CS349 Project Report

Team Members

Pediredla Sahil Sriram	22B1062
Nuthakki Rithik	22B1008
Kajjayam Varun Guptha	22B1030
Rushikesh Rathamsetty	22B0948

Goal of the Project

The goal of this project is to develop a **shared review platform** where users collaboratively engage with content by **rating**, **reviewing**, and **recommending** movies, TV shows, and books. It goes beyond individual ratings by adding **social features** like **friend connections**, **mutual suggestions**, and visibility into others' watchlists.

A key component is a **recommendation algorithm** that uses **collaborative filtering** to suggest new content based on user preferences, making suggestions **personalized** and **socially relevant**.

The platform also highlights **popular trends** with sections like **Top Rated**, **Most Watched**, and **Most Reviewed**, making content discovery more **interactive**, **trusted**, and **community-driven**.

User Perspective

From a user's point of view, the application offers:

- A **personalized profile page** displaying ratings, reviews, watchlist, and received suggestions.
- Ability to **rate and review** movies, shows, and books.
- Maintain a unified **Watchlist** with a **Plan to Watch** status for all types of content.
- **Friend management** – add and remove friends, and view their activity.
- View and act on **suggestions made by friends**.
- Access to **automated recommendations** generated from their rating history using collaborative filtering techniques.
- Browse trending content through dedicated sections like **Top Rated**, **Most Watched**, and **Most Reviewed**.
- A more **interactive and social experience** through visibility into friends' reviews, watchlists, and suggestions.

Database Schema and Architecture

The application uses a normalized relational database ensuring data consistency, referential integrity, and efficient feature support across user interactions, content tracking, and social functions.

- **Users**: Stores credentials, profiles, and timestamps.
- **Items**: Unified table for movies, TV shows, and books with metadata and category field.
- **Genres** & **ItemGenres**: Enable many-to-many genre categorization.
- **UserRatings** & **UserReviews**: Link ratings (1–5) and reviews to users and items.

- **Watchlist**: Tracks user progress with content under types like `To be done`, `Already Done`.
- **Friends**: Manages bidirectional friendships with statuses and cascading rules.
- **Suggestions**: Stores peer content suggestions with optional reasons.
- **Recommendations**: Saves system-generated personalized suggestions with rationale.
- **Notifications**: Handles alerts for actions like friend requests and suggestions.

Composite keys and `CHECK/ON DELETE CASCADE` constraints preserve uniqueness and simplify cleanup. This modular design enables scalable backend features like recommendations, social features, and cross-category tracking.

Backend Overview

The backend of the shared rating system is built using `Node.js` with the `Express` framework and `PostgreSQL` for data storage. It handles user authentication, item management, ratings, and session management via RESTful APIs.

Authentication System

The authentication system includes:

- **Sign-Up API**: Registers new users with hashed passwords.
- **Login API**: Authenticates users and creates a session.
- **Logout API**: Destroys the session to log users out.
- **IsLoggedIn API**: Checks if the user is logged in.

Item Management and Rating System

The backend supports item management and rating:

- **Item Listing API**: Fetches items with search filters.
- **Item Details API**: Provides detailed item information.
- **Rating API**: Allows users to rate items.
- **Average Rating API**: Computes average ratings.
- **Rating Counts API**: Retrieves rating distribution.

Session Management

Sessions are managed with `express-session`, allowing users to stay logged in until they log out.

Database Interaction

The database is managed using the `pg` client:

- **Users**: Stores credentials and session data.
- **Items**: Contains item details.
- **Ratings**: Stores user ratings for items.

API Endpoints

Key endpoints include:

- **GET /list-tvshows**: Fetches TV shows.
- **GET /api/items/:category/:itemId**: Fetches item details.
- **POST /rate-item**: Allows users to rate items.
- **GET /api/user/stats**: Retrieves user statistics.

File Upload and Cloud Storage

`Multer` handles file uploads, with images stored on `Cloudinary`.

Friendship and Follow System

The system supports user interactions:

- **POST /api/friends/add**: Sends friend requests.
- **POST /api/follow**: Allows users to follow others.
- **POST /add-review**: Submits item reviews.

