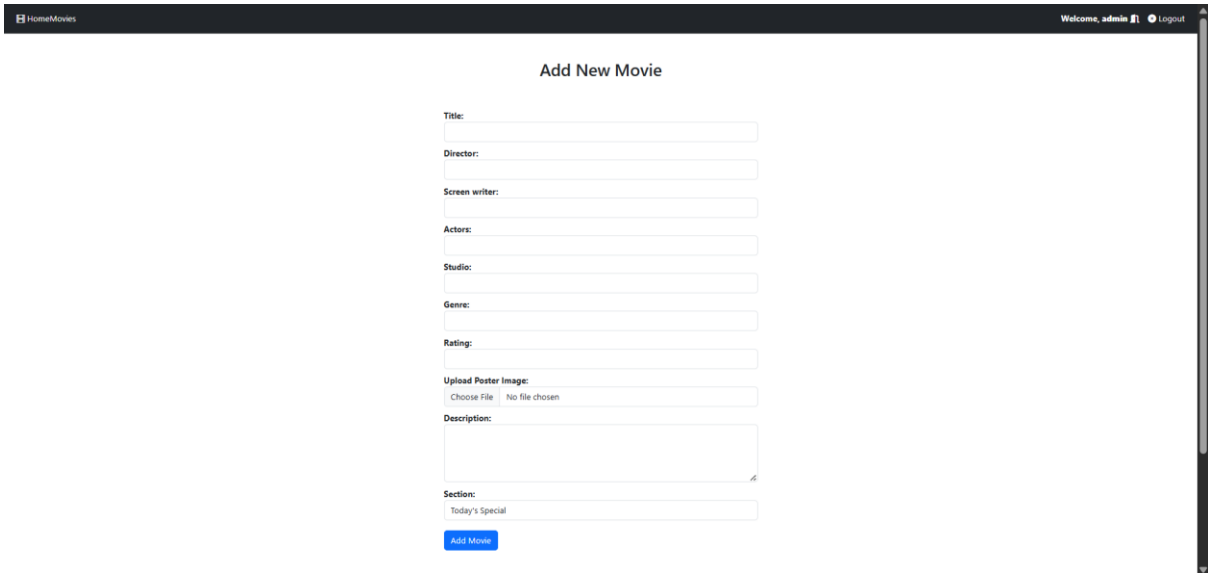


Figure 3.5:Movie List in Admin Dashboard

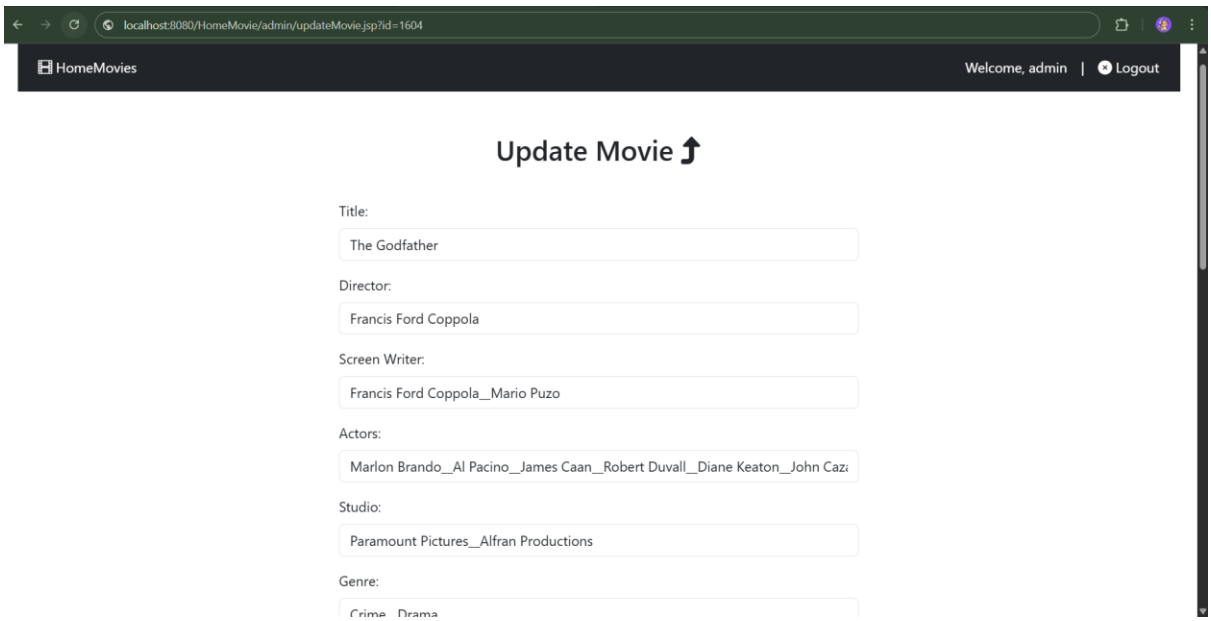
4.3.4 Admin/addMovies:

