CS349 Project - WatchGuru

Final Report

Bhavya Sri Mamidi (22B0918)
Jasmitha Jajala (22B0997)
Jalajakshi Palli (22B1047)
Mahak Sahu (22B1051)

May 2025

GitHub Repository

1 Project Goals

The primary goals of **WatchGuru** are:

- **Simplify Content Discovery**
  To help users easily discover movies, dramas, and anime that match their tastes through personalized recommendations, reducing time spent searching.
- **Build a Community-Centric Platform**
  To create a social space where users can connect with others who share similar entertainment interests through reviews, discussions, and friend-based interactions.
- **Enable Personalized Ratings and Reviews**
  To allow users to rate and review content publicly, among friends, or anonymously—enabling multiple layers of opinion-sharing.
- **Provide Blended and Collaborative Recommendations**
  To generate a single movie recommendation by combining a user's preferences with those of their friends, ensuring an easy and conflict-free shared viewing experience.
- **Implement Advanced Search and Filtering**
  To allow users to explore content via genre, languages, cast, or by searching directly using the name, enhancing user control and experience.
- **Encourage User Engagement through Forums and Chats**
  To integrate community engagement features like forums and private chats, increasing retention and daily interaction.

2 Implementations from User Perspective

- **User Registration and Login**
  - Fields for Creating Account: Usename, Email, Password, Date of Birth
  - After clicking sign up, need to select atleast 3 favourite genres among the provided and then sign up will complete
  - Ensuring different usernames and different emails
  - Fields for Signing in: Email, Password
- **Dashboard**
  - **Personalized Recommendations**: Generated based on users' favorite genres to tailor content to individual preferences.
  - **Item-Based Collaborative Filtering**: Recommends content by identifying items with similar user rating patterns — based on the concept that *"people who liked this item also liked these similar items."*
  - **User-Based Collaborative Filtering**: Suggests content based on similarity between users' rating behaviors — following the idea that *"users like you also liked these items."*

- **Popular Content Recommendations**: Displays the top 10 most popular items across the platform.
- **Friend-Based Recommendations**: Shows the top 10 items highly rated by the user's friends.
- **Filtering Options**: Users can filter content based on genre and language to better match their interests.
- **User Interface Elements**:
  - A **navigation bar** featuring a search bar, notification icon, and profile icon.
  - A **search functionality** allowing users to find content by name or cast members.
  - A **notification panel** that informs users about:
    - Incoming friend requests
    - Accepted friend requests
    - When received a message from a friend
    - Content recommended by friends
  - Clicking on a notification for a friend request redirects the user to the **Friends** page, and the notification is marked as read.
  - Clicking on a notification for a new message from a friend opens the respective chat, and the notification is marked as read.
  - Clicking on a notification for content recommended by a friend takes the user to the **Content Information** page for that particular recommendation, and the notification is marked as read.

- **Profile**
  - **Navigation Bar**: A top navigation bar displays the user's profile name along with a notification button.
  - **Main Interface Buttons**: The interface includes four primary buttons — **Friends**, **Chats**, **Reviews**, and **Profile** (default selected).
  - **Friends Section**: On clicking the **Friends** button, users are redirected to a page displaying:
    - Existing friends
    - Incoming and outgoing friend requests
    - Friend suggestions
  - **Chats Section**: The **Chats** button opens a chat page where users can initiate conversations with friends by selecting their names from a list.
  - **Reviews Section**: The **Reviews** page displays all the reviews submitted by the user for various content.
  - Below the profile section, users can view their personal information, including their bio and selected favourite genres, which were initially provided during the signup process.
  - This information can be updated by clicking the edit icon, allowing users to modify their bio and change their favourite genres as desired.

- **Friend System**
  - **Navigation Bar**: Contains a profile icon and a notification button.
  - **User Search Bar**: Enables users to search for other users by name. Each search result displays either an option to send a friend request or a "Friends" label if the users are already connected.
  - **Friend Management Interface**: The friends section is organized into three columns:
    - **Friends**: Displays a list of users who are already added as friends.
    - **Requests**: Shows incoming friend requests with *Accept* and *Reject* options, as well as outgoing requests with a *Cancel* option.
    - **Suggestions**: Recommends users who are either:
      - Friends of existing friends, or
      - Users who share at least two common favorite genres.
  - **Profile Navigation**: Clicking on another user's icon redirects to their public profile page, allowing users to view their shared information.
- **Content Browsing and Information**
  - Clicking a particular moviecard opens a dedicated content page displaying detailed information about that movie or show.
  - The content page includes:
    - Title and genres of the content.
    - A brief description or synopsis.
    - Additional metadata such as release date, director, duration, language, age rating, and average rating.
    - Two action buttons: **Share with Friends** and **Give a Rating**.
    - **Share with Friends** functionality:
      - Opens a dialog box titled *Select Friends to Notify*, displaying a list of the user's friends.
      - Options to *Select All*, *Deselect All*, or manually select specific friends.
      - Upon clicking the *Notify* button, selected friends receive a notification about the recommendation.
    - **Give a Rating** functionality:
      - Displays a 10-star rating scale for the user to select their rating.
      - Includes a review box for users to write comments, which remains editable for future updates.
    - A section displaying reviews from other users for the selected content.
    - Information about the cast and the streaming platforms where the content is available.
    - A link to watch the trailer.
    - A list of similar recommended movies or shows displayed at the bottom of the page.
- **Rating and Reviews**

