Final Design Report

Design Description

1. Logical Organization

We have a logically organized modern scalable MERN stack architecture as our system, shown in Figures 1 and 2, suggests. Using a component-based architecture encased in necessary providers for routing, state management, and authentication, the frontend is constructed using React. State management is achieved by leveraging custom hooks and the Context API. We used Express.js and Node.js as middleware and backend for security, request processing, and authentication. We used Mongoose as a layer between the backend and our DB and MongoDB is our primary database. Reddit API, Google Gemini API and Google OAuth are the external services that the system integrates.

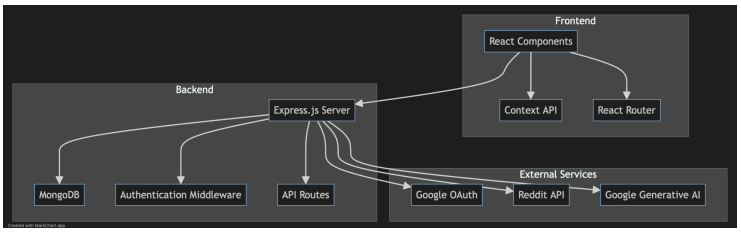

Figure 1: High-level system architecture

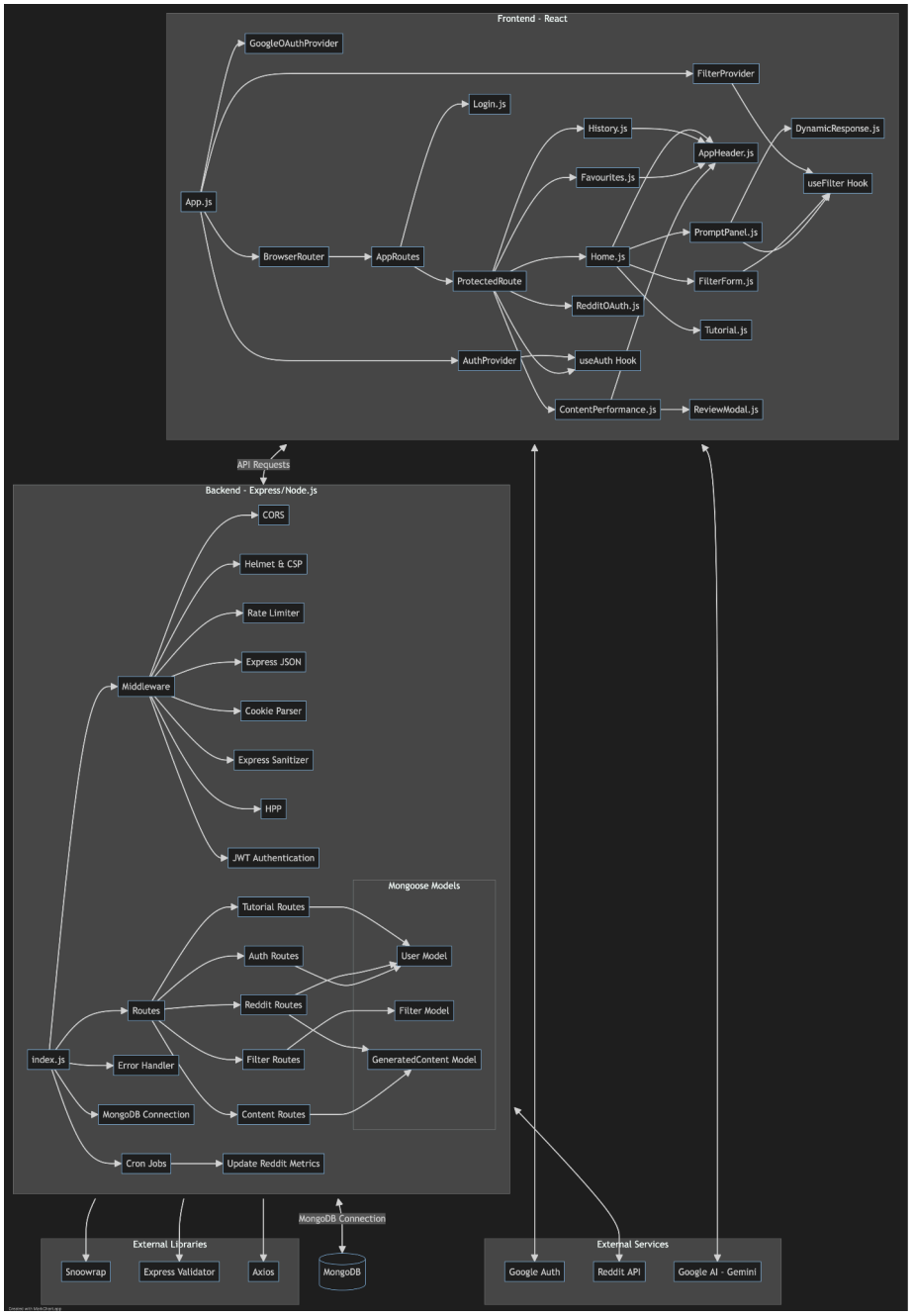

Figure 2: Low-level system architecture

2. Tech Stack

Frontend: React, Ant Design, React Router, Axios
Backend: Node.js, Express.js, MongoDB with Mongoose
Authentication: JSON Web Tokens (JWT), Google OAuth
External APIs: Reddit API, Google Generative AI (Gemini)
State Management: React Context API

3. Component Boundaries

There is a clear separation between frontend and backend components as referenced in Figure 2.

Frontend Organization: We organized the frontend of the application mainly by distinguishing between primary pages, reusable components (FilterForm, PromptPanel) and component contexts used for state management (AuthContext, FilterContext). In general, this kind of system of components architecture proved support for both maintainability and reusability of the system. But on retrospect, there are some aspects which could have been improved, for instance:

1. Redundant Code in History and Favourites: After reviewing, we found that the Favourites.js and History.js components were very similar. The approaches which both components' content items display and interaction phases are similar. Predictably, we should have built a more adaptable component which could be configured to display either favorites or history. This would have improved the factors of code maintenance and drastically reduced code redundancy.

Backend Organization: There is only one enormous index.js file on the backend in which there is every piece of server logic, route and controller code. Such a monolithic approach, while speeding up the early stages of development, showed growth-related problems as the project progressed.

1. Modular Backend Structure: While working on the backend module, we could have developed our backend code in many modules instead of the single monolithic block of code. Such an organizational structure would have improved both readability and maintainability. For instance, it would have been easier to update a certain feature if we had, for instance, separate user and content routes files.