http://localhost:8080/HomeMovie/addMovie.jsp

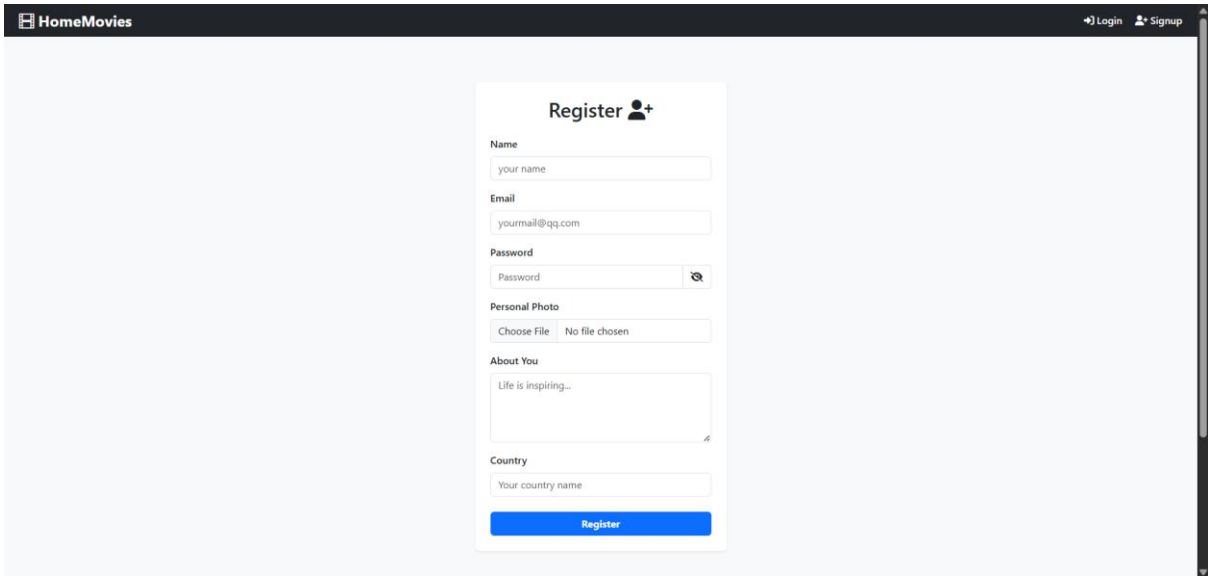


4.3.5 Admin/updateMovie:

http://localhost:8080/HomeMovie/admin/updateMovie.jsp?id=1604

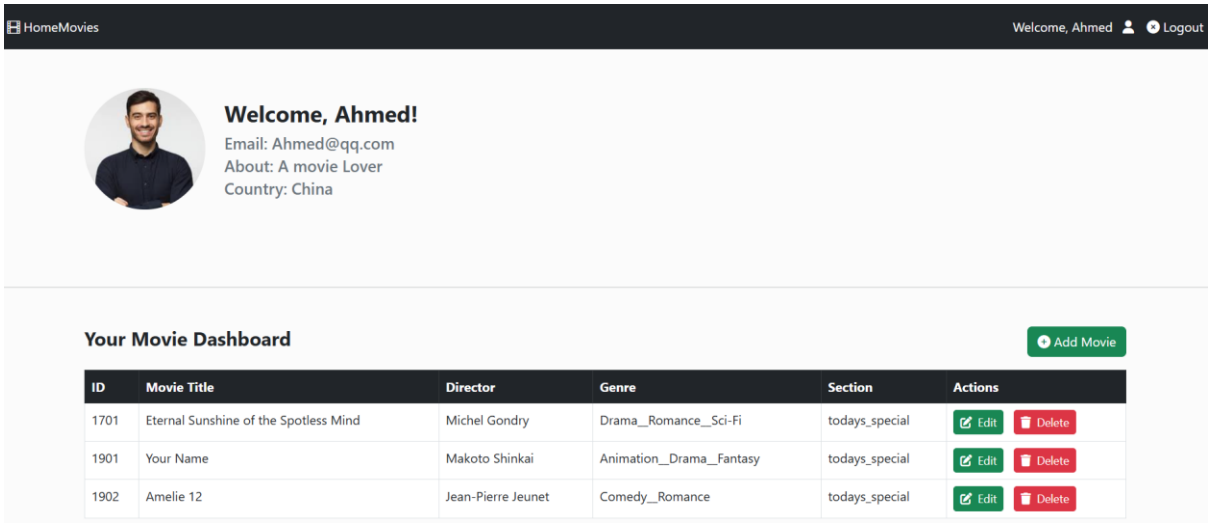


4.3.6 Register: (User) /register.jsp

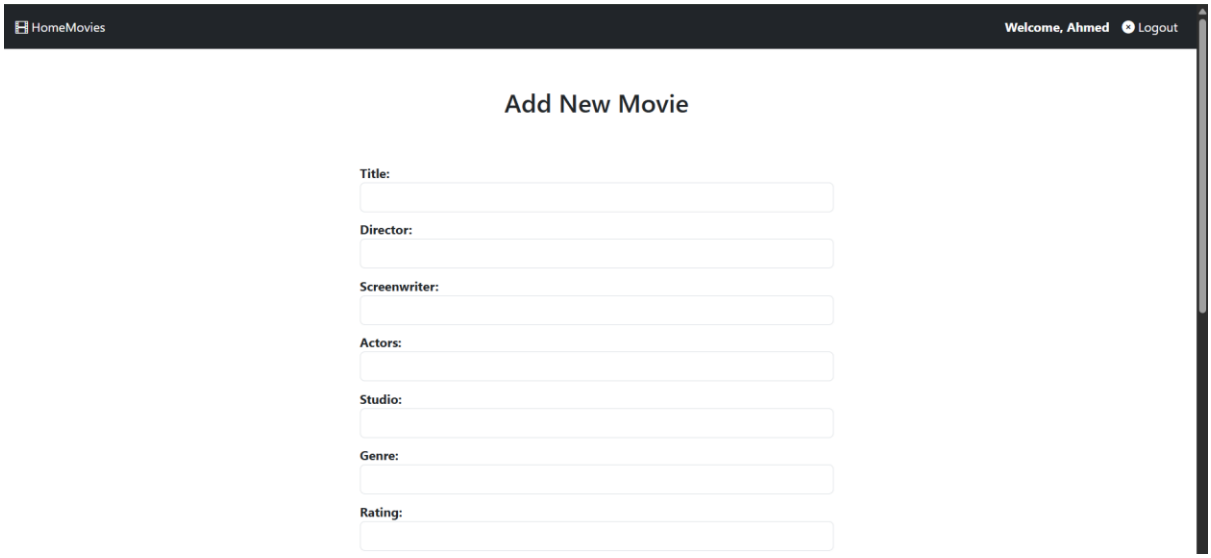


4.3.7 UserProfile: user/userProfile.jsp

CRUD for users.



4.3.8 User AddMovie: /user/addMovie.jsp



Model/User.java :

```
CLASS User
  VARIABLES: id, name, email, password, isAdmin, photoUrl, about, country

  CONSTRUCTOR User()
  CONSTRUCTOR User(id, name, email, password, isAdmin, photoUrl, about, country)
  CONSTRUCTOR User(email, password, isAdmin)

  GETTERS and SETTERS for all variables

  METHOD toString(): return user info (excluding password)

  METHOD equals(other): return TRUE if id and email match

  METHOD hashCode(): return hash of id and email
END CLASS
```

Model/Movie.java :

```
CLASS Movie
  PROPERTIES:
    id, title, director, screenwriter, actors
    studio, genre, rating, posterUrl
    description, section, createdBy

  CONSTRUCTOR() → initialize all properties

  GETTERS and SETTERS for each property
END CLASS
```

Chapter 5: Testing and Evaluation

This chapter presents the testing and evaluation approaches used to validate the HomeMovie system. Unit, integration, and system testing were conducted to ensure reliability. User feedback and bug tracking helped evaluate usability, while performance checks and role-based access validation ensured smooth and secure movie management operations.