  - This refers to the **Give a Rating** feature described above in the Content Information section.

- **Favourites, Watchlists and Watched**
  - Each movie/show card displays the title, associated genres, and three action icons allowing the user to:
    * Like the content
    * Save the content for later
    * Mark the content as watched
  - Content marked as liked will appear under the **Favourites** page.
  - Content that is saved will be listed under the **My List** page.
  - Content marked as watched will be displayed on the **Watched** page.

- **Discussion Forums**
  - The discussion forum is dedicated to movie recommendations.
  - Clicking on **Ask Question** prompts the user to provide a Question Title, a detailed description, and select relevant tags from a predefined list.
  - Upon clicking **Post Question**, a bot generates a list of recommended content based on the provided question.
  - A search bar is available to search for questions.
  - Users can view all questions posted by other users in the forum.
  - Any user can respond to a question, which will be shown in the form with their profile names.
  - Can upvote or downvote a response.

- **Group Recommendation**
  - The **Group Recommendations** page allows users to choose from three categories: **Movies**, **Anime**, or **TV Shows**.
  - It features a **Select Your Squad** section where users can choose friends for a movie night.
  - By clicking the **dice button**, the app recommends a title based on the shared interests of the selected squad.

- **Personalized Recommendations**
  - Users receive personalized movie and show recommendations, tailored to their preferences and viewing history. These recommendations are displayed on the dashboard.

- **Search and Filters**
  - The dashboard provides a search bar to find content by name, along with filter options to refine results based on language and genre.

- **Personal Chats**

  - The Chats page displays a list of all friends, each with an option to initiate a conversation.
  - Clicking on the chat icon beside a friend's name opens a chat window where users can exchange messages.

- **Notifications**
  - The notifications will be sent when a user sends a friend requests, when a user accepts a friend request, when a user sends a message and when a user recommends movie to their friends.
  - The **Navbar** includes a notification button, accessible from all pages, with its functionality and description outlined in the **Dashboard** section.

- **Shows Movies Anime**
  - These include three separate pages: the **Shows** page displaying only TV shows, the **Movies** page listing all movies, and the **Animes** page showcasing anime content. Each page presents filtered content specific to its category.

3 Architecture of WatchGuru

WatchGuru is designed as a modular full-stack web application with a clear separation between the frontend, backend, and database layers. It follows a scalable and maintainable architecture, allowing collaborative development and easy addition of features.

- **Frontend (Client-Side)**
  - **Technology:** React.js
  - **Responsiblites:**
    * User Interface and Navigation
    * Content display and search
    * Watchlist management, rating forms
    * Interacting with backend via REST API

  Deployed at: `http://localhost:3000/`

- **Backend (Server-Side API)**
  - **Technology:** Node.js + Express.js
  - **Responsiblites:**
    * Authentication (Login, Signup)
    * RESTful APIs for user, content, friends, ratings, etc.
    * Data validation and access control
    * Business logic (recommendations, filtering)
    * AI Model for personalized recommendations

  Runs on: `http://localhost:5000/`

- **Database (PostgreSQL)**
  - **Technology:** PostgreSQL
  - **Tables:**
    - `Users` - stores user details and profile information
    - `Content` - stores movies/shows/anime details
    - `Ratings` - user ratings and reviews
    - `Friends` - friend requests and relationships
    - `Messages` - Stores chats with friends
    - `Reviews` - Stores reviews and ratings
    - `Notifications` - Stores user notifciations
    - Tables for Recommender ML model - `collab_recommendations`, `svd_user_factors`, `svd_item_factors`, `movie_similarity`, `user_similarity`
    - Tables for User Lists - `likes`, `watchlist`, `watch_history`
    - Tables for Discussion Forum logic - `questions`, `answers`, `votes`

