DESIGN DOCUMENT

YEN PHAN – THU TRAN – BOCHENG PENG

Chapter I. Introduction.

The **Media Content Management and Browsing System** is designed to provide a robust platform for managing, browsing, and exploring a variety of media content, such as movies and series. The system is developed using Java, Spring Boot, and PostgreSQL, and aims to offer both a seamless experience for end-users and efficient management tools for administrators. It integrates features such as user authentication, subscription management, content management, and detailed user engagement tracking. This system emphasizes core functionalities including user account management, content browsing and filtering, content access control, history tracking, and admin tools for content moderation and analytics. By incorporating a flexible and user-friendly interface, this system aims to empower both administrators and users, offering personalized content and insights while ensuring efficient content management and access control. This document serves as a comprehensive guide to the system's architecture, design, and functionality. It outlines the technical and functional requirements necessary for development, ensuring that the solution is scalable, secure, and adaptable for future enhancements.

Chapter II. System Design

2.1 System Overview

The **Media Content Management and Browsing System** provides a platform for users to interact with a catalog of media content, including movies and series. Users can manage watch histories, download histories, and profiles, as well as access for enhanced viewing experiences. Subscription management and admin functionalities are also integrated into the system. The backend is developed using **Spring Boot**, with a relational database powered by **PostgreSQL**, and **Java** as the primary programming language. The system integrates external APIs (e.g., TMDB) to enhance the available content information.

2.2 Stakeholders

- **End Users:** Regular users can browse and interact with media content, maintain profiles, track their watch and download histories, and manage their subscriptions.
- **Administrators:** Users with advanced privileges, such as managing media content, overseeing user activity, and accessing insights and logs.
- **System Integrators:** Developers responsible for integrating TMDB API and ensuring data synchronization with external sources.

2.3 Functional Requirements

User and Profile Management

- **User Accounts**: Each user has an account with details such as email, password, subscription plan, and account status.

- **Profiles:** Users can create and manage individual profiles under their account for personalized viewing experiences. Profiles include preferences such as profile image, age, and watch history.

Media Content Management

- **Movies and Episodes:** The system stores details for movies, episodes, and associated metadata (e.g., genres, subtitles, and durations).
- **Genres:** Media content is categorized by genres for filtering.
- **Subtitles:** Users can access subtitle files associated with movies and episodes in different languages.

Subscription Management

- **Subscription Plans:** Users subscribe to predefined plans that include descriptions, subscription fees, and types of plans.
- **Billing and Transactions:** Users can view transaction histories, including subscription payments and billing details.

Watch and Download Histories

- **Watch History:** Tracks user activity, including the date watched, duration, and resume points for movies and episodes.
- **Download History:** Tracks downloaded content, including status and timestamps.

User Interaction Features

- Filter and browse content by genres, ratings, or release dates.
- Access specific episodes or movies.

Admin Management and Activity Tracking

- Admins perform CRUD operations on media content and subscriptions.
- **Activity Logging:** All actions are logged for accountability.

2.4 Non-Functional Requirements

- **Scalability**: The database design supports growth in user base and content, optimized by indexing and normalization.
- **Security**: Uses Spring Security for encrypted passwords, secure authentication, and role-based access control.
- **Performance**: Implements query optimization and caching for frequently accessed data.
- **Reliability**: Logs actions and monitors system health for consistent service.

2.5 System Constraints

- **Data Consistency**: Maintain synchronization between manually added content and data fetched from external APIs like TMDB.
- **API Rate Limits**: Implement caching to reduce repetitive API calls.

- **Database Integrity**: Enforce constraints like foreign keys for reliable relationships between entities.

Chapter III. Architectural Design

The system will utilize a three-layer architecture common for Spring Boot applications. Below is an overview of each component:

3.1 Architectural Components

Frontend (Client Layer)

- Provides user and admin interfaces for managing profiles, subscriptions, and media content.
- Implements RESTful API calls to interact with the backend.

Backend API (Middle Layer)

- Developed in **Spring Boot**, exposing endpoints for user, profile, media, and subscription management.
- Role-based access control secured with **Spring Security**.
- Integrates TMDB API to fetch TV series metadata.