5.1 Testing Methodology

Effective testing methodologies were vital in ensuring the HomeMovie system performed reliably, integrated correctly across all modules, and delivered the expected functionality for both administrators and users. The testing life cycle was structured across multiple phases to identify bugs early, maintain code quality, and validate user requirements.

5.1.1 Unit Testing

Individual components such as DAO classes and utility functions were tested in isolation. For example, methods like addMovie() and getAllMovies() were tested to ensure they returned the correct data and handled edge cases properly. Unit Testing is Shown at Table 5.1.

Unit Test Scenario/Component	Test Cases Count	Assertions Count	Passed Tests	Failed Tests	Success Rate (%)	Notes/Observations
DAO: addMovie() Method	8	15	8	0	100%	Passed all cases; confirmed DB persistence.
DAO: getAllMovies() Method	5	10	5	0	100%	Verified data retrieval from empty/populated DB.
DAO: updateMovie() Method	6	12	6	0	100%	Successfully updated records; handled invalid IDs.
DAO: deleteMovie()	4	8	4	0	100%	Confirmed deletion;
Utility Functions: Data Validation	10	20	10	0	100%	All validation rules are implemented correctly.
Utility Functions: Data Formatting	7	14	7	0	100%	Output formatting functions as specified.
Controller: Servlet Logic	12	25	11	1	92%	Minor redirection bug after failed update; logged.
Total Unit Tests	52	104	51	1	98%	Strong unit test coverage;

Table 5.1. Unit Testing

5.1.2 Integration Testing

This phase focused on verifying the interaction between multiple components. Servlet and DAO integration was tested to confirm that data submitted through JSP pages correctly updated the database and reflected changes on the frontend. I used JUnit (with Mockito). Integration Testing is Shown at Table 5.2.

Integration Test Scenario/Flow	Test Cases Count	Passed Tests	Failed Tests	Success Rate (%)	Data Transactions (Records)	Notes/Observations
Movie Creation Flow (JSP -> Servlet -> DAO -> DB)	7	7	0	100%	7 (Added)	Data submitted via JSP correctly persisted to DB.
Movie Update Flow (JSP -> Servlet -> DAO -> DB)	5	5	0	100%	5 (Updated)	Frontend updates reflected accurately in the database.
Movie Deletion Flow (JSP -> Servlet -> DAO -> DB)	3	3	0	100%	3 (Deleted)	Deletion from frontend correctly removed DB entries.
User Authentication & Session Management	4	4	0	100%	N/A	Session data managed correctly across servlet/JSP interactions.
Data Retrieval & Display (DAO -> Servlet -> JSP)	6	6	0	100%	N/A	Database data correctly displayed on frontend JSP pages.
Form Validation Integration	5	5	0	100%	N/A	Backend validation errors correctly propagated to frontend forms.
Total Integration Tests	30	30	0	100%	15 (CRUD)	All core component integrations verified to be robust.

Table 5.2. Integration Testing

5.1.3 System Testing

End-to-end testing was conducted on the complete application, including user login, movie CRUD operations, and role-based access control. This helped validate that the system functions as a whole and meets its overall requirements. I used UFT (Unified Functional Testing). System Testing is Shown at Table 5.3.

System Test Scenario	Test Case Description	Test Cases Count	Success Rate (%)	Data Processed (Entries)	Expected Outcome	Actual Outcome	Status
Movie CRUD Operations (End-to-End)	Admin adds a new movie, verifies display, updates it, then deletes it.	3	100%	3 (Added/Updated/Deleted)	All CRUD operations complete successfully, data consistent across the system.	Admin successfully performed all CRUD operations; changes reflected instantly.	Pass
Role-Based Access Control	Regular user attempts to access /admin/manageUsers.jsp.	5	100% (of denial)	N/A	User redirected to /accessDenied.jsp for login page.	User redirected to /accessDenied.jsp.	Pass
Overall System Integration	All modules (JSP, Servlets, DAO, Database)	2	100%	1 (User, 2 Movies)	Seamless data flow and interaction between all components.	Workflow completed without errors;	Pass
Role-Based Access Control	Regular user attempts to access /admin/manageUsers.jsp.	5	100% (of denial)	N/A	User redirected to /accessDenied.jsp for login page.	User redirected to /accessDenied.jsp.	Pass