Frontend

Books.jsx – Books Exploration Page

`Books.jsx` enables logged-in users to search books via the OpenLibrary API and browse results in a responsive grid. Clicking a book adds it to the database and navigates to its details.

Key features:

- Checks login status using `GET /isLoggedIn`.
- Live search via OpenLibrary's `/search.json?q=`.
- Defaults to dummy books when query is empty.
- Adds selected books to DB and redirects to detail page.
- Responsive layout with styled search bar and grid.

Dashboard.jsx – User Dashboard

`Dashboard.jsx` displays a personalized dashboard for logged-in users, verifying session status and rendering navigation.

Key features:

- Verifies login via `GET /isLoggedIn`, redirects if not authenticated.
- Displays username with a welcome message.
- Integrates shared `Navbar` for navigation.
- Uses `useNavigate()` for redirection logic.

Details.jsx

The `Details.jsx` component displays detailed information about a selected item, including ratings, reviews, and cast/crew info, with support for rating and review submission.

Key features:

- Extracts `category` and `itemId` from URL parameters using `useParams`.
- Fetches and displays item details, average rating, rating distribution, cast/crew, and user reviews.
- Allows users to submit ratings (1–5 stars) and reviews.
- Displays rating distribution and percentage breakdown.
- Uses asynchronous fetch calls with error handling.
- Shows up to 10 cast members and filtered crew (Director, Writer, Producer).

Followers.jsx

The `Followers.jsx` component shows the logged-in user's followers, including their usernames and profile pictures, with search and navigation capabilities.

Key features:

- Fetches followers list via `/api/user/followers` API.
- Displays followers' profile pictures and usernames.
- Includes a search bar to filter followers by name.
- Handles loading and error states with appropriate messages.
- Shows default profile picture if not set.
- Allows navigation to a follower's profile page on click.

Following.jsx

The `Following.jsx` component shows the users the logged-in user is following, with search and navigation functionality.

Key features:

- Fetches following users via `/api/user/following` API.
- Displays users' profile pictures and usernames.
- Includes a search bar to filter users by name.
- Handles loading, error states, and redirects to homepage if unauthorized.
- Shows default profile picture if not set.
- Allows navigation to a user's profile page on click.
- Displays a message if no users match the search query.

Friends.jsx

The `Friends.jsx` component manages user friendships and following functionality.

Key features:

- Verifies login status and fetches username.
- Allows searching for users by query.
- Sends friend requests to other users.
- Enables following users for activity updates.
- Includes logout functionality.
- Handles API errors gracefully.
- Displays users' profile pictures with fallback.
- Conditionally renders search results and shows "No users found" message.

Friends1.jsx

The `Friends1.jsx` component displays and filters the user's friend list, with session-based authentication.

Key features:

- Fetches friends using `useEffect` and session-based authentication.
- Manages internal state for friends list, loading, and search.
- Provides a search bar for case-insensitive filtering.
- Displays a placeholder image for users without a profile picture.
- Redirects to a friend's profile on card click.
- Handles loading and empty states.
- Applies CSS from `friends1.css`.

Genres.jsx

This page displays items from a selected genre, fetched from the backend using the genre ID in the URL.

Key Features:

- Uses `useEffect`, `useState`, `useParams`, and `useNavigate`.
- Fetches genre items from `/list-genre-items/:genreId`.
- Displays a loading message until data loads.
- Dynamically sets the genre name and renders items in a responsive grid.
- Navigates to detailed views on item click.
- Includes hover effects and stylized cards.

Home.jsx

The homepage component manages user login, searches, and displays content such as movies, TV shows, and books with average ratings.

Key Features:

- Tracks user login and fetches the username.
- Provides a search bar for movies, TV shows, and books.
- Displays content based on login status and search query.
- Shows average ratings with stars and count.
- Allows navigation to detailed item views.
- Logs out the user and redirects to the homepage.
- Handles errors and uses dummy data for non-logged-in users.

Login.jsx

The Login component manages user authentication for the Movie Rating App.

Key Features:

- Checks if the user is already logged in and redirects to the home page.
- Manages form data for email and password inputs using `useState`.
- Sends login request to the server and redirects to home on success.
- Displays error messages and a loading state during login.
- Provides a link to the signup page for new users.

