Instalite - *A social media platform geared towards movie lovers*

The following is a report on a multi-use social media app. In this report, we'll go through different features / parts of the tech state of the application, and describe the technical components as well as the design decisions and reasons for each of them. Changes and lessons made along the way are noted with the features/components that they relate to.

Tech Stack

AWS RDS and S3

The app uses AWS RDS instead of DynamoDB for its database due to how it would be easier to work with the SQL queries, since most of the functionality relied on rigidity of SQL tables. Also, there were some imagined complex SQL queries to be used for user management, authentication, and some of the follower/following logic and RDS works better with these than DynamoDB.

AWS S3 (Simple Storage Service) was used to keep track of user information, specifically the user photos and other static assets. The reason for this was that S3 is robust and highly available, which made it ideal for storing and retrieving any amount of data at any time, from anywhere on the web. It ensures high durability of data and is cost-effective for scaling making it a beneficial tool for the application.

ChromaDB

ChromaDB was beneficial due to its efficient handling of high-dimensional data and its ability to perform nearest neighbor searches, which are essential for facial recognition tasks. This choice supports scalable and fast searches for matching facial embeddings. ChromaDB was used for the celebrity face matching functionality by storing and querying facial embeddings.

Next.js

The application uses Next.js as the React framework for building the user interface of the application. Next.js is useful because of how it simplifies routing between pages and works well with backend API routes in Node.js. Next.js was used on the frontend user interface of the application throughout various features.

The reason Next.js was so useful was because of how it enhances user experience by organizing content across multiple pages rather than a single-page application (SPA). This structure aids in loading specific resources only when needed, which can improve

page load times and overall performance. Since the application involved many different features and a variety of pages, it was useful to have this improved functionality with Next.js

NodeJS

Node.js was used as the backend runtime environment for the application, because it is highly scalable and efficient for handling asynchronous tasks, which is crucial for a multi-use social media app with real-time interactions and updates. It also is easily compatible with JavaScript, which helped to have a unified language across the tech stack.

Tailwind CSS

Tailwind CSS was leveraged to design the user experience on the front end, while using standard .css files for different pages to style them each individually. The reason for this was to facilitate a better user experience, particularly in low-light environments, and meet modern web design trends and accessibility standards. The use of Tailwind CSS also helped us implement a dark mode theme as an extra credit feature.

Components of the Application

Session Management

There is session management that uses req.session to manage user sessions on the server-side. This session includes the information of user_id and email and helps with registration.

The reason for implementing this feature was to ensure that session state is maintained securely server-side, reducing client-side vulnerabilities. It allows for reliable tracking of logged-in users and handling authentication state efficiently. It also just makes sense to have some sort of session management when thinking about sessions and logins for a social media application.

Chat Mode

The application allows the user to enter a chat mode which facilitates real-time communication between users. The feature allows one on one messaging as well as group chats where any user in the group can send a message and see the group's messages. Users can also send and receive invitations to join chat rooms and subsequently view messages specific to each room. The application uses websockets

(throughSocket.IO) for the real-time data exchange ensuring state updates to all active users.

In terms of the technical aspects, this component uses Next.js and both React and jQuery on the frontend, while using Node.js and Express.js on the backend. It also uses a MySQL database to store the data of the chatrooms, specifically tables for chat_rooms, room_users, and messages.

There were various decisions made along implementing this component that led to certain design choices. One challenge encountered was differentiating between the types of state updates needed to be made, and which components needed to be updated based on what state they had access to. The states of the current user sessions and sockets and states of the RDS database needed to be kept in sync at different parts in the code, which was difficult to keep track of. This was resolved by (tediously) using the socket.io library to emit events to all connected clients, and updating the state of the components accordingly.

Natural Language Search

The natural language search feature allows users to query a LLM model to query about items related to the application. This can be anything including posts, people, movie reviews, actors, general content, and more. This was built using React on the frontend to manage the state of the current and former search results as well as the search queries. The backend is built using Node.js and Express.js. This part of the application queries the OpenAI API, specifically the GPT-3.5-turbo model to generate search results in plain language based on the user queries. It combines this with Retrieval Augmented Generation (RAG) to add a vector store to provide a more relevant context for more accurate search results.

Some design choices made along the way involve how to utilize the search history. The app feeds the LLM the history of the user's search queries to provide it a more localized context between the model and user.

Social news streaming Adsorption ranking

Additional Features

Dark mode

Using Tailwind CSS, there is a dark mode implementation where users can click and toggle between the modes.

Forgot password

In user authentication, there is a forgot password feature that allows a user to reset their password by clicking a “forgot password” button.

WebSockets in chatbot for real-time communication

By using WebSockets in the chatbot feature, messages were sent and received instantly in both the one on one chats and also in the group chats, improving the overall conversation experience. This implementation not only made communication faster but also opened up possibilities for further improvements and growth in the chatbot application.