INF1005: Web Systems and Technologies

Project Report

Class: P5 Group 7	**Assignment:** Group Project
Professor: Goh Hui Lin Serena	
Name of Team Members	**Student ID**
Ng Yong Xian	2403911
Wong Gao Xiang Benson	2403915
Premanand Aishwarya Shri	2401690
Yeo Quan Ren	2403928
Lee Zhang Hui	2403896

Declaration:

We hereby pledge that this <INF1005 Group Project> task is not plagiarised and has been written wholly as a result of our own research and compilation of information.

Signed: Ng Yong Xian, Wong Gao Xiang Benson, Premanand Aishwarya Shri, Yeo Quan Ren,

Lee Zhang Hui

Dated: 4th April 2025

Introduction

The development team has implemented the Model-View-Controller (MVC) architecture for the PEOPLE'S MOVIE web application. This platform allows users to browse movies, submit, update, and delete their reviews, manage personal profiles, and track movies through their watchlist and watch history. Built using PHP, MySQL, HTML, and CSS, the application incorporates PHP Ratchet WebSocket technology to support real-time discussions between users about movies.

Design & Implementation of System

In the design and implementation of the PEOPLE'S MOVIE web application, the development team has utilized the Model-View-Controller (MVC) architecture to ensure a scalable, maintainable, and user-friendly system.

The Model interacts with the backend, performing crucial CRUD (Create, Read, Update, Delete) operations on the MySQL database. This includes managing user data such as profiles, reviews, and movie data, ensuring that all database transactions are securely handled. In this application, the Model abstracts the complexity of the database, ensuring that business logic is separated from data storage and retrieval, which promotes a cleaner, more modular structure.

The Controller acts as the intermediary between the Model and the View. It processes user actions such as submitting a review or adding a movie to a watchlist, performs necessary updates to the Model, and triggers changes in the View. This structure allows for better control over the application's flow and ensures that user interactions are smoothly processed.

For the View, the front-end is built using HTML, CSS, and JavaScript, with a focus on responsive design. The user interface is designed to be intuitive, allowing users to browse movies, manage their profiles, and interact with other users in real time. The integration of PHP Ratchet WebSocket enables seamless real-time communication between users, enhancing the social interaction features of the platform.

The application also utilizes the LAMP stack (Linux, Apache, MySQL, PHP) to power the backend. The choice of this stack ensures robust, reliable server-side functionality, supporting the dynamic content generation and interaction requirements of a movie review platform .

This architecture not only adheres to modern web development principles but also allows for future scalability and flexibility in adding new features such as advanced search filters, personalized recommendations, or even incorporating AI technologies for movie suggestions.

1. Model View Controller Architecture (MVC)

Model

The Model is responsible for interacting with the underlying database, such as performing CRUD operations (Create, Read, Update, Delete) and ensuring that business logic is implemented. In PEOPLE'S MOVIE:

- The Model manages the data related to movies, reviews, and user watchlists.
- It handles direct communication with the database to store user reviews, retrieve movie data, and manage watchlist updates.

The use of a Model allows for data abstraction, which means that the application logic can be written separately from the actual data storage, promoting a cleaner and more organized structure. The Model ensures that data access is centralized, simplifying future updates or changes to the database schema.

View

The View layer in the MVC architecture is responsible for displaying the user interface. It presents the data in a user-friendly way without directly manipulating or interacting with the underlying business logic or data. In PEOPLE'S MOVIE:

- The View represents the movie browsing pages, review submission forms, watchlist display, and real-time discussions.
- It displays information such as movie details, user reviews, and watchlists, dynamically updating as the user interacts with the platform.
- The separation of the View allows for changes in the layout or design without impacting the business logic or database layer.

The View can also be customized for different types of users or devices (e.g., desktop vs. mobile), improving user experience.

Controller

The Controller layer acts as an intermediary between the Model and View. It is responsible for handling user input, invoking the appropriate Model functions, and updating the View based on changes in the data. In PEOPLE'S MOVIE:

- The Controller handles user actions such as submitting reviews, updating them, adding movies to the watchlist, and managing user sessions.
- It performs data validation, sanitization, and ensures the proper flow of logic based on the user's actions.
- It is also responsible for managing real-time interactions through PHP Ratchet WebSocket, enabling users to chat about movies while browsing.

