PekkaTV
DATABASE MANAGEMENT SYSTEM PROJECT PHASE-II REPORT

Ece Nur BAYRAKTAR
2022510102
ecenur.bayraktar@ogr.deu.edu.tr

Eda PINTZAL
2023510062
eda.pintzal@ogr.deu.edu.tr

Serkan AYAŞAN
2022510058
serkan.ayasan@ogr.deu.edu.tr

Lecturer
Prof. Dr. Semih Utku

IZMIR
23.12.2025

TABLE OF CONTENTS

1. Abstract

This project presents the design and implementation of a movie review web application that allows users to browse movies, view detailed information, and manage personal interactions such as reviews and watched lists. The system is developed using a layered architecture based on the Model–View–Controller (MVC) pattern, with Spring Boot handling backend logic, Thymeleaf managing dynamic views, and SQL Server used for persistent data storage. Core functionalities include movie listing, filtering and sorting, user authentication, review management, and user-specific watch history tracking. The application emphasizes modular design, data consistency, and extensibility, providing a solid foundation for future recommendation and analytics features.

2. Completion Report

This project successfully achieved its primary objective of designing and implementing a database-driven movie review web application with core user interaction functionalities. The system supports essential features such as movie browsing, detailed movie views, filtering and sorting mechanisms, user authentication, review management, and maintenance of a personal watched list. The application was implemented using a structured Model–View–Controller (MVC) architecture, ensuring a clear separation of concerns and improved maintainability. Database operations were handled through a relational schema designed to preserve data integrity and support extensibility.

While the core system functionalities were completed as planned, some advanced features initially considered during the project's early stages were not fully implemented. These include a personalized recommendation system based on watched history and genre preferences, as well as advanced analytics features for user behavior. The main reasons for these limitations were time constraints and the increased complexity of integrating recommendation algorithms within the existing project timeline.

Several technical challenges were encountered during the implementation phase. One of the key challenges involved ensuring consistency between the database schema and the application layer, particularly during data import and entity mapping processes. Additionally, integrating filtering and sorting features while maintaining query performance required careful query design and iterative testing. These challenges were addressed through incremental development, debugging, and refactoring, resulting in a stable and functional system.

Overall, the project met its fundamental design and implementation goals, delivering a robust and extensible application while providing valuable experience in full-stack database application development.

3. Functional Decomposition

This project aims to design and implement a database-driven movie review web application that enables users to browse movies, access detailed movie information, and perform personalized interactions such as writing reviews and maintaining a watched list. The primary objective of the system is to provide a structured, user-oriented platform where movie-related data is presented in an organized and accessible manner, while ensuring data consistency, usability, and extensibility. The application focuses on core database management system concepts, including data storage, retrieval, and relationship management, within a web-based environment.

To achieve this objective, the system is decomposed into a set of core functional components, each responsible for a specific aspect of the application’s behavior. These components include movie data management, user authentication and account handling, review management, and user-specific watched list operations. By separating the system into well-defined functional units, the application ensures modularity and clarity, allowing each functionality to be developed, maintained, and extended independently while still operating cohesively as part of the overall system.

The application is organized using an MVC-based architectural structure to support this functional decomposition. In this structure, the Model layer represents the application’s data entities and their relationships, the View layer is responsible for presenting dynamic content to the user through Thymeleaf templates, and the Controller layer manages user requests and coordinates interactions between the model and the view. This approach provides a clear separation of responsibilities without enforcing a strictly layered architecture, making it suitable for a database-oriented academic project while maintaining readability and maintainability.

In the following subsections, each of the main functional components of the system is explained in detail in order to clarify their responsibilities, interactions, and contributions to the overall operation of the application.

3.1 Movie Management

The Movie Management component is responsible for managing all movie-related data and interactions within the application. It forms the core of the system by enabling users to browse the movie catalog, access detailed movie information, and view related content such as reviews and cast information. Movie data is retrieved from the database and dynamically presented to users, ensuring a consistent and informative browsing experience.

This component is structured into multiple sub-functional areas in order to clearly separate responsibilities while maintaining cohesive integration within the movie detail view. These sub-functional areas include general movie information handling, review management, and actor information presentation.

3.1.1 Movie Information and Browsing

This sub-component handles the presentation of general movie information and supports core browsing functionalities. Users can view a list of movies and access individual movie detail pages that display attributes such as title, release year, genre, rating, and description. To improve usability, the system provides filtering and sorting options that allow users to refine movie results based on criteria such as release year or genre and organize them according to predefined ordering preferences.

These functionalities enable efficient navigation through the movie catalog and form the basis for all movie-related interactions within the system.

3.1.2 Review Management

The Review Management sub-component is responsible for handling user-generated reviews associated with movies. Reviews are retrieved through the movie-related request flow and displayed directly on the movie detail page, allowing users to view feedback and opinions in the same context as the movie information.

In order to submit a review, users are required to be registered and authenticated. Only logged-in users are permitted to write reviews and provide ratings for movies, ensuring that all reviews are associated with valid user accounts. This design choice supports accountability and data integrity within the review system.

Additionally, the system provides a spoiler management feature that allows users to mark their reviews as containing spoilers. Reviews flagged as spoilers are displayed in a blurred format by default, preventing unintended content exposure. Users may choose to reveal these reviews manually, which enhances user experience by respecting individual viewing preferences while maintaining transparency.

3.1.3 Actor Information

The Actor Information sub-component is designed to present cast-related data for each movie in a structured manner. On the movie detail page, a dedicated section displays the actors who participated in the movie, allowing users to easily access information about the cast without navigating away from the movie page.

By including actor information directly within the movie detail view, the system enriches the contextual understanding of each movie and improves content completeness. This feature also lays the groundwork for potential future enhancements, such as actor-based search or cross-movie navigation.

3.2 User Authentication and Account Management

The User Authentication and Account Management component is responsible for managing user identities, access control, and personalized user interactions within the system. This component ensures that only authenticated users can perform actions that require authorization, such as writing reviews or managing personal movie lists, thereby maintaining data security and integrity throughout the application.

User account functionality includes essential operations such as account creation, authentication, and session-based access control. In addition to basic authentication mechanisms, users are provided with account management capabilities that allow them to update sensitive information, including changing their account password, as well as permanently deleting their account if they choose to do so. These features enhance user control over personal data and align with standard user-centered system design principles.

Beyond authentication, this component also enables users to personalize their interaction with the movie catalog. Authenticated users can mark movies as favorites, add movies to their watched list, or include them in a watch list intended for future viewing. These lists are maintained on a per-user basis and are dynamically linked to the movie data, allowing users to track their viewing history and organize movies according to their preferences. The separation between watched movies and movies intended to be watched supports clearer categorization and improves overall usability.

By integrating account management with personalized movie interaction features, this component plays a critical role in delivering a tailored user experience. It also provides a flexible foundation for future enhancements, such as recommendation systems or user behavior analysis, without requiring significant changes to the existing system structure.

3.3 Diagrams

The functional structure described above is further clarified through a set of system diagrams presented in the following sections. These diagrams include the Entity–Relationship (ER) diagram, class diagram, and other supporting design diagrams (class, activity, use case etc.), which visually represent the data model and interactions among system components. Together, these diagrams provide a comprehensive overview of the system architecture and complement the functional explanations provided in this section.