4 FrontEnd Functionality

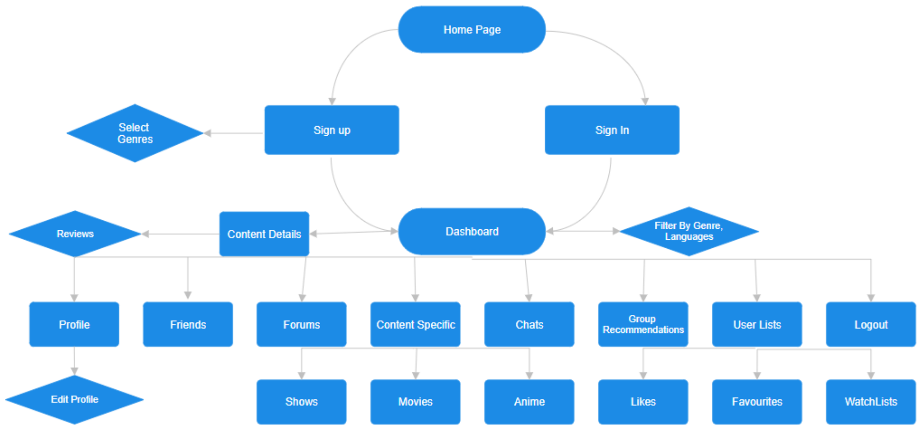

Figure 1: Frontend Design

5 BackEnd Functionality

5.1 User Authentication & Profile Management

- Implemented secure registration and login routes using bcrypt.
- Protected routes using middleware (`isAuthenticated`, `isLoggedIn`) that checks for session management.
- Enabled profile data retrieval (bio, genres, profile image).
- Supported profile updates through PUT routes.

5.2 Friend System & Social Connectivity

- Enabled sending, accepting, and removing friend requests.
- Maintained friend relationships in a relational table.
- Provided APIs to fetch friend lists and friend requests.

5.3 Content Browsing & Metadata Storage

- Provided APIs fetch content data (movies, shows, anime) from the `Content` table.
- Stored content details like title, genre, description, and poster URL in the `Content` table.
- Provided filterable and searchable content endpoints.

5.4 Watchlists, Favourites & Watched History

- Allowed users to add content to watchlists, favourites, and mark as watched.
- Created association tables to track user-specific lists.
- Exposed endpoints to fetch and update user-specific content states.

5.5 Ratings & Reviews System

- Enabled users to rate content and post reviews.
- Supported anonymous and friend-only review visibility.
- Calculated and cached average ratings per content item.

5.6 Personalized Recommendation Integration

Our system integrates personalized recommendations using a combination of collaborative filtering and content-based filtering methods, backed by a structured database and periodic updates. The key aspects are:

1. **User Reviews Storage:** All user ratings are stored in the `reviews` table, which records `user_id`, `content_id`, and `rating`. This forms the user-item matrix for collaborative filtering.
2. **Model Training and Similarity Calculation:** We employ three major collaborative filtering techniques:
   - *SVD Filtering:* Reduces the dimensionality of the user-item matrix and stores factors in `svd_user_factors` and `svd_item_factors`.
   - *Item-Based Collaborative Filtering:* Computes cosine similarity between items, storing pairs in the `movie_similarity` table.
   - *User-Based Collaborative Filtering:* Computes cosine similarity between users, storing pairs in the `user_similarity` table.

3. **Batch Processing and Freshness:** To manage large datasets, training is performed in batches. Before new data is inserted, existing tables are truncated to keep only the latest computations.

4. **Serving Recommendations:** When a user fetches recommendations, six strategies are run in parallel:

   - Content-based recommendations,
   - Item-based CF,
   - User-based CF,
   - SVD-based filtering,
   - Popular content,
   - Friends' top-rated content.

   Each strategy returns a maximum of 10 results, ensuring no duplication across methods.

5. **Explore More and Personalization:** Content already shown to the user is excluded from the `Explore More` section. Additionally, flags such as `liked`, `watchlist`, and `watched` are fetched using joins with the `likes`, `watchlist`, and `watch_history` tables, personalizing further exploration.

Periodic Updates: This entire process is triggered automatically every 10 minutes to ensure recommendations stay fresh.

5.7 Chats

- Implemented messaging feature by storing messages in a relational table.
- Designed discussion forums with votes.

5.8 Group Recommendation Selection Process

The group recommendation feature is designed to suggest content that is most suitable for a group of users who want to watch something together. The system follows a structured multi-step process to find the best possible recommendation by analyzing the preferences of all selected group members. Here is the process flow:

1. **Common Watchlist Check:** The system first looks for items that are present in the watchlists of *all group members*. This ensures that the suggested content is something everyone has already expressed interest in watching. The content can also be filtered by type (movie, show, or anime) based on the group's selection.

2. **Common Recommendations:** If there are no common watchlist items, the system generates individual recommendations for each group member and looks for overlap. It identifies content that is recommended for *every single member* of the group, ensuring a strong match in taste across the group.