The Controller allows for a clear separation between the logic of user interaction and the actual data processing, enabling better maintainability and scalability of the codebase.

2. Back-End

API used

- **Themoviedb** - open source movie database
    - https://api.themoviedb.org/3/trending/movie/day?api_key=<API KEY>
    - https://api.themoviedb.org/3/movie/now_playing?api_key=<API KEY>
    - https://api.themoviedb.org/3/movie/upcoming?api_key=<API KEY>
- **Purgomalum** - open source service that returns masked vulgar comments
    - https://www.purgomalum.com/service/json?text=<input>

Libraries used

- PHPMAILER
- RATCHET

Endpoint Handling

The team has developed a robust routing system for the PEOPLE's MOVIE web application, designed to efficiently handle both static and dynamic routes. Static routes are utilized for predefined pages such as the homepage, login, and registration, ensuring direct access to essential functionalities with minimal processing overhead. Meanwhile, dynamic routes allow for flexible URL structures, enabling seamless navigation for user-generated content such as movie reviews, and chatrooms by dynamically retrieving relevant parameters from the URL.

This structured approach enhances both scalability and security within the application. By enforcing strict route definitions and pattern matching using regular expression, the system helps prevent forced browsing attacks, reducing the risk of unauthorized access to restricted resources. Additionally, the well-organized routing framework contributes to a smoother user experience by ensuring that navigation remains intuitive and efficient, while maintaining strict access control over sensitive functionalities.

Chat Server

For real-time communication and discussion among users, PEOPLE'S MOVIE utilizes Ratchet, a PHP WebSocket library, to implement a chat server. This enables users to communicate instantly about movies, share opinions, and interact with others while browsing the platform.

Chat Room Management

The chat server is designed to manage users across various chat rooms, which are typically based on movie titles or genres. Each user can join a relevant chat room to engage in discussions in real time. The Ratchet WebSocket server handles the following:

- **User Connections**: The server accepts WebSocket connections from users, ensuring that each user can join a chat room and start communicating with others.
- **Session Management**: The server keeps track of active users within each room and manages the flow of real-time messages between users. However, due to the lack of time, the team was not able to implement session authentication on the chat server, hence, highly skilled users can impersonate another person to chat with other users.
- **Real-Time Messaging**: Messages sent by users are immediately broadcasted to all connected participants in the relevant chat room, creating an interactive, live experience.

Non-Persistent Chat Rooms

However, it is important to note that the chat rooms in PEOPLE'S MOVIE are not persistent. The messages exchanged between users are not stored in the database, and the chat history is not retained after the WebSocket connection is closed. This design decision was made for several reasons:

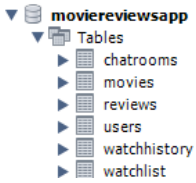
- **Focus on Real-Time Interaction**: The primary purpose of the chat feature is to facilitate real-time, spontaneous discussions. Storing every message could potentially impact the performance of the chat server and complicate the system unnecessarily. By not persisting messages, the chat server can maintain speed and efficiency in handling real-time communication.
- **Reduced Database Load**: Storing chat messages in the database would increase the load on the system, especially as user activity grows. By not saving every message, the application avoids unnecessary database writes, leading to more efficient resource management and ensuring better overall system performance.
- **Privacy and Data Retention**: Since the chat server does not store messages, users' conversations are temporary and not retained after the session ends. This provides a layer of privacy for users, as they know that their interactions are not logged for long-term storage. This can be especially important for platforms where user data privacy is a concern.
- **Simplified Architecture**: By avoiding the complexity of storing chat data, the architecture remains simple, reducing potential points of failure and maintenance overhead. This also helps ensure that the chat feature remains lightweight and does not complicate the overall application infrastructure.
- **Scalability**: As the user base grows, having non-persistent chatrooms ensures the application remains scalable. Storing large volumes of chat messages would require additional storage, complicate indexing, and potentially slow down database queries. With a non-persistent model, the system can easily scale without needing to allocate significant resources for message storage.