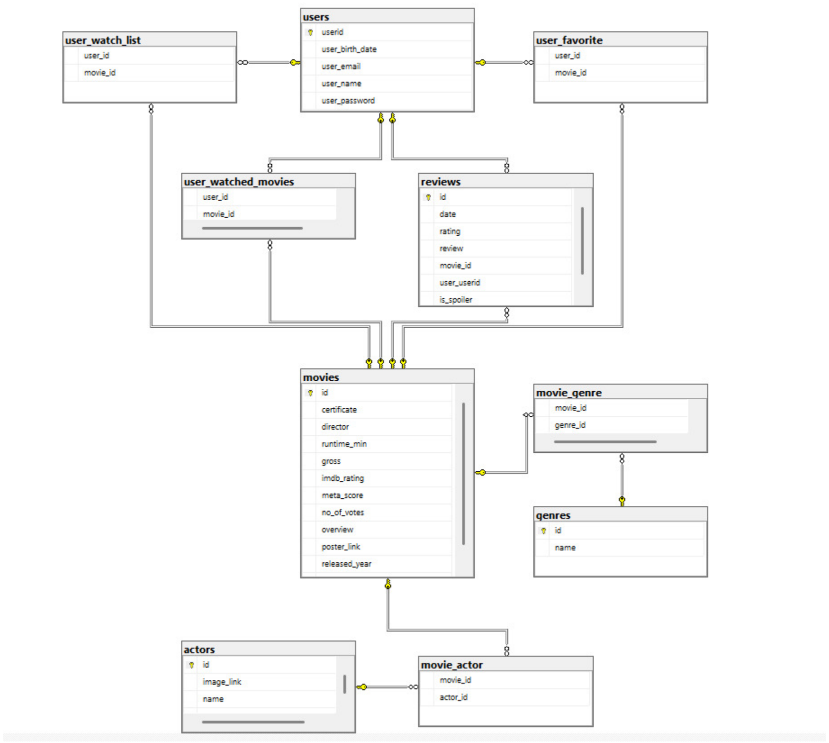


Figure 3.3.1 Database Diagram

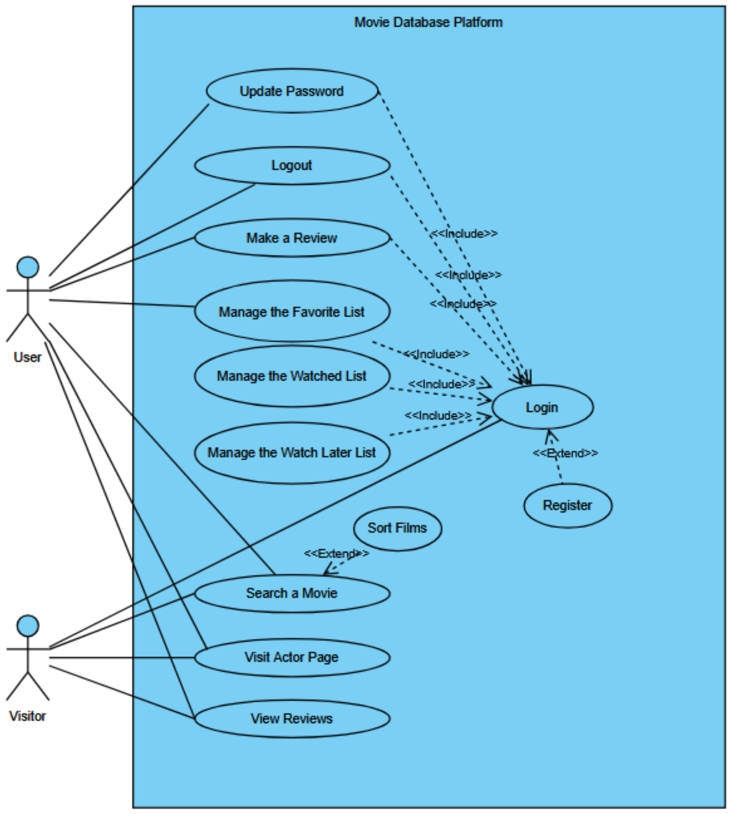


Figure 3.3.2 Use Case Diagram

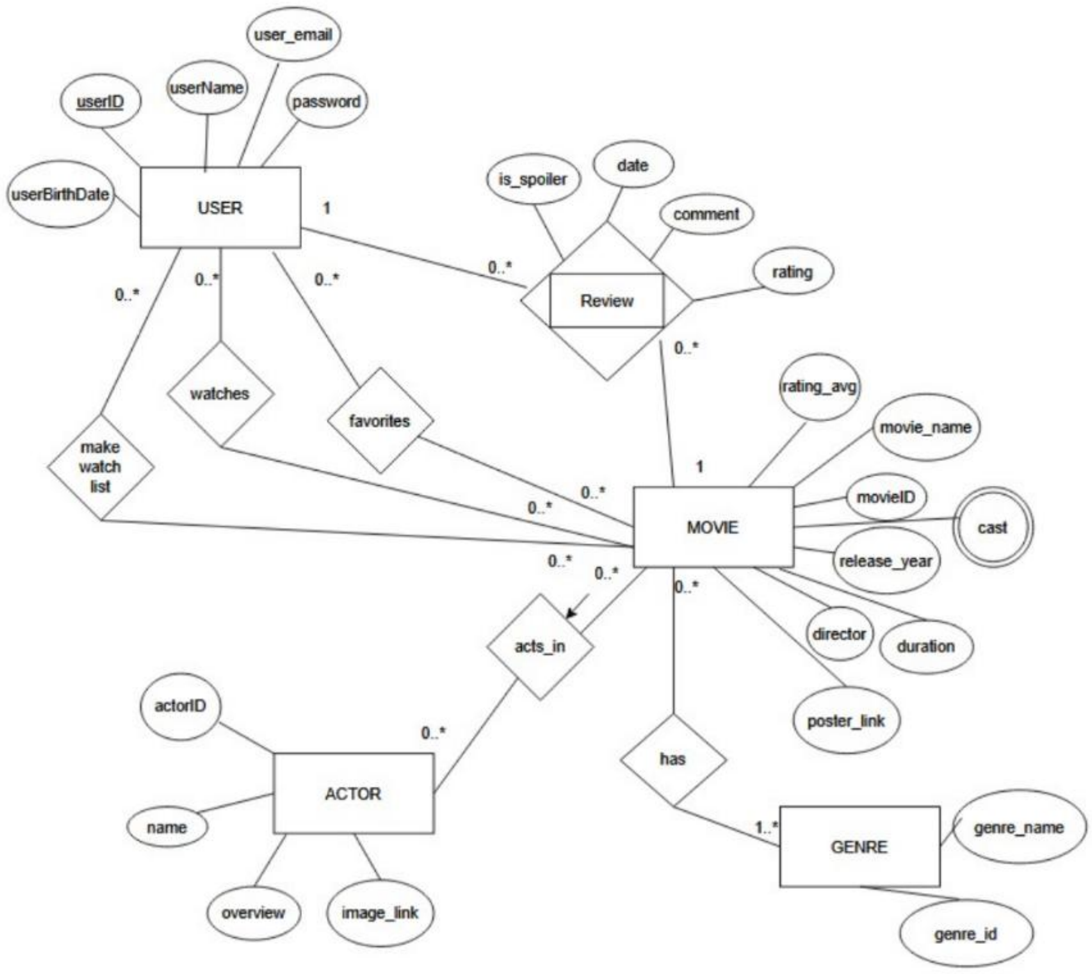


Figure 3.3.3 ER Diagram

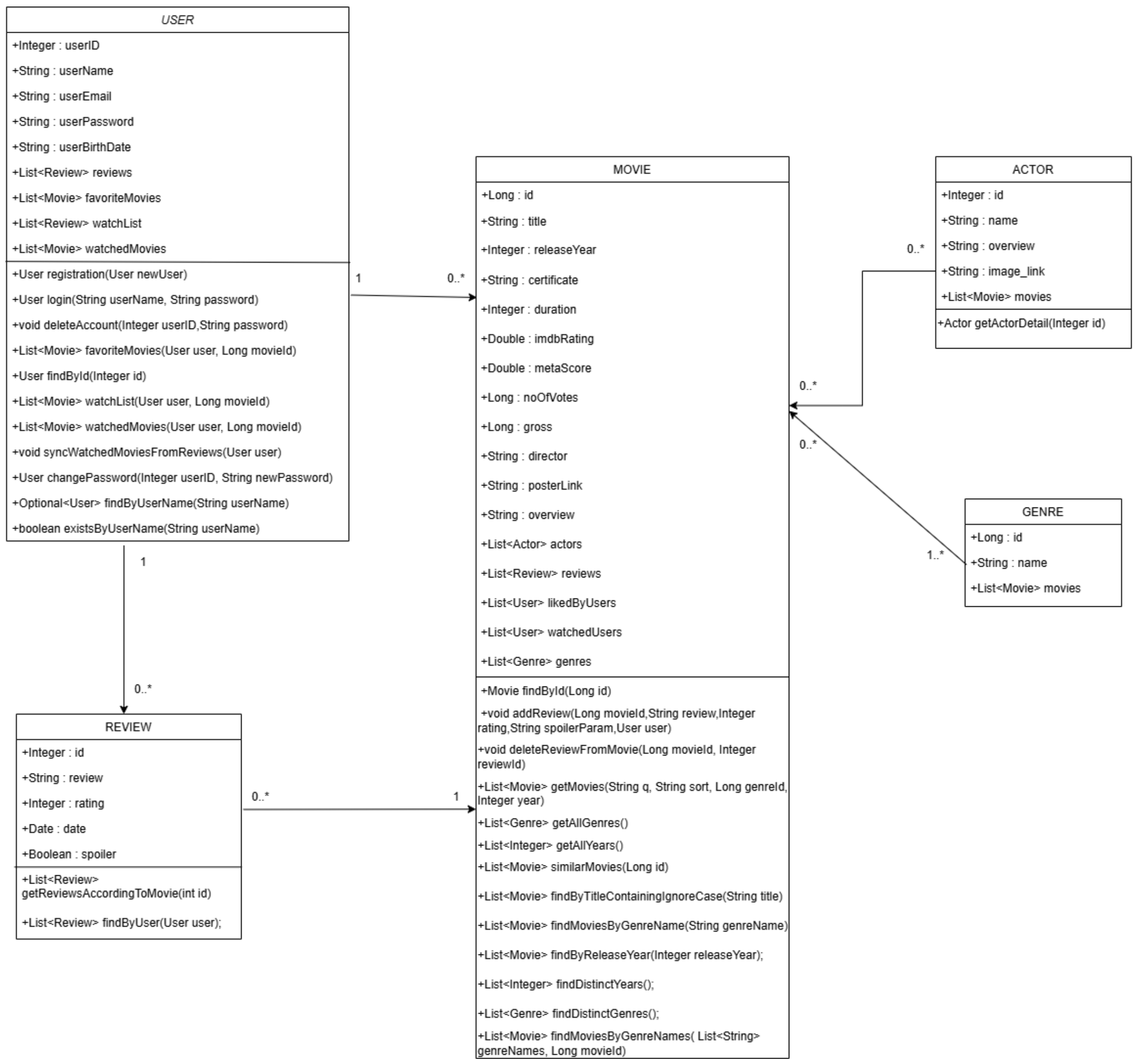

USER
+Integer : userID
+String : userName
+String : userEmail
+String : userPassword
+String : userBirthDate
+List<Review> reviews
+List<Movie> favoriteMovies
+List<Review> watchList
+List<Movie> watchedMovies
+User registration(User newUser)
+User login(String userName, String password)
+void deleteAccount(Integer userID,String password)
+List<Movie> favoriteMovies(User user, Long movieId)
+User findById(Integer id)
+List<Movie> watchList(User user, Long movieId)
+List<Movie> watchedMovies(User user, Long movieId)
+void syncWatchedMoviesFromReviews(User user)
+User changePassword(Integer userID, String newPassword)
+Optional<User> findByUserName(String userName)
+boolean existsByUserName(String userName)
1
0..*
1
0..*
MOVIE
+Long : id
+String : title
+Integer : releaseYear
+String : certificate
+Integer : duration
+Double : imdbRating
+Double : metaScore
+Long : noOfVotes
+Long : gross
+String : director