3. **Highest Overlap:** In cases where no content is recommended to everyone, the system finds content that has the *highest overlap*, meaning it is recommended to as many group members as possible. The system also informs how many group members liked the content, helping the group decide even when there is partial agreement.

4. **Popular Content Fallback:** If the above steps do not yield any results, the system then falls back to suggesting *popular content* (e.g., trending movies or shows). This ensures that the group still receives a reasonable recommendation even when their tastes do not fully align.

5. **Exhausted Scenario:** If no content is found at all—whether common, overlapping, or popular—the system finally notifies the group that there are no more available recommendations for their selection, ensuring transparency and clarity.

Content Refresh: The recommendation logic also allows the group to exclude already considered items, so repeated calls can keep refreshing the recommendations without duplication.

5.9 Discussion Forums

Embedding Generation Pipeline

The embedding system leverages **Sentence-BERT (all-MiniLM-L6-v2)** to generate semantic vector representations of movie and show content. The key steps include:

- **Structured Metadata Processing:** Content metadata is transformed into structured text prompts by combining key fields:
  - `Title`
  - `Description`
  - `Genres`
  - `Director`
  - `Cast`
- **Special Handling:** Korean content is specifically tagged with the "Korean Drama" categorization to enhance relevance in recommendations.
- **Semantic Encoding:** Each content item is converted into a 384-dimensional embedding using the sentence transformer.
- **Storage:** Both the embeddings and original metadata are stored in JSON format for efficient retrieval.
- **Input Normalization:** Language inputs are normalized, e.g., converting "korean" to lowercase, to ensure consistency.

Query Processing System

The system supports natural language queries about movies and shows:

- **Encoding:** Queries are encoded using the same **Sentence-BERT (all-MiniLM-L6-v2)** model to maintain consistency with the content embeddings.
- **Similarity Computation:** Cosine similarity is calculated between the query embedding and all stored content embeddings.

Recommendation Logic

The recommendation module:

- Returns the **top 3** matches.
- Ranks results in descending order of similarity score.
- Includes detailed metadata (e.g., genres, director, cast) in each response to provide context-rich recommendations.

End-to-End Workflow

- **Training Phase:** A batch process takes JSON input files and generates embeddings for all content items.
- **Serving Phase:** The system handles real-time query answering, computing similarity scores on-the-fly.
- **Consistency:** Identical text processing and embedding methods are maintained across both training and inference stages to ensure reliable performance.

5.9.1 Integration with Backend Server

The embedding and recommendation pipeline is implemented in the `bot_recom.py` script. This script is invoked by the backend server whenever a **POST** request is made to the `/questions` endpoint. The server handles incoming user queries, routes them to `bot_recom.py` for processing, and returns the generated recommendations to the client.

5.10 Notifications

- Generated event-based notifications (friend requests, new messages, replies, recommendations from friends).
- Stored notifications in a table with read/unread status.
- Provided APIs to fetch and mark notifications as read.

6 Implementation Details

We began the development of the front end by using a pre-built free UI template to ensure a modern, responsive layout. The base template we adopted was sourced from [insert template link here – Jass will provide].

For backend functionality such as middleware integration and session management, we initially built upon code provided in our lab assignments. This included the implementation of CORS policies and token-based authentication mechanisms to secure API interactions.

To populate our database with realistic and diverse content, we utilized publicly available datasets from Kaggle, which included metadata for movies, shows, and anime.

Additionally, we used AI tools like ChatGPT-4 to refine and enhance the frontend user experience. These tools assisted with improving interface components, optimizing user flows across the application.

7 Future Work

- **Parental Control Mode:** Implement age-based filters and a child-safe browsing experience.
- **Collaborations:** Build collaborative watchlist and add functionalities of Friend Groups.
- **What's Trending:** Use APIs to fetch what content is globally trending.
- **Content Upload for Independent Creators:** Allow independent creators to showcase their work on the platform.
- **Gamification:** Introduce badges, achievements, and contribution scores to increase user engagement.
- **Mobile App Development:** Extend WatchGuru to Android/iOS for a seamless cross-platform experience.
- **Advanced Review Features:** Use NLP techniques to classify and display review sentiments and provide AI-based review summarization.

8 Conclusion:

In conclusion, WatchGuru has successfully achieved its goal of creating a centralized platform for entertainment enthusiasts to rate, review, and discover movies, shows, and anime. By combining personalized recommendations with social features like friend connections, private chats, and discussion forums, the platform offers a unique and engaging user experience. The integration of AI-powered recommendation models further enhances the relevance and usefulness of the suggestions provided to users. Throughout the development process, careful attention was given to building a scalable architecture and a clean, user-friendly interface. While the core features were implemented effectively, there is ample scope to expand the platform's capabilities in the future.