3. SQL Database

TABLES	COLUMNS
USERS	CREATE TABLE users (userid INT AUTO_INCREMENT PRIMARY KEY, username VARCHAR(50) UNIQUE NOT NULL, email VARCHAR(100) UNIQUE NOT NULL, password VARCHAR(255) NOT NULL);
MOVIES	CREATE TABLE movies (movieid INT PRIMARY KEY, moviename VARCHAR(255) NOT NULL);
REVIEWS	CREATE TABLE reviews (reviewid INT AUTO_INCREMENT PRIMARY KEY, userid INT NOT NULL, movieid INT NOT NULL, moviename VARCHAR(255) NOT NULL, rating INT NOT NULL, review_text TEXT NOT NULL, created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP, FOREIGN KEY (userid) REFERENCES users(userid), FOREIGN KEY (movieid) REFERENCES movies(movieid));
WATCHLIST	CREATE TABLE watchlist (userid INT NOT NULL, movieid INT NOT NULL, moviename VARCHAR(255) NOT NULL, PRIMARY KEY (userid, movieid), FOREIGN KEY (userid) REFERENCES users(userid) ON DELETE CASCADE, FOREIGN KEY (movieid) REFERENCES movies(movieid) ON DELETE CASCADE);
WATCHHISTORY	CREATE TABLE watchhistory (userid INT NOT NULL, movieid INT NOT NULL, moviename VARCHAR(255) NOT NULL, PRIMARY KEY (userid, movieid), FOREIGN KEY (userid) REFERENCES users(userid) ON DELETE CASCADE, FOREIGN KEY (movieid) REFERENCES movies(movieid) ON DELETE CASCADE);
CHATROOMS	CREATE TABLE chatrooms (chatroomid INT AUTO_INCREMENT PRIMARY KEY, chatroom_name VARCHAR(255) NOT NULL,

	userid INT NOT NULL, FOREIGN KEY (userid) REFERENCES users(userid) ON DELETE CASCADE);

The PEOPLE'S MOVIE web application uses a structured relational database to manage its core functionality, including user management, movie reviews, watchlists, watchhistory and chatrooms. The moviereviewsapp, schema, consists of the following tables, each with specific roles:



users Table

This table stores essential information about each user, such as their username, email, and password. It serves as the foundation for user authentication and authorization in the application. The *userid* field is used as a unique identifier for users and links them to reviews, watchlists, watchhistory and chatrooms.

movies Table

This table holds data related to the movies featured on the platform, specifically the movie name and a unique movie identifier. This table is referenced by other tables to associate user reviews, watchlist, watchhistory entries, and chatrooms with specific movies.

reviews Table

This table stores user reviews for each movie. Each review is linked to a specific user and movie through foreign keys (*userid* and *movieid*). The table contains the review text, rating, and the date the review was submitted. By connecting the review to both the user and the movie, it enables users to rate and share opinions about movies within the platform.

watchlist Table

This table tracks movies that users have added to their personal watchlists. Each entry links a user to a movie via foreign keys, allowing users to manage and track movies they intend to watch. The composite primary key (combining *userid* and *movieid*) ensures that each user can only add a specific movie once. Deleting a user will automatically remove related entries in the watchlist due to the ON DELETE CASCADE constraint.

watchhistory Table

This table tracks movies that users have added to their watch history. Each entry links a user to a movie via foreign keys, allowing users to manage and track movies they have already watched. The composite primary key (combining userid and movieid) ensures that each user can only add a specific movie once to their watch history. Deleting a user will automatically remove related entries in the watch history due to the ON DELETE CASCADE constraint. This structure ensures that users' watch history is accurately managed and tied to their profiles, providing a clear record of movies they have watched.

chatrooms Table

This table manages real-time discussions among users. Each chatroom is linked to a specific user (the creator of the chatroom) via a foreign key (userid) and contains the chatroom name. While chatrooms are not persistent in the database, this table serves to manage the creation and ownership of chatrooms.

4. Features Implemented (Functions)

Authentication/User Controller Functions

Login

After the user inputs their email and password, the login function handles user authentication by first checking if the request method is POST, else it redirects back to the home page. Firstly, it starts a session to store feedback messages, then trims and sanitizes both the email and password input. It then checks if either email or password fields are empty, and if so, prints out an error message. Then, it calls the login function from the user model to authenticate the user. If successful, it stores the user’s ID and username in the session. If authentication fails, it prints an error message in the outcome. Finally, it redirects back to the home page.