Database (Data Layer)

PostgreSQL serves as the system's persistent storage, handling

- **Account, Profile, and Subscription**: Stores user account details, profiles, and subscription information.
- **TV series and Episodes**: Tracks media content with relationships to genres, and seasons.
- **Movies**: Store the favorite movie data updated by end users. Rows can be added, updated, and deleted.
- **History Tables**: Maintains watch history for user activity tracking.
- **Admin Activity Logs**: Logs all admin actions.

3.2 System Flow

User Registration and Login

- Users register and create accounts. Upon login, they can create profiles under their accounts.
- Secure session management via JWT tokens or session cookies.

Content Management (Admin-Only)

- Admins add or update content using CRUD operations.
- Movies and episodes are enriched using TMDB API metadata, with caching for efficient access.

Subscription and Billing

- Users manage subscription statuses and payments.
- Subscription fees and expiration dates are tracked per account.

Interaction with Media Content

- Users interact with content by adding it to their watch history or updating their watch statuses.
- contents are dynamically retrieved and displayed based on user preferences.

Admin Activity Tracking

- All admin actions, such as content updates or user management, are logged in the **Admin_Activity_Log** table.

Search and Filter

- Users can search for movies and episodes by title, genre, and rating.
- Filters are available to refine results by additional criteria.

3.3 Technical Stack

Backend:

- **Java 17+:** Programming language for the backend.
- **Spring Boot:** Framework for developing RESTful APIs.
- **Spring Security:** For role-based access control.

Spring Boot Starter Dependencies:

- **spring-boot-starter-data-jpa:** This dependency provides support for JPA (Java Persistence API), which is a standard for object-relational mapping (ORM). It allows you to interact with a relational database using Java objects.
- **spring-boot-starter-data-rest:** This dependency enables Spring Data REST, which allows you to expose your JPA entities as a REST API automatically.
- **spring-boot-starter-security:** This dependency integrates Spring Security into your application, providing features like authentication, authorization, and user management.
- **spring-boot-starter-thymeleaf:** This dependency brings in Thymeleaf, a template engine for creating dynamic web pages.
- **spring-boot-starter-web:** This dependency includes the basic building blocks for creating a web application, such as Tomcat or Jetty as the embedded web container.
- **spring-boot-starter-web-services:** This dependency adds support for building web services with Spring Web Services (WS).

- **spring-boot-starter-actuator:** This dependency provides actuator endpoints for monitoring and managing your Spring Boot application.
- **spring-boot-starter-mail:** This dependency allows you to send emails from your application.

Other Dependencies:

- **modelmapper:** This dependency provides a library called ModelMapper for simplifying mapping between different object types.
- **jackson-datatype-jsr310:** This dependency adds support for Java 8 date and time API (JSR-310) types for serialization and deserialization with Jackson.
- **thymeleaf-extras-springsecurity6**: This dependency extends Thymeleaf with Spring Security features, making it easier to secure your Thymeleaf templates.
- **h2 (or postgresql):** This dependency includes a database driver for either H2 (in-memory database) or PostgreSQL. You can choose one based on your preference.
- lombok: This dependency (annotation processor) simplifies your code by generating boilerplate code like getters, setters, and toString methods at compile time.
- **jakarta.persistence-api:** This dependency provides the Java Persistence API for JPA interactions.
- **spring-boot-starter-test and spring-security-test:** These dependencies provide libraries for testing your Spring Boot application and Spring Security components.
- **jjwt-api, jjwt-impl, and jjwt-jackson:** These dependencies are from the JSON Web Token (JWT) library, used for creating and parsing JWTs for authentication purposes.
- **jackson-dataformat-xml:** This dependency adds Jackson support for working with XML data formats.

Database:

- **PostgreSQL**: Relational database for all structured data.

Frontend:

- **Thymeleaf:** Java-based template engine that allows you to create dynamic and maintainable web templates

Chapter IV. Entities Relationship Diagram (ERD).

The **Entity-Relationship Diagram (ERD)** represents the relationships between the different entities in the system, highlighting how the data is structured and how entities interact with each other.