+String : posterLink
+String : overview
+List<Actor> actors
+List<Review> reviews
+List<User> likedByUsers
+List<User> watchedUsers
+List<Genre> genres
+Movie findById(Long id)
+void addReview(Long movieId,String review,Integer rating,String spoilerParam,User user)
+void deleteReviewFromMovie(Long movieId, Integer reviewId)
+List<Movie> getMovies(String q, String sort, Long genreId, Integer year)
+List<Genre> getAllGenres()
+List<Integer> getAllYears()
+List<Movie> similarMovies(Long id)
+List<Movie> findByTitleContainingIgnoreCase(String title)
+List<Movie> findMoviesByGenreName(String genreName)
+List<Movie> findByReleaseYear(Integer releaseYear);
+List<Integer> findDistinctYears();
+List<Genre> findDistinctGenres();
+List<Movie> findMoviesByGenreNames(List<String> genreNames, Long movieId)
ACTOR
+Integer : id
+String : name
+String : overview
+String : image_link
+List<Movie> movies
+Actor getActorDetail(Integer id)
0..*
0..*
0..*
1..*
GENRE
+Long : id
+String : name
+List<Movie> movies
REVIEW
+Integer : id
+String : review
+Integer : rating
+Date : date
+Boolean : spoiler
+List<Review> getReviewsAccordingToMovie(int id)
+List<Review> findByUser(User user);
0..*
1