Register

In order to register for a new account, new users will have to come up with their username, input their email, as well as set a password and confirm the password. After that, the register function will check if the request method is POST, if not the function does nothing. It starts a session, then trims and sanitizes the inputs. It then checks if any fields are empty, and if so, prints out an error message. Next, FILTER_VALIDATE_EMAIL is used to validate the email format, and if invalid, sets an error message and redirects. It also checks that the password matches the confirmation password. If not, it sets an error message and redirects. Then, the password strength is validated, to make sure that it is at least 8 characters long, contains an uppercase letter, lowercase letter, a number and special character. If the password fails validation, it will print out an error message, then it calls the ‘register’ method of the ‘user’ model to create a new user. If successful, prints a

success message. If the username or email is already in use, a duplicate error message will be printed. For other failures, sets a general error message and redirects back to the register page.

Reset Password

We have implemented a reset password feature which first checks if the user is logged in, if they aren't, they will be brought back to the home page. After ensuring the user is logged in, the website will prompt the user to enter their current password and the password they want to update it to, as well as confirm the new password, while also checking that the new password fulfills the strength requirements.

Forgot Password

The team has implemented PHPMailer to enable the Forgot Password functionality in the PEOPLE's MOVIE web application. This feature is integrated with Brevo's (formerly Sendinblue) SMTP server for secure and reliable email handling. When a user requests a password reset, the system generates a random temporary password and sends it to the user's registered email via the SMTP server. This allows users to regain access to their accounts and subsequently update their credentials through the Reset Password function, ensuring a seamless and secure account recovery process. However, due to the lack of time, the team was not able to improve on the function using two-factor authentication methodology.

Update Account Information

Under your main profile, you are able to edit your account information, which will allow users to edit their username and email. The information is then pushed into the AuthController, which will use the updateProfile function to check if the username or password is already in the system, if it is it will error out, else it will print out a success message.

The user is also able to delete their account permanently through the **Delete Account functionality.** When the user clicks the Delete Account button, the system first prompts a confirmation message to ensure the user intends to delete their account, as this action is irreversible. Once confirmed, the form sends a POST request to the backend with the user's userid. The deleteAccount function in the UserController handles the request. It starts by verifying if the user is logged in by checking the session; if the session is not set, the user is redirected to the login page.

If the user is authenticated, the system proceeds to clean up associated data by invoking the deleteReviewsByUser method in the Review model to remove all reviews linked to the user, and the deleteWatchlistByUser method in the Watchlist model to delete the user's watchlist entries. Finally, the deleteUser method in the User model is called to delete the user's data from the users table. After removing the user's data from the database, the session is destroyed, and the user is redirected to a "goodbye" page or the homepage. This comprehensive process ensures that all data linked to the user is removed from the system, making the account deletion permanent and secure.

Movie Controller Functions

The MovieController in the PEOPLE's MOVIE web application is designed to handle three primary endpoints that facilitate movie search and discovery:

1. Search by Query (/search/query/{query}) – This endpoint allows users to search for movies based on a keyword. The system does not fetch an exact match for a specific movie title but rather returns a list of movies that contain the search term in their title. The information is retrieved using the TMDb Search API (/search/movie), which takes the user's input and returns a list of relevant results based on keyword matching.

2. Search by Genre (/search/genre/{genre}) – This endpoint enables users to browse movies based on their preferred genre. When a user selects a genre, the application retrieves the corresponding genre ID from a locally stored JSON file and uses it to fetch a list of movies from the TMDb Discover API (/discover/movie). The response provides movies that fall under the selected genre, ensuring users can explore content within their interests.

3. Movie Details (/movie/{moviename}) – This endpoint is triggered when a user selects a movie from the search results. The application first verifies whether the selected movie exists by comparing its title with TMDb's search results. If a match is found, the movie's details are displayed on a dedicated page. Additionally, the system fetches user-generated reviews from the database and calculates the movie's average rating. If a user is logged in, the application also checks whether the movie is in their watchlist.

These endpoints work together to create an intuitive and seamless search experience. By leveraging The Movie Database (TMDb) API, the system dynamically retrieves and updates movie-related data, ensuring that users always have access to the latest information. The structured routing system improves platform navigation, enhances user engagement, and allows for efficient content discovery.

Review Controller Functions

The ReviewController in the PEOPLE's MOVIE web application is responsible for handling user-generated reviews. It provides two main functionalities: submitting or updating a review and deleting a review.