4.1. ERD.

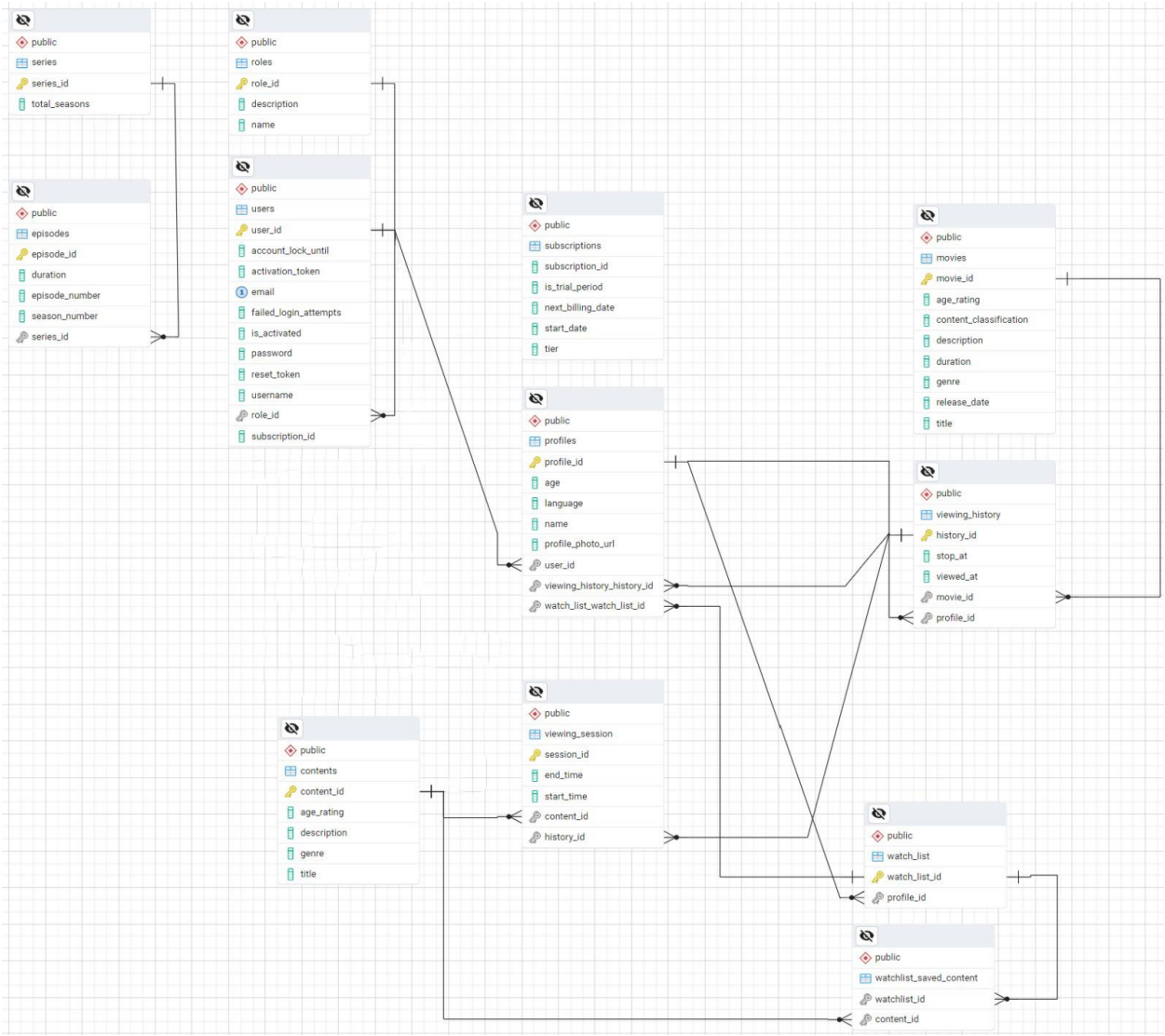

Figure 1: Entity Relationship Diagram

4.2. Detailed Explanation of the ERD.

The **Entity-Relationship Diagram (ERD)** represents the relationships between the different entities in the system, highlighting how the data is structured and how entities interact with each other.

Entities and Their Attributes:

- **Series**: Tracks information about series.
    - Attributes: series_id, title, total_seasons.
- **Episodes**: Represents individual episodes of a series.
    - Attributes: episode_id, duration, episode_number, season_number, series_id (foreign key).
- **Movies**: Represents movies available on the platform.