Figure 3.3.4 Class Diagram

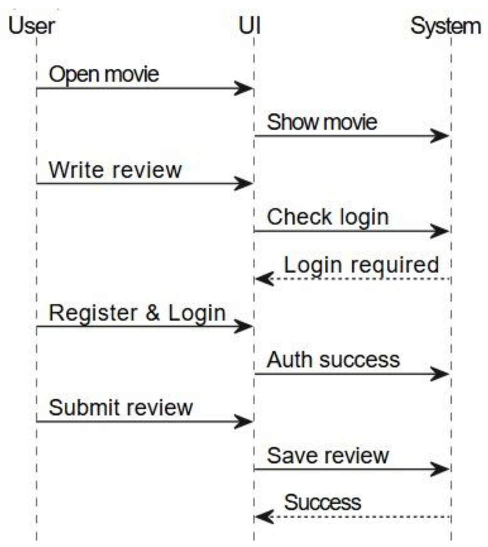


Figure 3.3.5 Sequence Diagram 1

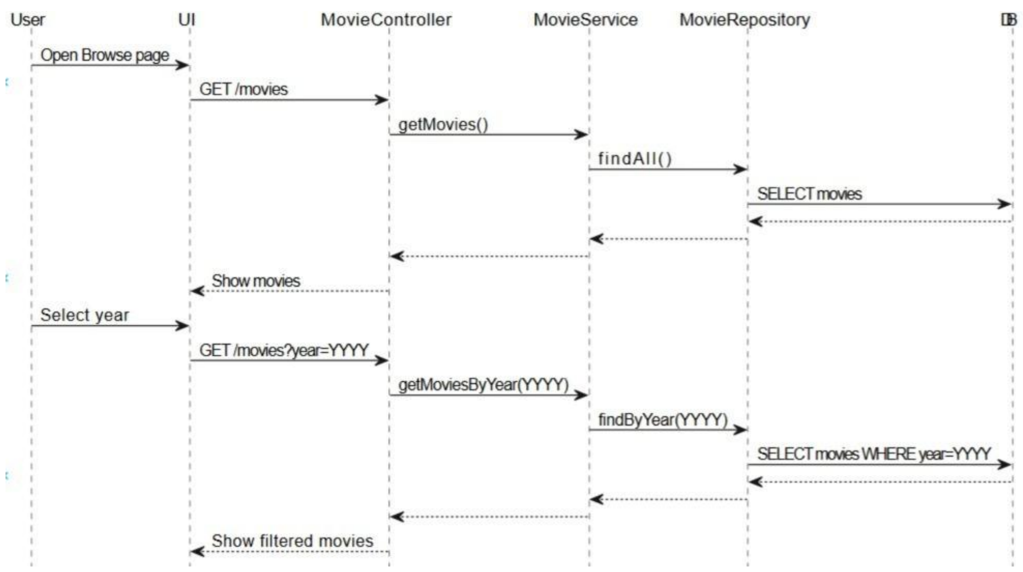
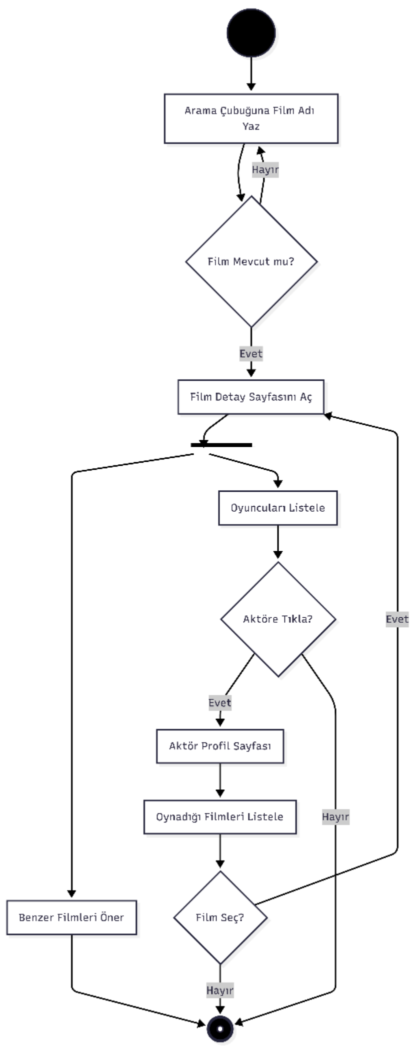


Figure 3.3.6 Sequence Diagram 2



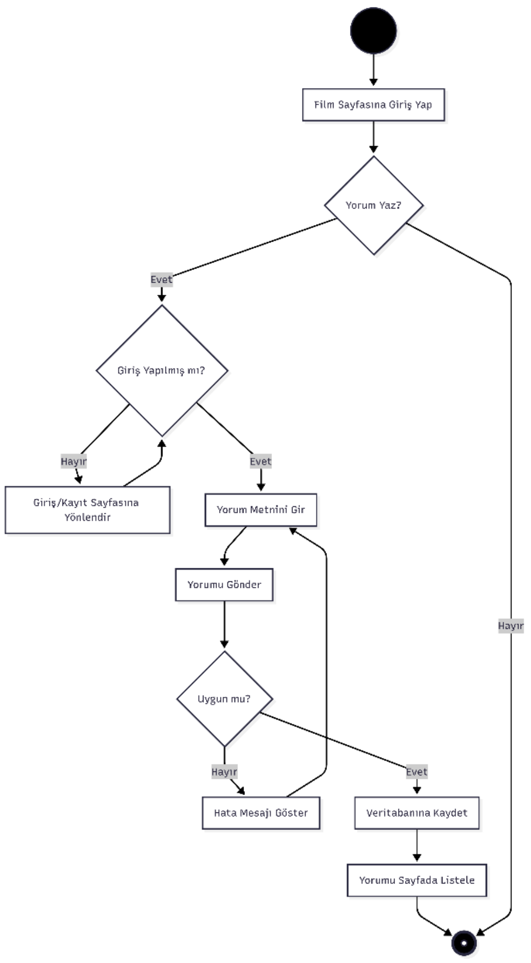


Figure 3.3.8 Activity Diagram 2

Figure 3.3.7 Activity Diagram 1

4. High-Level Organization

The application is an **IMDb-like web application** designed using a **layered MVC architecture**. The system is organized into clearly separated components to ensure **maintainability, scalability, and data integrity**. At a high level, the application supports both **visitor** and **authenticated user** flows, including browsing/searching films and actors, and performing user-specific actions such as managing favorites and watchlists.

4.1. Presentation Layer (Views)

The presentation layer provides the user-facing pages and navigation flow. It is responsible for:

- Rendering movie lists, search results, and sorting/filtering views
- Displaying movie detail pages including cast, recommended movies, and user reviews
- Displaying actor detail pages including biography and filmography
- Providing authentication and account-management pages (login, register, password change, account deletion)
- Exposing user library pages for favorites, watched, and watch-later collections

This layer does not directly communicate with the database; it only presents data returned by the application layer.

4.2. Controller Layer (Request Handling)

Controllers act as the entry point for user actions and HTTP requests. They coordinate:

- Visitor flows: browsing, searching, sorting, viewing movie/actor pages
- Authentication flows: register, login, logout
- User flows: adding/removing favorites, watched, and watch-later items
- Account operations: password change and account deletion

Controllers validate incoming requests at a basic level and delegate business operations to the service layer, ensuring that data access is not performed directly in the controller layer.

4.3. Service Layer (Business Logic)

Services implement the application’s core rules and workflows, including:

- Search and sorting logic for movies (e.g., by title, year, rating, popularity)
- Fetching movie detail aggregates (movie info + cast + recommendations + reviews)
- Fetching actor detail aggregates (actor info + filmography)
- Enforcing user-specific rules (e.g., only authenticated users can manage lists)
- Managing user collections: favorites, watched, and watch-later (add/remove/list)
- Account management operations such as password update and safe user deletion

The service layer centralizes business logic so that changes to rules do not require modifications in the view or controller layers.

4.4 Data Access Layer (Repositories)

The data access layer encapsulates all interactions with the DBMS. Its responsibilities include:

- Executing database queries for movies, actors, genres, and reviews
- Supporting search and sort queries efficiently
- Managing relationships between entities such as movies–actors and movies–reviews
- Persisting and querying user-library relations (favorites/watched/watch-later)

By isolating database operations in this layer, the system maintains consistent data access patterns and reduces coupling.

4.5. Database Layer (DBMS)

The database layer is responsible for storing and managing all **persistent data** required by the application. It maintains structured records for core entities and their relationships, including:

- Entity data representing movies, actors, users, and related domain objects
- One-to-many, many-to-one, and many-to-many relationships between entities
- User-generated reviews associated with movies and users
- User accounts and user-specific collections such as favorites, watched, and watch-later lists

Data integrity and consistency are enforced through **primary key and foreign key constraints**, as well as relationship constraints defined at the database level. These mechanisms ensure reliable data storage, prevent invalid references, and support consistent query and update operations across the system.

4.6. End-to-End Workflow Summary

1. The user interacts with the web interface (browse/search/sort or manage account/lists).
2. The request is routed to the appropriate controller.
3. The controller delegates processing to services.
4. Services fetch or update data via repositories in a controlled manner.
5. The DBMS returns results or confirms updates, and the response is rendered back to the user.

This high-level organization enables the application to support a feature-rich domain.

5. Clickstreams: A Brief Description of the Common Use Cases

5.1 Visitor Browsing and Discovery

Visitors can browse the platform without authentication. A typical clickstream includes:

- Accessing the home page
- Searching for movies using keywords
- Sorting and filtering movie lists based on user preferences
- Navigating to movie detail pages to view basic information, cast members, recommended movies, and reviews
- Navigating from a movie page to an actor page to explore actor details and filmography

This clickstream supports content discovery and exploration for unauthenticated users.