1. The Submitting and Updating Reviews endpoint (/submitReview) allows users to either submit a new review or update an existing one for a particular movie. Prior to storing the review, the system performs several validation checks. These checks ensure that the user is logged in, that the review text is not empty and does not exceed the specified character limit, and that the rating falls within the acceptable range of 1 to 5. Additionally, the system sanitizes the input to prevent security vulnerabilities and filters out inappropriate language using the Purgomalum API to maintain the quality of content. If the review passes validation, it is either saved as a new entry or updates an existing review. Once submitted, users are redirected back to the movie page to view their contribution.

2. Deleting Reviews (/deleteReview/{reviewId}) – This endpoint enables users to delete their own reviews. When a deletion request is made, the system verifies whether the user is logged in and ensures that they have permission to remove the review. If the operation is successful, the review is deleted, and the user receives confirmation of the action. In the Review Model, sql statement was used and structured such that only reviews owned by the user can be deleted, and users cannot delete other users reviews.

By incorporating these endpoints, the system ensures a seamless review experience, allowing users to provide feedback on movies while maintaining content integrity. The validation processes, input sanitization, and moderation mechanisms enhance reliability and security, ensuring that the review system remains user-friendly and effective.

Watchlist Controller Functions

The `WatchlistController` in the PEOPLE'S MOVIE web application is responsible for managing the user's watchlist. It provides two primary functionalities: adding a movie to the watchlist and removing a movie from the watchlist. These endpoints ensure that users can personalize their movie experience by keeping track of movies they wish to watch.

1. **Add Movie to Watchlist** (/addToWatchlist/{movieId}&{moviename})
   The system first verifies if the user is logged in by checking the session. Once the user is authenticated, and when the user clicks on the 'favourites' icon, the system checks if the movie exists in the database using the verifyMovieinDB method from the Movie model. After verifying the movie, the addMovie method in the Watchlist model is called to add the movie to the user's watchlist. If the movie is successfully added, the system returns a success message: "Added to watchlist."

2. **Remove Movie from Watchlist**(/removeFromWatchlist/{movieId}&{moviename})
   Similar to the add functionality, the system first verifies if the user is logged in. Once the user is authenticated, and clicks on the 'favourites' icon again, the removeMovie method in the Watchlist model is called to remove the movie from the user's watchlist. If the operation is successful, the system returns a success message: "Removed from watchlist."

The WatchlistController provides a responsive and user-friendly interface for managing the users' watchlist. It also ensures that the user is authenticated, uses the database to keep the data integrity, and provides clear feedback to the user in the form of success or error messages. The service even allows customers to customise their movie experience while still ensuring a secure and reliable operation.

Watchhistory Controller Functions

The WatchHistoryController in the PEOPLE'S MOVIE web application is responsible for managing the user's watch history. It provides two primary functionalities: adding a movie to the watch history and removing a movie from the watch history. These endpoints ensure that users can maintain a record of movies they have already watched.

1. **Add Movie to Watchhistory** (/addToWatchHistory/{movieId}&{moviename})
   The system first verifies if the user is logged in by checking the session. Once the user is authenticated, and when the user clicks on the 'eye' icon, the system checks if the movie exists in the database using the verifyMovieinDB method from the Movie model. After verifying the movie, the addMovie method in the

WatchHistory model is called to add the movie to the user's watch history. If the movie is successfully added, the system returns a success message: "Added to watch history."

2. **RemoveMoviefromHistory**(/removeFromWatchHistory/{movieId}&{moviename} Similar to the add functionality, the system first verifies if the user is logged in. Once the user is authenticated, and clicks on the ‘eye’ icon again, the removeMovie method in the WatchHistory model is called to remove the movie from the user's watch history. If the operation is successful, the system returns a success message: "Removed from watch history."

The WatchHistoryController provides a responsive and user-friendly interface for managing the user's watch history. It ensures that the user is authenticated, uses the database to maintain data integrity, and provides clear feedback to the user in the form of success or error messages. The service allows users to customize their movie experience while maintaining a secure and reliable operation.

Moreover, the user is also able to remove the movies from the watchlist and watch history under their profile directly. The code for removing movies from the watchlist/watch history under the user's profile reuses the existing logic from the watchlist/watch history feature. By adding a remove button to each movie in the watchlist/watch history, the frontend makes an AJAX request to the backend to delete the movie using the movie's ID. The backend handles the request by removing the movie from the user's watchlist/watch history in the database, and the movie is immediately removed from the profile page without any popups, providing a smooth and intuitive user experience.