    - Attributes: movie_id, title, genre, description, duration, release_date, content_classification, age_rating.
- **Users**: Contains user-related data.
    - Attributes: user_id, email, username, password, reset_token, is_activated, failed_login_attempts, account_lock_until, role_id (foreign key), subscription_id (foreign key).
- **Roles**: Represents user roles (e.g., admin, viewer).
    - Attributes: role_id, name, description.
- **Subscriptions**: Tracks user subscription details.
    - Attributes: subscription_id, start_date, next_billing_date, trial_period, tier.
- **Profiles**: Represents user profiles under a single account.
    - Attributes: profile_id, user_id (foreign key), name, age, language, profile_photo_url, viewing_history_history_id, watch_list_watch_list_id (both foreign keys).
- **Viewing History**: Tracks what content profiles have viewed.
    - Attributes: history_id, profile_id (foreign key), movie_id (foreign key), viewed_at, stop_at.
- **Viewing Session**: Tracks session details during content viewing.
    - Attributes: session_id, history_id (foreign key), content_id (foreign key), start_time, end_time.
- **Watch List**: Stores content saved by users to watch later.
    - Attributes: watch_list_id, profile_id (foreign key).
- **Watch List Saved Content**: Links saved content to the watch list.
    - Attributes: watch_list_id (foreign key), content_id (foreign key).
- **Contents**: A generalized entity for all content (series, movies, etc.).
    - Attributes: content_id, title, description, genre, age_rating.

Relationships:

- **Users** to **Roles**: users.role_id references roles.role_id (many-to-one).
- **Users** to **Subscriptions**: users.subscription_id references subscriptions.subscription_id (many-to-one).
- **Users** to **Profiles**: A user can have multiple profiles (users.user_id to profiles.user_id).
- **Profiles** to **Viewing History**: Each profile has a viewing history (profiles.profile_id to viewing_history.profile_id).
- **Profiles** to **Watch List**: A profile has a watch list (profiles.profile_id to watch_list.profile_id).
- **Viewing History** to **Movies**: Links watched movies (viewing_history.movie_id to movies.movie_id).
- **Viewing Session** to **Viewing History**: Tracks detailed viewing sessions (viewing_session.history_id to viewing_history.history_id).

- **Viewing Session** to **Contents**: Tracks sessions of viewed content (viewing_session.content_id to contents.content_id).
- **Watch List** to **Watch List Saved Content**: Connects watch lists to saved content (watch_list.watch_list_id to watch_list_saved_content.watch_list_id).
- **Watch List Saved Content** to **Contents**: Links saved content in a watch list to available content (watch_list_saved_content.content_id to contents.content_id).
- **Episodes** to **Series**: Links episodes to their series (episodes.series_id to series.series_id).

Chapter V. Class Diagram.

Figure 2: Class Diagram

Main Configuration of Classes:

- **AppConfig:** This class is responsible for configuring the main application context. It might define beans for various components, properties, and other application-wide settings.
- **SecurityConfig:** This class handles the project's security configuration. It would define how users are authenticated and authorized, such as setting up authentication providers, defining roles and permissions, and configuring security filters.
- **TMDBConfig:** This class is specific to the project, related to integrating with the TMDb (The Movie Database) API. It contains configuration details for interacting with the API, such as API keys, endpoints, or data mapping.
- **WebConfig:** This class handles web-related configurations. It configures things like message converters for handling request and response bodies, setting up view resolvers for rendering templates, or configuring static resources.

Controller Classes:

- **AuthController:** This controller is responsible for handling authentication-related operations, such as user login, registration, password reset, and possibly token-based authentication.
- **ContentController:** This controller is a general-purpose controller for managing content, depending on the specific content type the application handles.
- **EmailController:** This controller handles sending emails, such as welcome emails, password reset emails, or notifications.
- **EpisodeController:** This controller manages episodes, likely for a TV show or series application. It might handle operations like fetching episode details, listing episodes, or updating episode information.
- **JWTGenerator:** This class is responsible for generating JSON Web Tokens (JWTs) for authentication and authorization purposes.
- **MovieController:** This controller manages movies, fetching movie details, listing movies, and handling movie-related operations.
- **MovieFetchController:** This controller is a specific controller for fetching movie data, possibly from an external API like TMDb.
- **MovieFetchRestController:** This controller is a RESTful version of the MovieFetchController, handling HTTP requests and responses for movie data retrieval.
- **ProfileController:** This controller handles user profiles, allowing users to view, edit, or update their profile information.
- **RoleController:** This controller manages user roles and permissions, possibly assigning roles to users or managing role-based access control.
- **SeriesController:** This controller manages TV series, similar to the MovieController but for series. It handles fetching series details, listing series, and managing series-related operations.
- **SeriesPopularController:** This controller specifically handles popular series, possibly fetching a list of popular series or filtering series by popularity.
- **SeriesPopularRestController:** This controller is a RESTful version of the SeriesPopularController, handling HTTP requests and responses for popular series.