5.2 User Authentication and Account Access

Visitors may transition into authenticated users through:

- Navigating from the home or movie pages to the login or registration pages
- Registering a new account or logging into an existing account
- Being redirected to personalized sections of the application upon successful authentication

This clickstream enables access to user-specific features.

5.3 Personalized Movie Management

Authenticated users can manage their personal movie collections through the following clickstream:

- Opening a movie detail page
- Adding or removing the movie from favorites
- Marking or unmarking the movie as watched
- Adding or removing the movie from the watch-later list
- Navigating to dedicated pages to view and manage each list

This use case emphasizes personalized interaction with movie content.

5.4 Review Interaction

Users can interact with reviews through:

- Scrolling to the review section on a movie detail page
- Viewing existing user reviews
- Submitting a review with an optional spoiler flag and a rating score

This clickstream supports community-driven content creation and evaluation.

5.5 Actor Exploration

Users can explore actor-related information through:

- Navigating from a movie detail page to an actor profile
- Viewing actor biography and detailed filmography
- Selecting a movie from the actor's filmography to return to a movie detail page

This creates cyclic navigation paths that enhance content discovery.

5.6 Account Management

Authenticated users can manage their accounts through:

- Accessing the settings page
- Updating account information such as password
- Deleting the account, which terminates user access and removes associated user-specific data

These clickstreams reflect the primary interaction patterns supported by the system and illustrate how users navigate between content discovery, personalization features, and account management functionalities.

6. Layout

This section presents the wireframes of the primary views of the movie website. The wireframes illustrate the initial layout design of the system and focus on the placement of interface components rather than visual styling. Since the interface design was kept simple and consistent throughout the development process, the final implementation closely matches these initial layouts.

6.1 Home Page Wireframe

Figure 6.1 shows the wireframe of the home page. This page serves as the main entry point of the application, where users can browse the list of available movies. A search bar is located at the top of the page to allow users to search movies by title. Below the navigation area, movies are displayed in a grid layout, where each movie card represents a single movie with basic information such as title and rating.

LOGO (CineDB) Search Bar
[Movie Card] [Movie Card] [Movie]
[Movie Card] [Movie Card] [Movie]
[Movie Card] [Movie Card] [Movie]

Figure 6.1 Home Page – Movie List

6.2 Movie Detail Page Wireframe

Figure 6.2 illustrates the wireframe of the movie detail page. This page provides detailed information about a selected movie, including its title, rating, duration, genres, and synopsis. Action buttons allow users to add the movie to their favorites or watch-later list. The lower section of the page is dedicated to user reviews, where existing comments are displayed. Users can also submit their own reviews by entering their name, rating, and comment. Spoiler-related content is visually separated to prevent accidental exposure.

| Movie Poster | Movie Title | | | Rating / Year / Duration | | | Genres | | | Synopsis | | | [Add to Favorites] || | [Add to Watchlist] | User Reviews | | - Review 1 |

- Review 2 (Spoiler)
Write a Review
Name: [__________]
Rating: [★ ★ ★ ★ ★]
Comment: [______________]
[Submit Review]

Figure 6.2 Movie Detail Page

6.3 Filter Panel Wireframe

Figure 6.3 presents the wireframe of the filter panel. This panel enables users to refine the movie list based on different criteria such as genre, release year, duration, and minimum rating. The filter options are designed to be simple and intuitive, allowing users to quickly narrow down movies according to their preferences.

Filter Movies [X]
Genre:
[All] [Comedy] [Drama]
[Action] [Sci-Fi]

| Year: |

[All] [2024] [2023]
Duration Slider
[--------O--------]

| Minimum Rating Slider |

[----O------------]

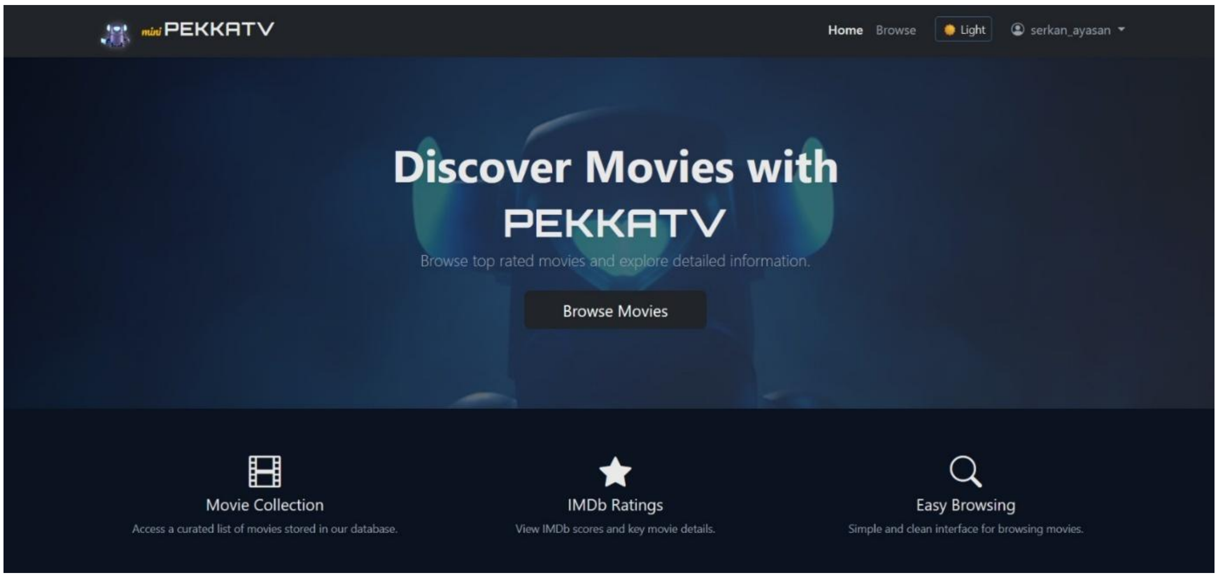
Figure 6.3 Filter Panel

7.Implementation

This section presents the implementation details of the PekkaTV application by demonstrating how the designed functionalities are realized in practice. The implementation is illustrated through key user interface screens that reflect the interaction between the backend logic, database operations, and the user interface.

The Home page serves as the entry point of the application and provides access to movie discovery features. The Browse page allows users to search, filter, and sort movies dynamically based on selected criteria. Movie detail pages integrate movie information, cast details, and user reviews within a single view, enabling seamless interaction with movie-related content. To improve user experience, the application also supports both light and dark display modes, allowing users to switch between themes according to their visual preferences.

Authentication-related functionalities, including user login and registration, are implemented through dedicated pages that control access to user-specific actions. The My Account page consolidates account management features such as password updates, account deletion, and personalized movie lists. The screenshots presented in this section collectively demonstrate the practical realization of the system's functional and architectural design.



Discover Movies with PEKKATV

Browse top rated movies and explore detailed information.

Access a curated list of movies stored in our database.

View IMDb scores and key movie details.

Simple and clean interface for browsing movies.