5. Front- End

Libraries used

FONT AWESOME FREE

Bootstrap Usage

All pages include bootstrap in the header include file. Bootstrap was used to center elements and make some elements responsive, as well as enabling the carousel in the home page.

CSS

No CSS framework, aside from Bootstrap for basic styling, was used. Responsiveness across pages is achieved through a combination of relative units, @media queries, viewports, and flexbox for most elements. The design also incorporates more advanced features like parallax scrolling for a polished look. Rather than adhering to a specific design philosophy, the focus was on prioritizing user experience (UX) by ensuring that all elements were easy to use, view, and access, while also making sure the site was fully mobile-friendly.

6. Javascript Features

DYNAMIC PASSWORD CRITERIA VALIDATION - in register.php

The team implemented a dynamic password criteria validation in register.php, to display the strength requirements needed for user passwords. This helps users to see what adjustments are needed for a secure password.

LOGIN MODAL - loginModal.js

This script manages the login, register, and reset password functionalities, including an eye toggle feature for password visibility. When users click the eye icon next to the password field, the togglePassword function is triggered. It checks the input type (either "password" or "text") and toggles between hiding and showing the password while changing the eye icon accordingly. This provides users with a convenient way to view and confirm their password while maintaining security when desired. Additionally, the script resets forms and clears error messages when modals are closed or switched, and handles the "Forgot Password" process by sending a POST request with the user's email, displaying success or error messages based on the response.

Carousell - home.js

Carousel is not JS feature, is a bootstrap feature

SEARCH BAR-AUTOCOMPLETE - searchbar.js

This autocomplete search feature significantly enhances the user experience by providing real-time, relevant movie suggestions as users type their search queries. By utilizing the TMDb API, the system ensures that suggestions are based on actual, up-to-date movie data, offering users a wide range of accurate and contextual options. This functionality reduces the need for users to manually type out full movie titles, minimizing the potential for errors and speeding up the search process. The immediate feedback in the form of suggestions helps guide users towards their desired movie, making the interface more intuitive and user-friendly. Additionally, the feature streamlines navigation, enabling users to quickly find and access movie details without unnecessary page loads or complicated search steps. Overall, it creates a smoother, more efficient search experience that meets the needs of users seeking to explore movies on the platform.

REVIEW-FORM MODAL - reviewform.js

The modal functionality implemented within the application provides users with an intuitive and efficient interface for creating, editing, and deleting movie reviews. This approach significantly enhances the user experience by minimizing page reloads and maintaining a seamless interaction flow.

- **New Review Modal**: The "New Review" button triggers a modal that allows users to enter a new review. Within the modal, users can input their review text and select a star rating, streamlining the process of submitting feedback without navigating away from the current

page. This enhances usability by providing a quick and intuitive means of submitting reviews.

- **Edit Review Modal**: Upon clicking the "Edit" button on an existing review, the modal appears with the current review details pre-filled, allowing users to modify the review as needed. This eliminates the need for users to leave the page or navigate through multiple steps to update their review, making the editing process more efficient and user-friendly.

- **Delete Review Confirmation Modal**: When a user opts to delete a review, a confirmation modal is displayed to prevent accidental deletions. This confirmation step ensures that the user explicitly intends to delete the review, providing an additional layer of protection and enhancing user confidence in the interface.

- **Star Rating System**: The modal includes a dynamic star rating system, where users can hover over the stars to highlight their desired rating and click to select it. This visual feedback mechanism facilitates a clear and engaging review process, making it easy for users to rate movies intuitively.

- **Seamless User Experience**: The modal windows are designed to appear and disappear smoothly, providing users with flexible control over their actions. Users can close the modals either by clicking the close button or by clicking outside the modal, further enhancing the convenience and usability of the feature.

By consolidating the review creation, editing, and deletion processes into modals, the application offers a streamlined and efficient interface that minimizes disruptions, ensuring a smooth and pleasant user experience. This design contributes to an intuitive, modern user interface, enhancing overall usability and satisfaction.