- **SeriesRatedController:** This controller handles series ratings, allowing users to rate series or fetching series with specific ratings.
- **SeriesRatedRestController:** This controller is a RESTful version of the SeriesRatedController, handling HTTP requests and responses for series ratings.
- **SubscriptionController:** This controller manages user subscriptions, such as subscriptions to premium content or notifications.
- **UserController:** This controller manages user accounts, handling user creation, deletion, and updates.
- **ViewingHistoryController:** This controller manages user viewing history, tracking the movies or series that a user has watched and possibly providing recommendations based on viewing history.

Classes under 'dto' package:

- **AuthResponseDTO:** This class represents the data structure returned by the authentication process. It might contain information like user ID, token, and other relevant authentication details.
- **ContentDTO:** This class could represent a generic content object, potentially holding common properties like title, description, release date, etc., that can be used across different content types (movies, series, episodes).
- **EpisodeDTO:** This class would specifically represent an episode, containing information like episode number, title, duration, and possibly a link to the corresponding series.
- **GenreDTO:** This class represents a genre, holding information about the genre name or ID.
- **LoginDTO:** This class would represent the data structure for user login requests, containing fields like username/email and password.
- **LoginRequest:** This class represents a login request, potentially with additional fields or validation rules.
- **MovieDTO:** This class represents a movie, holding information like title, release date, genre, cast, crew, plot summary, etc.
- **ProfileDTO:** This class represents user profile information, including details like name, bio, preferences, and possibly a profile picture.
- **RegisterDTO:** This class represents the data structure for user registration requests, containing fields like username/email, password, and possibly other required information.
- **ReleaseDateDTO:** This class represents the release date of a movie or series, possibly with different formats or time zones.
- **ReleaseDateResponseDTO:** This class represents the response structure for release date information, potentially including multiple release dates or additional details.
- **ReleaseDateResultDTO:** This class represents a specific result or outcome related to release date information.
- **ResetPasswordRequest:** This class represents the data structure for password reset requests, containing information like username/email or a reset token.
- **RoleDTO:** This class represents user roles, holding information about role names or IDs.
- **SeriesDTO:** This class represents a TV series, containing information like title, release date, number of seasons, genre, cast, crew, etc.
- **SubscriptionDTO:** This class represents user subscriptions, holding information about the subscribed content, subscription plan, and payment details.

- **SubscriptionResponse:** This class represents the response structure for subscription operations, such as successful subscription or error messages.
- **TMDBMovieDTO:** This class is specific to application, potentially representing movie data fetched from the TMDb API, with specific fields or data structures required for the application.
- **UserDTO:** This class represents user information, holding details like username/email, password, roles, profile information, and other user-related data.
- **ViewingHistoryDTO:** This class represents a user's viewing history, tracking the movies or series they have watched, along with timestamps or other relevant information.
- **WatchListDTO:** This class represents a user's watchlist, containing a list of movies or series that the user intends to watch.

Classes under 'exception' package:

- **ProfileLimitExceededException:** This exception is thrown when a user attempts to create more profiles than allowed.
- **ProfileNotFoundException:** This exception is thrown when a requested profile is not found.
- **ResourceAlreadyExistsException:** This exception is thrown when an attempt is made to create a resource that already exists (e.g., a user with the same username).
- **SubscriptionNotFoundException:** This exception is thrown when a requested subscription is not found.
- **TMDBApiException:** This exception is thrown when an error occurs while interacting with the TMDb API.
- **UserNotFoundException:** This exception is thrown when a requested user is not found.
- **ViewingHistoryNotFoundException:** This exception is thrown when a requested viewing history is not found.