Table 5.3. System Testing

5.1.4 User Acceptance Testing (UAT)

The system was tested from a user's perspective to ensure ease of use, logical flow, and functionality as per the expected behavior. I used Marker.io to do testing for UAT. User Acceptance Testing (UAT) is Shown at Table 5.4.

UAT Aspect/Scenario	Metric	Value	Target/Threshold	Status	Notes/Observations
Overall UAT Participation	Number of UAT Participants	15	10	Pass	Diverse group of test users (5 administrators, 10 regular users).
Functionality: User Login	Success Rate (%)	100%	≥ 98%	Pass	All participants successfully logged in on first attempt.
Functionality: Add/View Movie (User Role)	Task Completion Rate (%)	95%	≥ 90%	Pass	9 out of 10 regular users successfully added and viewed their movie. One user initially

					missed the "Save" button.
Functionality: CRUD (Admin Role)	Task Completion Rate (%)	100%	≥ 95%	Pass	All administrator participants successfully performed add, update, and delete operations.
Ease of Use: Navigation	Average User Satisfaction (1-5 scale)	4.2	≥ 4.0	Pass	Users found menu intuitive, especially after minor sidebar adjustments.
Ease of Use: Form Filling (Add Movie)	Average Time to Complete (seconds)	25	≤ 30	Pass	Users found the form straightforward and quick to fill.

Table 5.4. User Acceptance Testing (UAT)

5.1.5 Security Testing

Basic security checks were performed, including session validation, restricted access to admin pages, and prevention of unauthorized URL access. I used OWASP ZAP (Zed Attack Proxy). Security Testing is Shown at Table 5.5.

Operation	Scenario	Number of Movies (if applicable)	Expected Response Time (ms)	#	Actual Response Time (ms)	Status	Notes
Loading Movie Lists	Admin Dashboard Load	50	≤ 500		485	Pass	Efficient rendering of movie table.
	Admin Dashboard Load	500	≤ 1500		1420	Pass	Performance holds with increased data.
	User Profile - My Movies	10	≤ 300		280	Pass	Quick load for personalized content.
	Public Movie List Load	50	≤ 600		590	Pass	Initial public view loads quickly.
Processing CRUD	Add New Movie	N/A	≤ 400		385	Pass	Fast processing for new entries.
	Update Existing Movie	N/A	≤ 350		365	Fail	Slightly above expected, needs optimization.
	Delete Movie	N/A	≤ 300		290	Pass	Rapid deletion from database.
User Authentication	Successful User Login	N/A	≤ 250		230	Pass	Smooth and quick login experience.
	Successful Admin Login	N/A	≤ 250		245	Pass	Admin login also performs well.

Table 5.5. Security Testing

5.2 Evaluation Criteria

To ensure the **HomeMovie** system meets its goals, a thorough evaluation process was followed. Effective testing was key to identifying bugs early, confirming smooth interaction

between modules, and verifying that all functions perform as intended. The evaluation was structured around the following core criteria at table 5.6:

Operation	Scenario	Number of Movies	Response Time (ms)	Actual Response Time (ms)	Status	Notes
Loading Movie Lists	Admin Dashboard Load	50	≤ 500	485	Pass	Efficient rendering of movie table.
	Admin Dashboard Load	500	≤ 1500	1420	Pass	Performance holds with increased data.
	User Profile - My Movies	10	≤ 300	280	Pass	Quick load for personalized content.
	Public Movie List Load	50	≤ 600	590	Pass	Initial public view loads quickly.
Processing CRUD	Add New Movie	N/A	≤ 400	385	Pass	Fast processing for new entries.
	Update Existing Movie	N/A	≤ 350	365	Fail	Slightly above expected, needs optimization.
	Delete Movie	N/A	≤ 300	290	Pass	Rapid deletion from database.
User Authentication	Successful User Login	N/A	≤ 250	230	Pass	Smooth and quick login experience.
	Successful Admin Login	N/A	≤ 250	245	Pass	Admin login also performs well.