Chatting feature - chat.js

The chat feature of the application enables users to engage in real-time discussions within specific chat rooms. Here’s a breakdown of how it works:

1. **WebSocket Connection**: Upon loading the page, the system establishes a connection to a WebSocket server, which facilitates real-time communication. This allows users to send and receive messages instantly, creating a dynamic chat experience without needing to refresh the page.

2. **Joining a Chat Room**: Users can join a chat room either by selecting an existing room or creating a new one. Once a user joins, the room's messages are displayed, and the user is automatically linked to that room. The room name is shown on the interface, and all messages sent within the room are visible to every participant in real-time. If the user leaves the page and returns later, they are automatically rejoined to their last visited room.

3. **Sending and Receiving Messages**: When a user sends a message, it is transmitted to the WebSocket server, which then relays it to all participants in the room. Messages are displayed in the chat interface, with special formatting for messages sent by the user (e.g., different colors or styles). Additionally, user count is updated in real-time to reflect how many people are currently in the room.

4. **Creating and Leaving Rooms**: Users can create new chat rooms by entering a name for the room. Once the room is created, they are automatically joined. Users can also leave

a room at any time, which removes them from the chat and updates the UI to reflect that the room is no longer active for them.

5. **Message Formatting and User Interface**: The system automatically handles the display of messages, including distinguishing between different types of messages (e.g., system messages, user messages). It also provides features like real-time user count updates and user-specific styling for their own messages.

6. **Security and Input Handling**: To ensure safe communication, the system sanitizes input to prevent harmful content or malicious attacks, ensuring that all messages are safe to display.

This real-time chat feature enhances user experience by fostering immediate interaction, offering a straightforward interface for communication, and maintaining a seamless and dynamic chat environment.

7. Security Implementations

To safeguard the PEOPLE's MOVIE web application from common security vulnerabilities, the team has implemented a few security measures.

Prevention of Injection Attacks
To mitigate the risk of injection attacks, such as SQL injection, the team has incorporated strict input validation and sanitization procedures across the application. All user inputs are thoroughly validated to ensure that they adhere to expected formats, such as rating values being within a defined range or ensuring that review texts are not empty. Additionally, user input is sanitized to eliminate potentially dangerous characters or malicious code, such as HTML or JavaScript, using techniques like stripping tags and escaping special characters. This minimizes the possibility of executing harmful code, safeguarding the application from potential exploits.

Prevention of Forced Browsing
To prevent unauthorized users from accessing restricted or private pages through direct URL manipulation (force browsing), the team has implemented a robust routing system along with security configurations in the .htaccess file. The router ensures that users can only access routes that are defined in the system, which helps prevent unauthorized access to non-public areas of the site. Moreover, through .htaccess, additional access controls are enforced, such as restricting direct access to sensitive files or enforcing user authentication where necessary. These combined measures help to ensure that users cannot bypass intended access controls by simply altering URLs.

Session Management
The team utilized PHP's built-in session management to maintain user state across different pages of the PEOPLE's MOVIE web application. PHP sessions allow the application to store user-specific data, such as login status, user ID, and other relevant information, securely on the server side. This ensures that once a user logs in, they can navigate the website without needing to authenticate repeatedly. Additionally, the system uses session variables to handle important information, like the movie a user is viewing or their review status, which persists across multiple requests.

Conclusion

Overall this project was a very good learning experience of modern web development, offering valuable insights into real-world development processes. It gave us the opportunity to engage with the entire web development lifecycle, from front-end design to back-end logic.

Throughout the development, we were able to refresh and enhance our knowledge of HTML and CSS, learning how to structure a responsive and visually appealing user interface. Additionally, we gained hands-on experience with PHP and MySQL, utilizing these technologies to build dynamic, data-driven web pages. This project also allowed us to explore the fundamentals of website hosting through cloud services, which expanded our understanding of deploying and maintaining web applications.

However, there are several areas where the project could be improved or enhanced. For instance, while the MVC architecture was implemented effectively, integrating modern JavaScript frameworks (such as React or Vue.js) could improve the user experience by making the site more dynamic and responsive. Additionally, incorporating user authentication mechanisms, like multi-factor authentication, would enhance security for user accounts.

In terms of performance optimization, implementing caching strategies or lazy loading for media content could improve load times, especially as the platform scales. On the back-end, introducing more advanced error handling and logging systems would help maintain the stability of the application in production.