Movies.jsx

The Movies component allows users to search and explore movies in the Movie Rating App.

Key Features:

- Verifies user login status before displaying movie list.
- Allows movie search by title, updating state and calling the TMDB API.
- Displays movie titles, posters, and a placeholder if no poster is available.
- Adds selected movies to the database and navigates to the movie detail page.
- Includes error handling for failed login, movie fetch, and database addition.
- Displays loading state and dummy movies when no query is entered.
- Provides a logout button to redirect users to the homepage.

NotFound.jsx

Displays a simple 404 error message when a user navigates to a non-existent page.

Key Features:

- Shows ”404 - Page Not Found” heading and brief informational text.
- Minimal UI with no logic.

Notifications.jsx

Displays and manages notifications for the logged-in user.

Key Features:

- Checks login status and fetches notifications.
- Allows accepting/rejecting friend requests, deleting notifications, or clearing all.
- Displays loading and error messages; uses inline styling.

Profile Page

Displays the logged-in user’s profile with stats and profile picture.

Key Features:

- Verifies login; redirects if unauthenticated.
- Shows followers, following, items rated, and reviews.
- Allows updating profile picture and navigating to related pages.
- Includes logout functionality.

Profile1.jsx

Renders another user's profile using their username from the URL.

Key Features:

- Uses `useParams` to extract username and fetch profile data.
- Displays profile picture (or fallback) and user stats.
- Allows uploading a new profile picture.

Signup.jsx

Handles user registration and redirects based on login status.

Key Features:

- Checks if already logged in; redirects to dashboard.
- Manages form input for username, email, and password.
- Submits signup data and handles success or error.
- Provides link to login for existing users.

TVShows.jsx

Allows users to search and view TV shows via the TMDB API and interact with the backend.

Key Features:

- Verifies login; redirects if unauthenticated.
- Manages search, loading state, and session using `useState`.
- Fetches and displays TV shows based on search input; uses dummy data as fallback.
- Clicking a poster saves metadata via `/add-to-db` and redirects to details.
- Includes logout via `/logout`, topbar, and persistent `Navbar`.

Watchlist.jsx

Displays the logged-in user's watchlist of saved items from the backend.

Key Features:

- Fetches watchlist via `/list-watchlist` using credentials.
- Manages state and loading via `useState`; shows fallback on empty list.
- Displays items with metadata in a grid layout.
- Includes `Navbar`, `Navbar1`, and logout handling.
- Uses `watchlist.css` for styling.

Recommendations.jsx

Renders horizontally scrollable recommendations from a given user.

Key Features:

- Extracts username from URL via `useParams`.
- Fetches recommendations via `/list-recommendations/:username`.
- Displays items with images and titles; navigates to details on click.
- Handles loading/errors; shows fallbacks if empty or broken images.
- Styled with `recommendations.css`.

Implementation

The shared rating system was built using a modular full-stack stack: React.js (frontend), Node.js + Express (backend), and PostgreSQL (database).

Development was accelerated using AI tools like ChatGPT for API structure. The frontend used `create-react-app`, React Router, and modular JSX components with responsive utility-first CSS.

The backend employed Express.js routing, session middleware, and controller-based logic. PostgreSQL queries were refined using DBeaver and AI-suggested optimizations. The schema was prototyped with dbdiagram.io and implemented via migrations.

Security was enforced via bcrypt hashing, input validation, and session protection, using AI-reviewed best practices. This AI-assisted workflow enabled fast prototyping and iterative refinement focused on usability and coherence.

Future Work

To further enhance functionality and user engagement, future improvements could include:

- **Smarter Recommendations:** Add collaborative/content-based or hybrid filtering for improved suggestions.
- **Custom Watchlists:** Enable creation and sharing of themed playlists (e.g., ”Top 10 Sci-Fi”).
- **Enhanced Reviews:** Highlight top or recent reviews to guide user decisions.
- **Cross-Device Sync:** Allow seamless use across devices via cloud storage.
- **Social Sharing:** Add integration with platforms like Facebook or Twitter to boost reach.
- **Engagement Analytics:** Track user activity and reward contributions with badges or ranks.
- **Multilingual Support:** Localize content for global accessibility and inclusion.

These additions would elevate user experience and support platform scalability.