Figure 7.1 Home page

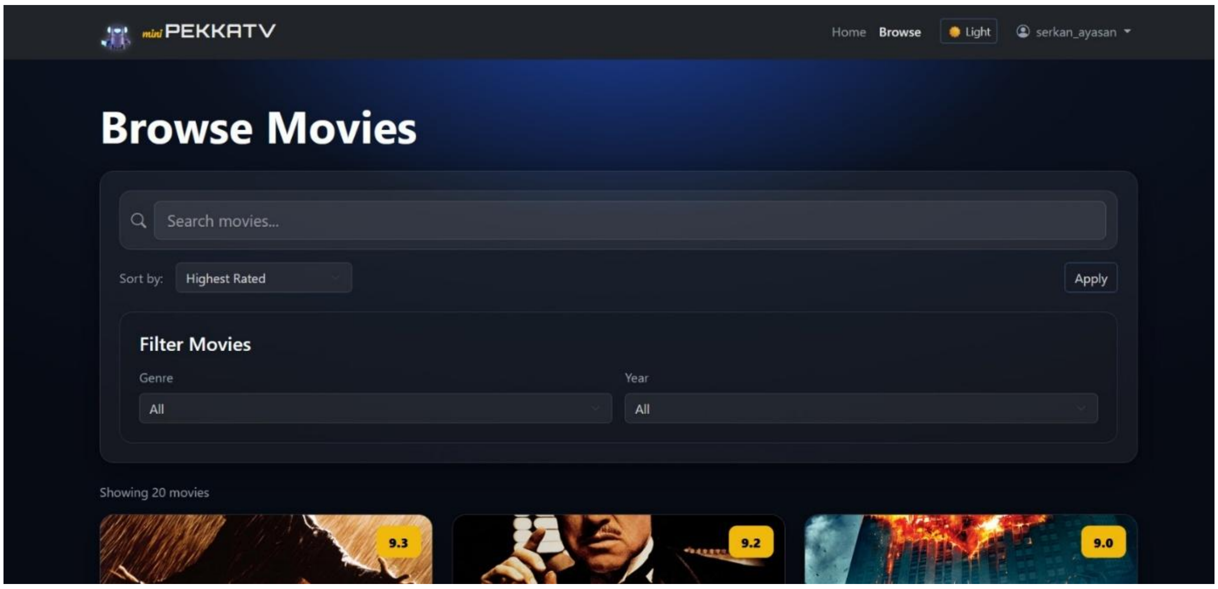


Figure 7.2 Browse Page

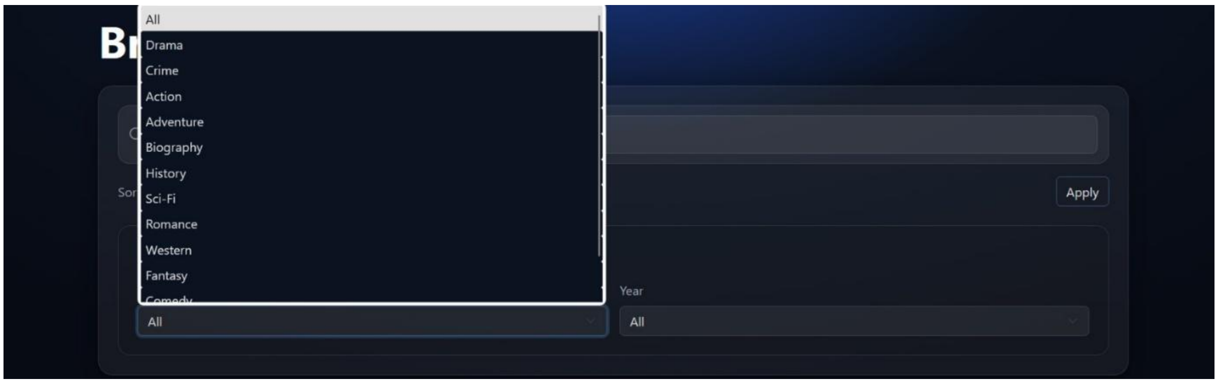


Figure 7.3 Filter movies based on selected genre

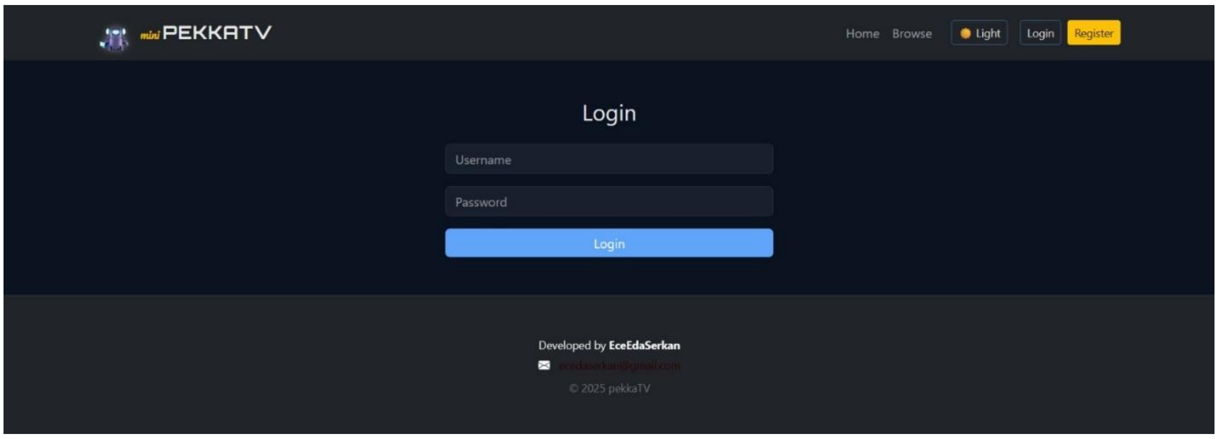
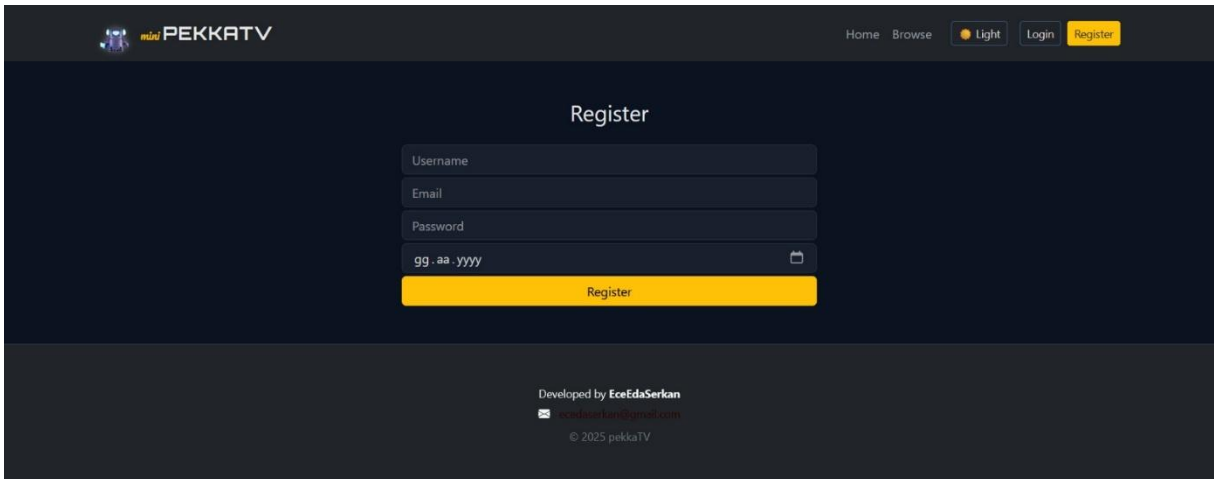
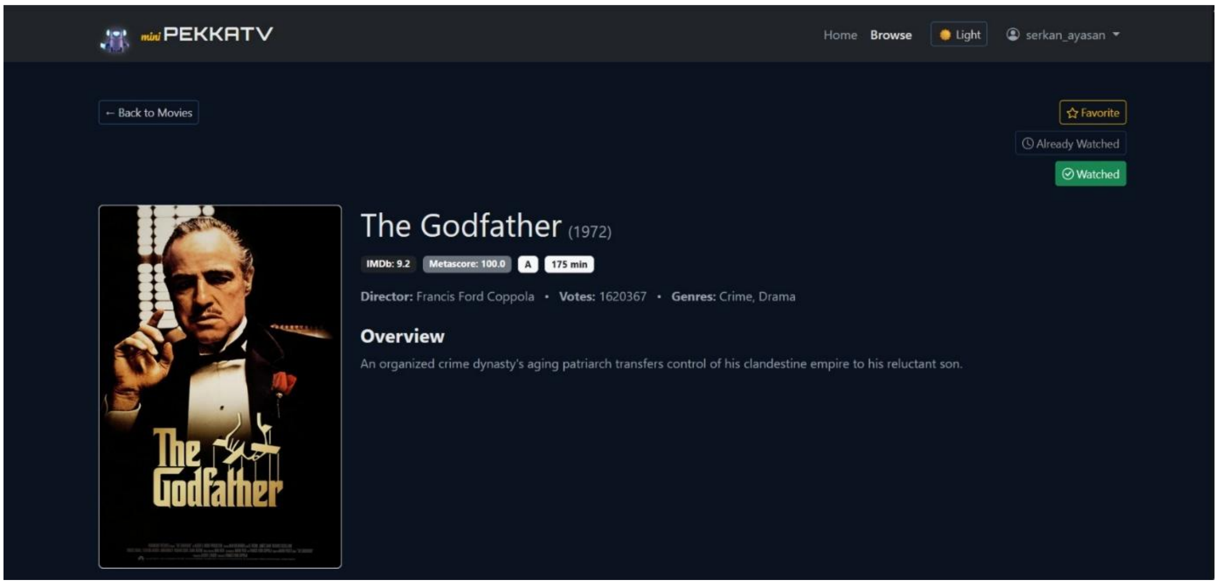