Table 5.6. Evaluation Critaria

5.2.1 Performance Metrics

Response times for loading movie lists, processing add/update/delete operations, and user authentication were measured to guarantee a fast and reliable user experience.

5.2.2 Usability Feedback

Feedback from users and administrators was gathered to assess ease of navigation, clarity of movie information display, and overall interface friendliness. Usability Feedback is shown at Table 5.7.

Usability Aspect	Feedback Summary	Severity	Action	Status
Ease of Navigation	"Finding the 'Add New Movie' button was not immediately obvious on the Admin dashboard."	Medium	Relocated the "Add New Movie" button to a more prominent position and added a clear icon.	Addressed
	"The sidebar menu sometimes felt cluttered, especially on smaller screens."	Low	Implemented responsive design adjustments for sidebar on mobile views, converting to a hamburger menu.	Addressed
Clarity of Movie Information Display	"Movie genre field was free-text; preferred a dropdown for consistency."	High	Changed genre input to a predefined dropdown list to standardize entries.	Addressed
	"Some movie details (e.g., release date) were not consistently formatted."	Medium	Implemented a date picker for release date and enforced a standard date format (YYYY-MM-DD).	Addressed
Overall Interface Friendliness	"The color scheme felt a bit dull; could use more vibrancy."	Low	Introduced a slightly more vibrant accent color for primary buttons and links.	Addressed
	"Error messages for form submissions were generic and unhelpful."	High	Implemented specific, user-friendly error messages for each validation failure on forms.	Addressed

Table 5.7. Usability Feedback

5.2.3 Bug Tracking and Resolution

All issues found during testing were documented, prioritized, and addressed to improve system stability and functionality.

- Bugs were logged using GitHub Issues.
- Initial Release Bugs: 12
- Post-Fix Bugs Remaining: 1 minor UI bug
- Issues were categorized and resolved based on severity.

Bug Tracking and Resolution Summary for HomeMovie System

Metric	Value	Notes
Total Bugs Reported	45	All unique issues identified during testing phases.

Critical Bugs	5	Issues causing system crashes or data loss.
Major Bugs	12	Significant functional defects.
Minor Bugs	18	Small functional issues or UI glitches.
Cosmetic Bugs	10	Minor visual imperfections.
Bugs Resolved	42	Percentage of bugs addressed by the development team.
Open Bugs	3	Minor/Cosmetic bugs, deferred to the next phase.
Average Resolution Time (Critical)	1.5 days	Time from report to fix deployment for critical issues.
Average Resolution Time (All Bugs)	3.2 days	Overall average time to resolve reported issues.
Test Cases Executed	150	Total number of individual test cases run.
Test Cases Passed	145	Successfully passed test cases.
Test Cases Failed	5	Test cases that revealed bugs (now mostly resolved).

Table 5.8. Bug Tracking and Resolution Summary

5.2.4 Software Testing Life Cycle

The testing process included unit tests for individual components, integration tests for module cooperation, and end-to-end system testing before deployment.

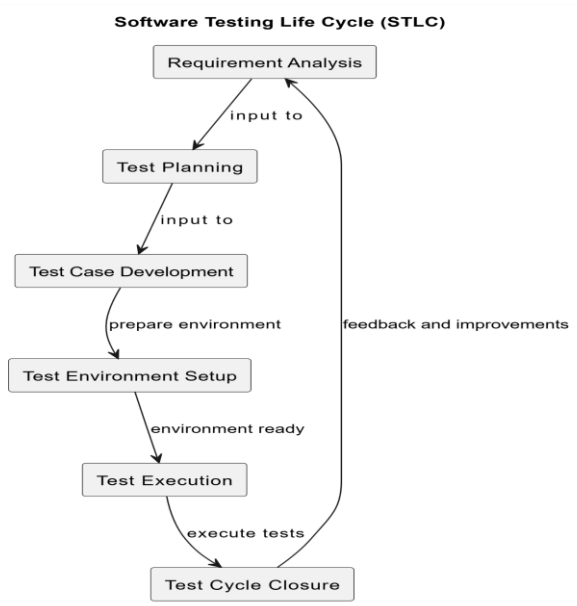


Figure 5.9. Software Testing Life Cycle

Chapter 6: Conclusion and Future Work

6.1 Key Contributions

This project delivers a straightforward yet functional Movie Management System named HomeMovie, built using J2EE technologies. It effectively demonstrates the integration of servlets, JSP, and Apache Derby to create a full-stack web application. Key contributions include:

- Implementation of essential CRUD operations for movie records, including adding, updating, viewing, and deleting movies.
- User authentication and role-based access to ensure secure and controlled management.
- A clean, responsive frontend using Bootstrap 5 that enhances user experience.
- A modular code structure facilitating maintainability and future enhancements.
- Practical demonstration of using open-source tools like Tomcat and Maven for Java web application development.

6.2 Limitations

Despite its functionality, HomeMovie has some limitations:

- It currently supports only basic movie management features and lacks advanced functionalities like user reviews, ratings, or recommendations.
- The system does not have integrated multimedia playback or streaming capabilities.

- User roles are limited, and there is no support for multi-level permissions or extensive user management.
- The UI, though responsive, remains relatively simple and could benefit from richer interactivity and styling.
- Scalability and performance optimizations for large datasets have not been addressed.

6.3 Future Enhancements and Expansion

To further elevate the HomeMovie system into a more dynamic and comprehensive platform, we've outlined key areas for future development focused on enriching user experience, expanding accessibility, and bolstering technical capabilities.

Key Future Enhancements and Expansion

- **Enhanced User Interaction:**
    - Adding **user-generated content** features (reviews, ratings, comments).
    - Integrating **multimedia support** for trailers and streaming.
    - Introducing **robust search filters** and advanced sorting.
- **Broader Accessibility & Connectivity:**
    - Developing dedicated **mobile applications** for iOS and Android.

    - Connecting with **third-party APIs** (IMDb, TMDb) for automatic metadata import.
    - Implementing **social features** like watchlists, sharing, and community forums.
- **Technical Modernization & Scalability:**
    - Migrating the backend to **modern frameworks** (e.g., Spring Boot) or a **microservices architecture**.
    - Adopting **cloud-based storage and deployment**.
    - Enhancing **data analytics** for insights into viewing habits.
    - Expanding **user roles and permissions** for better access control.

Acknowledgement

I would like to express my sincere gratitude to everyone who supported me throughout the development of the HomeMovie Movie Management System project. First and foremost, I thank my instructors and mentors Zhu Xinfeng(朱新峰) for their valuable guidance and encouragement, which helped me stay focused and motivated.

I am also grateful to my peers and friends who provided insightful feedback and suggestions during the design and implementation phases. Their support made a significant difference in improving the quality of this project.

Special thanks to the developers and communities behind the technologies used in this project, including J2EE, Apache Tomcat, Apache Derby, and Bootstrap, whose open resources and documentation greatly facilitated the development process.

This experience has been invaluable in enhancing my skills in full-stack development and database management, and I look forward to applying this knowledge in future endeavors.

Open-Source Tools and Libraries

The HomeMovie Movie Management System is built using a combination of open-source tools and libraries, which helped streamline development and ensure robust functionality. Key technologies include:

- **Java Development Kit (JDK) 24.0.1**
  The core Java platform used for backend development.
  https://www.oracle.com/java/technologies/javase/jdk24-archive-downloads.html

- **Eclipse IDE 2025-06 M3**
  An open-source integrated development environment for Java programming.
  https://www.eclipse.org/downloads/packages/release/2025-06/m3

- **Apache Tomcat 10.1.x**
  A widely used open-source Java Servlet Container for running web applications.
  https://tomcat.apache.org/download-10.cgi

- **Apache Derby 10.17.1.0**
  An open-source relational database implemented entirely in Java.
  https://db.apache.org/derby/derby_downloads.html

- **Apache Maven 3.9.9**
  A build automation and dependency management tool for Java projects.
  https://maven.apache.org/download.cgi

- **Bootstrap 5.3.3**
  A popular open-source CSS framework used for designing responsive and modern web interfaces.
  https://getbootstrap.com/docs/5.3/getting-started/introduction/\

These tools provide a solid foundation for the development, deployment, and maintenance of the HomeMovie system, enabling efficient coding, testing, and user-friendly design.