Classes under 'models' package:

- **AgeRating:** This class represents age ratings for content, such as "G", "PG", "PG-13", "R", etc.
- **Content:** This class could be a base class or interface for different types of content (movies, series, episodes), holding common properties like title, description, release date, etc.
- **ContentClassification:** This class represents content classifications, such as "TV-MA", "PG-13", "G", etc.
- **Language:** This class represents languages, holding information like language code (e.g., "en", "es", "fr") or language name.
- **Genre:** This class represents genres, holding information about genre names or IDs.
- **Movie:** This class represents a movie, holding information like title, release date, genre, cast, crew, plot summary, etc.
- **PopularSeries:** This class represent popular series, potentially with additional fields or properties related to popularity.
- **Profile:** This class represents user profile information, including details like name, bio, preferences, and possibly a profile picture.
- **RatedSeries:** This class represents a series that a user has rated, potentially with additional fields like the user's rating.

- **RatedSeriesList:** This class represent a list of rated series for a user.
- **Role:** This class represents user roles, holding information about role names or IDs.
- **Series:** This class represents a TV series, containing information like title, release date, number of seasons, genre, cast, crew, etc.
- **SubscriptionTier:** This class represents different subscription tiers or plans, holding information about pricing, features, and content access.
- **Subscription:** This class represents user subscriptions, holding information about the subscribed content, subscription plan, and payment details.
- **User:** This class represents user information, holding details like username/email, password, roles, profile information, and other user-related data.
- **ViewingHistory:** This class represents a user's viewing history, tracking the movies or series they have watched, along with timestamps or other relevant information.
- **ViewingSession:** This class represents a single viewing session for a user, tracking the start and end time of a viewing session for a specific content item.
- **WatchList:** This class represents a user's watchlist, containing a list of movies or series that the user intends to watch.

Classes under 'repository' package:

- **ContentRepository:** This interface defines methods for interacting with the data storage for content objects.
- **EpisodeRepository:** This interface defines methods for interacting with the data storage for episodes.
- **GenreRepository:** This interface defines methods for interacting with the data storage for genres.
- **MovieFetchRepository:** This interface might be specific to fetching movie data, potentially from an external API like TMDb.
- **MovieRepository:** This interface defines methods for interacting with the data storage for movies.
- **ProfileRepository:** This interface defines methods for interacting with the data storage for user profiles.
- **RoleRepository:** This interface defines methods for interacting with the data storage for user roles.
- **SeriesRepository:** This interface defines methods for interacting with the data storage for TV series.
- **SubscriptionRepository:** This interface defines methods for interacting with the data storage for user subscriptions.
- **UserRepository:** This interface defines methods for interacting with the data storage for users.
- **ViewingHistoryRepository:** This interface defines methods for interacting with the data storage for user viewing history.
- **WatchListRepository:** This interface defines methods for interacting with the data storage for user watchlists.

Classes under 'response' package:

- **PopularResponse:** This class represents the response structure for popular content, potentially containing a list of popular movies or series.

- **RatedResponse:** This class represents the response structure for rated content, potentially containing a list of rated movies or series.

Classes under 'services' package:

- **AuthService:** This interface likely defines methods for handling authentication operations, such as user login, registration, password reset, and token management.
- **ContentService:** This interface might define methods for managing content, such as fetching, creating, updating, and deleting content objects.
- **EmailService:** This interface defines methods for sending emails, such as welcome emails, password reset emails, or notifications.
- **EmailSender:** This class is an implementation of the EmailService interface, responsible for actually sending emails using a specific email library or service.
- **MovieFetchService:** This service is likely responsible for fetching movie data, potentially from an external API like TMDb.
- **MovieService:** This interface defines methods for managing movies, such as fetching movie details, listing movies, and handling movie-related operations.
- **ProfileService:** This interface defines methods for managing user profiles, allowing users to view, edit, or update their profile information.
- **SeriesPopularService:** This service is likely responsible for handling popular series, potentially fetching a list of popular series or filtering series by popularity.
- **SeriesRatedService:** This service might handle series ratings, allowing users to rate series or fetching series with specific ratings.
- **SubscriptionService:** This interface defines methods for managing user subscriptions, such as subscribing to premium content or notifications.
- **TMDBService:** This interface might define methods for interacting with the TMDb API, such as fetching movie and TV show data.
- **UserService:** This interface defines methods for managing user accounts, handling user creation, deletion, and updates.
- **ViewingHistoryService:** This interface defines methods for managing user viewing history, tracking the movies or series that a user has watched and possibly providing recommendations based on viewing history.
- **WatchListService:** This interface defines methods for managing user watchlists, allowing users to add or remove movies or series from their watchlist.