Figure 7.4 Login Page



Developed by **EceEdaSerkan**
ecedaserkan@gmail.com
© 2025 pekkaTV

Figure 7.5 Register Page



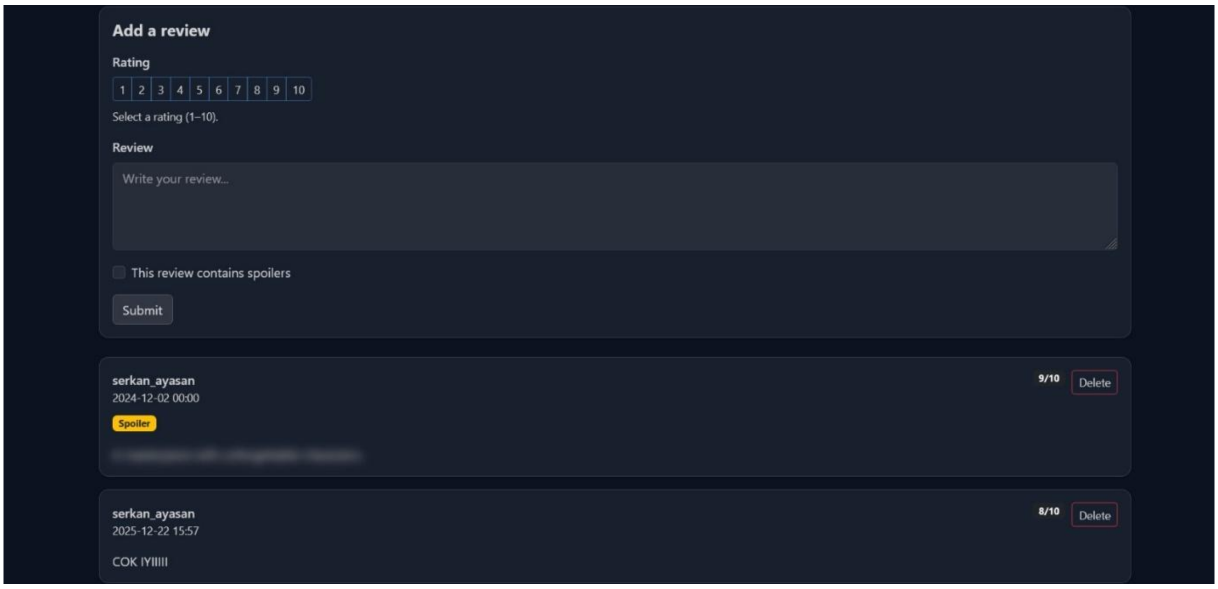
The Godfather (1972)

Director: Francis Ford Coppola • **Votes:** 1620367 • **Genres:** Crime, Drama

Overview

An organized crime dynasty's aging patriarch transfers control of his clandestine empire to his reluctant son.

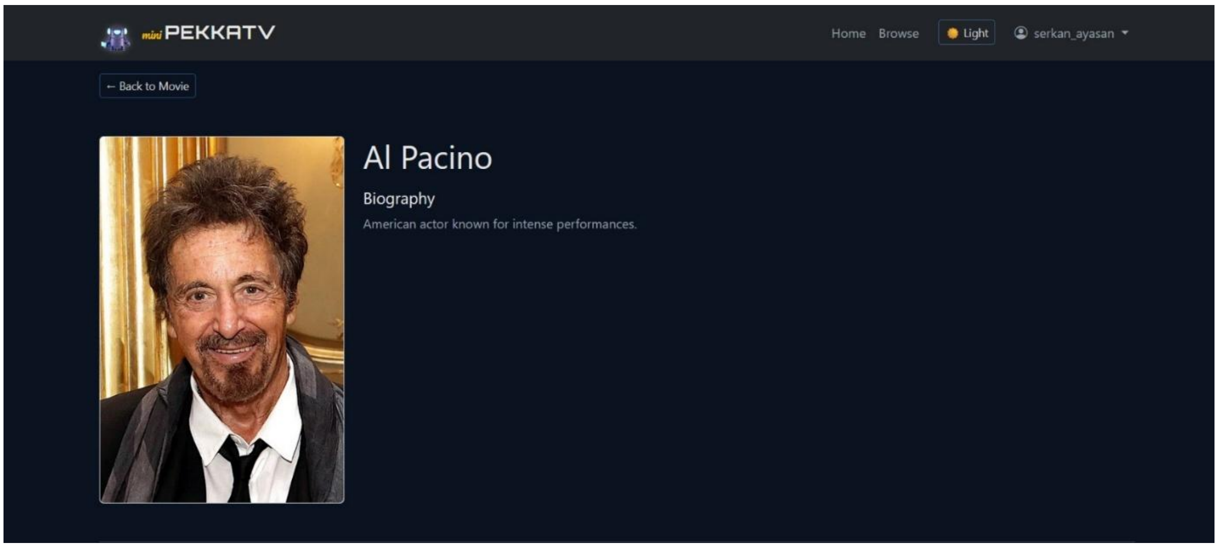
Figure 7.6 Movie's detail page



serkan_ayasan 9/10
2024-12-02 00:00
[illegible]

serkan_ayasan 8/10
2025-12-22 15:57
COK IYIIIII

Figure 7.7 Movie's reviews



Al Pacino

Biography

American actor known for intense performances.