Lastly, expanding the project to include machine learning algorithms for personalized movie recommendations or implementing a social media integration for users to share reviews could elevate the user experience, making it more engaging and interactive.

Despite these areas for improvement, this project provided a strong foundation in modern web development, and the knowledge gained will definitely be useful for future web development projects.

AI Usage

- CHATGPT was the team’s main assistant in building the PEOPLE’S MOVIE web application.
- Through the assistance of the AI, the Model View Controller Architecture/File structure was set up in the team’s repository for easy management and better organisation of code. It also defines clear separation of the layers that each individual will be working at for prevention of merge conflicts.
- Through the assistance of AI, the team were able to gain ideas that were recommended/suggested from the AI. One example is implementing chat rooms using web sockets to allow users to chat with each other in the web application.

- As the team has no idea how web sockets worked, the use of AI helped to understand the flow of a Chat server. Additionally, the Ratchet library was suggested by Chatgpt for implementation in our web application. Hence, functionalities of the chat server were created through the assistance of the AI.
- Other instances where AI was used is when some of the html/php/javascript code was not working as intended, hence the team had used AI to figure out what was the issue or error and fix the code through the use of AI.
- AI was used to rephrase the paragraphs the team had written to improve on the phrasing and sentence structure to make it more professional.
- AI was used to generate some mobile friendly HTML and CSS (specifically using @media and other similar tags) for reference purposes.
- Add more if have…

Contributions

Features		**Contributor**
Setting up Model View Controller Architecture and Routing		Ng Yong Xian
Back-end	Setting up chat server using web socket - Ratchet library Configuration of Apache server to proxy web socket connections	Ng Yong Xian
	Created a Router Class to handle endpoints for static and dynamic URI routes	Ng Yong Xian
	Setting up sendinblue smtp server to automate sending mail for Forgot Password function using PHPMailer library	Ng Yong Xian
	Setting up SQL database relations	Ng Yong Xian & Benson
	Create, update, delete reviews functions [Controller & Model]	Ng Yong Xian
	Fetching movies from themovieDB API by genre/search [Controller]	Ng Yong Xian
	Forgot password function [Controller & Model]	Ng Yong Xian
	Usage of Purgomalum API service to mask vulgar comments [Controller]	Ng Yong Xian
	Login Function	Lee Zhang Hui
	Register Function	Lee Zhang Hui
	Reset Password	Lee Zhang Hui
	Update Account Information	not
	Model handling to about page	Yeo Quan Ren
	Watchlist Function	Benson Wong
	Profile	Benson Wong

	Linking Review and Watchlist to Profile	Benson Wong
	Sanitize review input	Benson Wong
	NavBar backend	Yeo Quan Ren
	Watch History Function, Link to Profile	Aishwarya
	Delete User Account Permanently Function	Aishwarya
Front-end	Search bar autocomplete/movie suggestions (Javascript)	Ng Yong Xian
	Create/Update review form & delete review warning modal (Javascript)	Ng Yong Xian
	Search result page - search.php (CSS) [View]	Ng Yong Xian
	Chatroom page - chatroom.php and chat.js (CSS & Javascript) [View]	Ng Yong Xian
	Home page CSS & JS	Yeo Quan Ren
	Dynamic password criteria validation (JS)	Lee Zhang Hui
	About page CSS & JS	Yeo Quan Ren
	Movie review page CSS & JS	Yeo Quan Ren
	Home page Responsive/Mobile Friendly	Yeo Quan Ren
	About Page Responsive/Mobile Friendly	Yeo Quan Ren
	Movie Review Page Responsive/Mobile Friendly	Yeo Quan Ren
	Profile Page CSS & JS	Yeo Quan Ren
	Profile Page Mobile Friendly/Responsive	Yeo Quan Ren
	Watchlist CSS & JS	Benson Wong

	NavBar styling and responsiveness	Yeo Quan Ren
	Watchistory CSS & JS	Aishwarya
	Privacy Statement, FAQ pages, Footer CSS	Aishwarya
	Watchlist, WatchHistory JS - from profile.php	Aishwarya
	Edit Profile CSS (for Delete Function), Goodbye Page CSS	Aishwarya
	Forgot Password page, Reset Password page CSS	Aishwarya
	Register/Login Page CSS & JS	Aishwarya