Chapter VI. Activity Diagram.

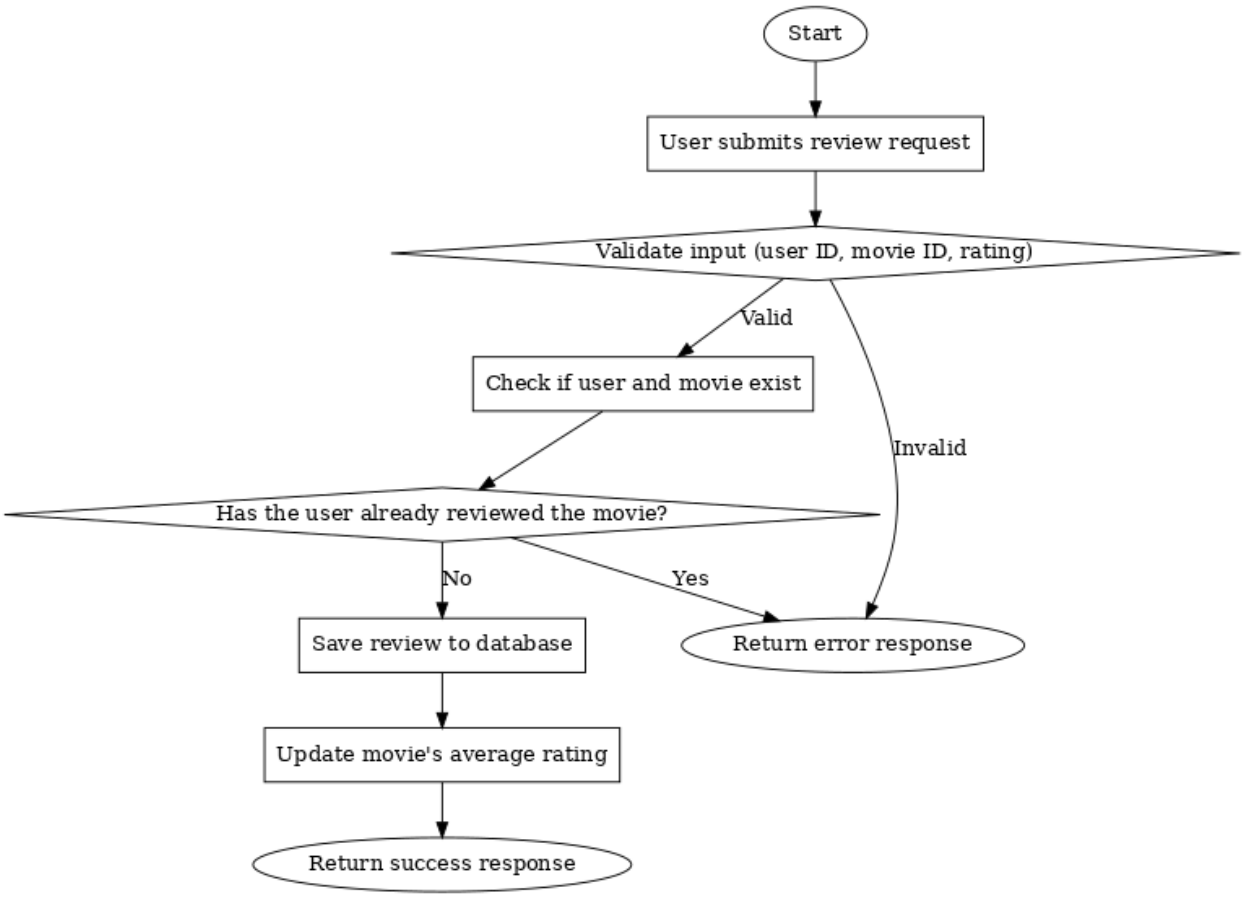

Figure 3: Activity Diagram – user adds review

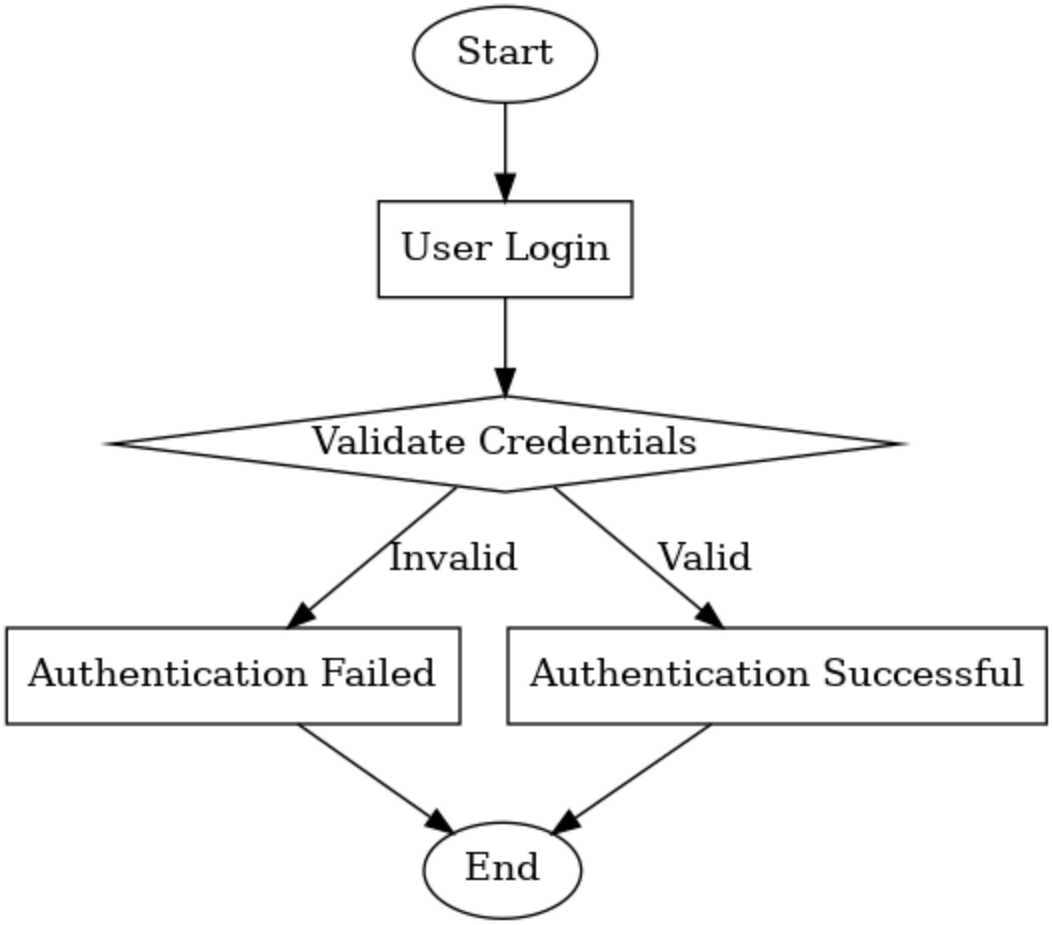

Figure 4: Activity Diagram – user logs in

Chapter VI. Endpoints.

Endpoints are specific URLs within an API that allow different functions of a system to be accessed by external applications or clients. Each endpoint represents a unique path within the API that maps to a specific operation, such as retrieving, creating, updating, or deleting data. They enable interaction with the server by specifying the HTTP method (e.g., GET, POST) and defining the data input and output formats.

This API section will be written using the standards of the OpenAPI, which is an open standard for describing REST APIs. It provides a structured and language-agnostic way to define endpoints, request and response structures, and other key aspects of an API. The OpenAPI Specification (OAS) standardizes API documentation, making it easier for developers to understand and integrate with the API. By adhering to the OpenAPI 3.1.0 standard, we ensure that our API is well-documented, consistent, and accessible, enabling seamless integration and ease of use.

Appendix.

Version	Date	Author	Description
0.1	17.11.2024	Thu Tran, Bocheng Peng, Yen Phan	Initial Version