Figure 7.8 Actor's page

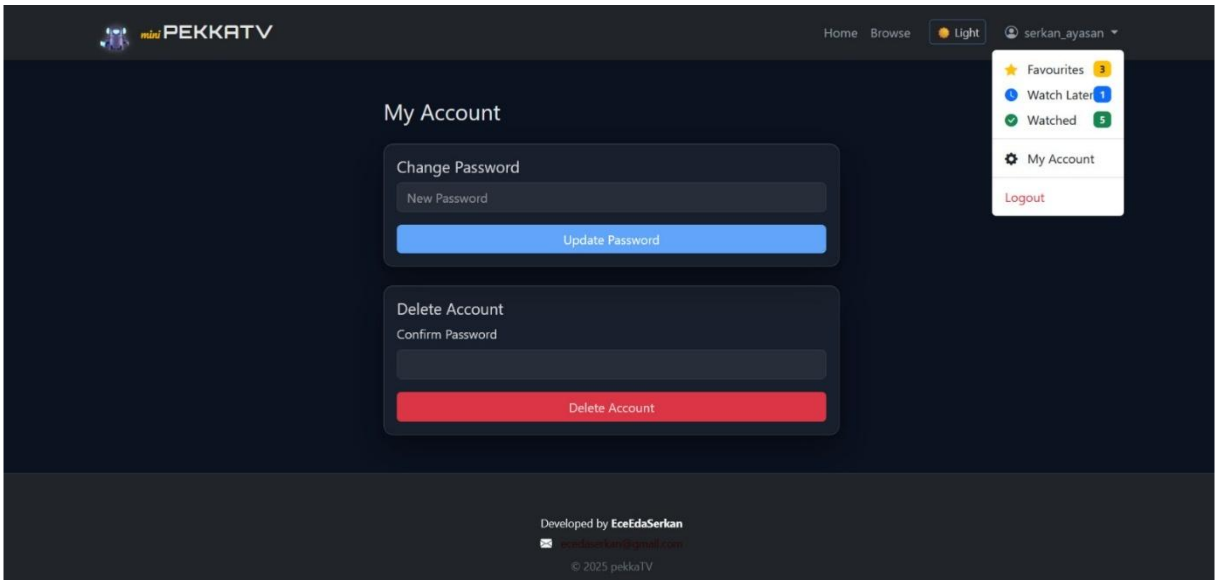


Figure 7.9 My Account Page

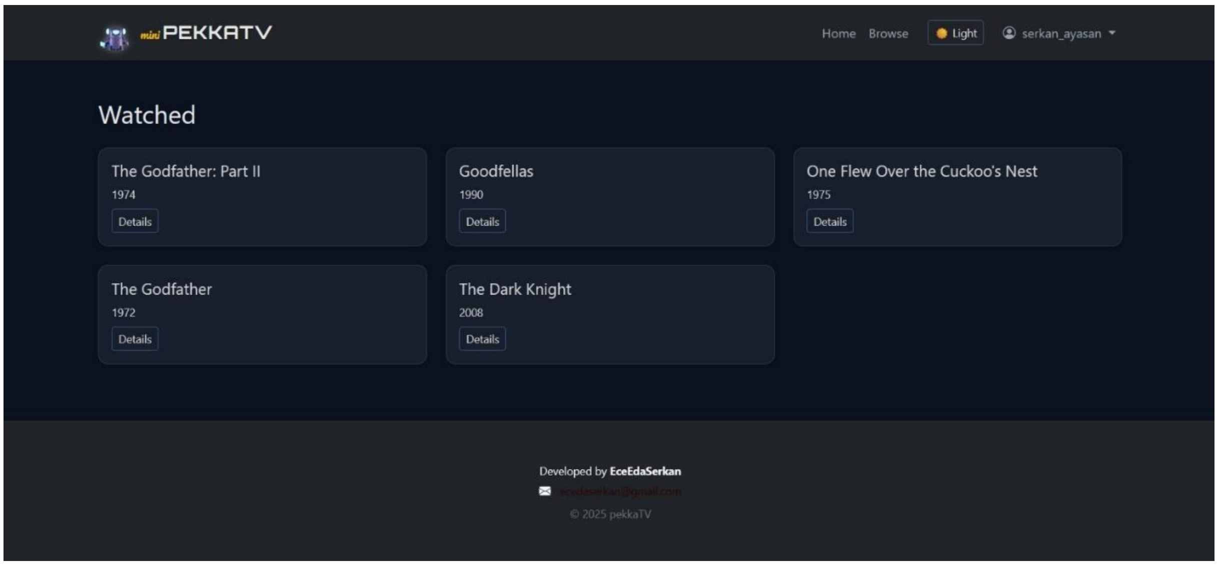


Figure 7.10 User's Watched Page

8. Code Structure and Class Design

This section describes the structural organization of the application code and the design principles adopted for defining classes and their responsibilities. The objective of this structure is to ensure clarity, modularity, and maintainability while supporting the functional requirements of the system. The application is implemented using a Spring Boot–based framework and follows an MVC-based layered organization, where responsibilities are distributed across well-defined components.

8.1 Overall Code Organization

The codebase is organized into logical packages that reflect the main responsibilities of the system. This layered structure separates request handling, data representation, and database access concerns, allowing the application to remain readable and easier to maintain. By grouping related classes together, the system minimizes coupling between components and supports incremental development.

The overall organization aligns with the functional decomposition of the system, ensuring that each functional component is mapped to a corresponding set of classes. This approach simplifies debugging, testing, and future extensions, such as adding new user interaction features or expanding existing functionalities.

8.2 Controller Layer

The Controller layer is responsible for handling incoming HTTP requests and managing the interaction between the user interface and the underlying data model. Controllers process user actions such as browsing movies, viewing movie details, managing account settings, and performing authenticated operations. Each controller coordinates the necessary data retrieval and forwards the results to the appropriate view templates.

The MovieController plays a central role within this layer, as it manages movie-related operations including retrieving movie details, associated reviews, and actor information. By handling these closely related operations within a single controller, the system ensures that movie-specific data is presented cohesively on the movie detail pages. Other controllers handle authentication-related processes and user account management tasks, such as password updates and account deletion, while maintaining access control for restricted actions.

This structure allows request-handling logic to remain centralized and clearly separated from data persistence concerns.

8.3 Model and Data Access Layer

The Model layer represents the core data structures of the application and consists of entity classes that correspond to database tables. These entities encapsulate movie data, user information, reviews, and user-specific lists such as favorites, watched movies, and watchlists. Relationships between entities are defined to maintain data consistency and accurately represent real-world associations within the system.

Database access operations are handled through repository interfaces, which abstract the interaction with the underlying SQL Server database. These repositories provide standardized methods for retrieving, storing, and updating data without exposing database-specific logic to the controller layer. This separation ensures that changes to the database schema or access strategy can be managed with minimal impact on the rest of the application.

8.4 Design Considerations

The adopted code structure emphasizes simplicity and clarity rather than strict enforcement of complex architectural layers. While the system follows an MVC-based approach, certain business logic is intentionally kept close to the controller layer to maintain readability and reduce unnecessary abstraction, which is appropriate for the scope of an academic database management system project.

Overall, the code structure supports modular development and aligns with the system's functional design, providing a solid and extensible foundation for future improvements such as advanced recommendation mechanisms or enhanced analytics features.

9.Future Work

In future work, the movie website can be extended and improved in several ways if more time and resources are available. One possible improvement is the development of a more advanced recommendation system. By analyzing users' favorite movies, watch history, and watch-later lists, the system could suggest personalized movie recommendations, which would enhance the overall user experience.

Another potential extension is improving the filtering and search functionality. More detailed filters such as actor, director, and language could be added. In addition, combining multiple filters simultaneously and enabling sorting options (e.g., by popularity or rating) would allow users to find movies more efficiently.

The comment system can also be further enhanced. For example, users could like or dislike comments, reply to other users' comments, and report inappropriate content. The existing spoiler feature could be improved by allowing users to automatically hide spoiler comments by default and reveal them only if they choose to do so.

User profile management is another area for improvement. Users could be given more control over their profiles, such as editing personal information, viewing detailed statistics about watched movies, and exporting their favorite or watch-later lists. Social features like following other users and viewing their public movie lists could also be added.

Finally, performance and security optimizations can be considered. Implementing caching mechanisms, improving database queries, and strengthening authentication and authorization processes would make the system more scalable and secure. Additionally, developing a mobile-friendly interface or a dedicated mobile application could significantly increase accessibility and usability.