2. Social Media Integration: At first, we only worked on integrating Reddit, and we did so for a number of different components and routes. But since we think that in the future there will be a need to integrate into more social networking sites, we would like to adopt a different strategy that is more flexible and scalable. In one of the future versions we would create a more generic integration module, which was easy to extend to other platforms.

If we were to rebuild the application, it is most probable that there would be a similar separation of the frontend and the backend code, although some restructuring of the code has already proven necessary. However, the overall effect would be the creation of more large-scale components which would make it unnecessary to rewrite the same code and in this case, the social media integration system would be designed in a way that many platforms would be supported easily and lastly, modular codes cut across the backend allowing ease maintenance in the future.

4. Justification of Logical Organization

The choice of technology in the project was decided by the team's active practical work of a JavaScript developer within the MERN stack. React made the most sense as the frontend framework, mainly because of its principles of building user interface within components and because we know how to use the library. For state management solutions, Context API was favored instead of redux because it offered sufficient functionality to the application while the codebase was simpler. Backend consisted of a combination of Node.js and Express.js framework because of its large base of the middleware. As there was no clarity on the DB schema at the project inception, MongoDB was preferred owing to its capability to deal with non-structured data. Also, the reason for using Ant Design and Ant Design templates UI components was because we were looking for a similar style and not for development of a lot of custom CSS.

5. System Usage and Operation

It is required to have the following installed on your machine, which are: Node.js (version 14 and above), npm, MongoDB (v 4.4 and above) and Git. First, clone the repo and change the directory to the project directory. Head to the my-backend folder to install the backend dependencies and execute 'npm install'. Add some environment variables in the .env file in the my-backend for managing database, username, api key amongst others. For the front end, go to the my-app folder and run 'npm install' to install the dependencies. Similarly, create a .env file in the my-app folder for environment variables pertaining to the front end. Then, go to the my-backend directory and type 'npm start' to start the backend server then open another console and go to the my-app directory and type 'npm start' to start the frontend development server. From now on, the application should run on http://localhost:3000.

6. User Interaction

Once a user logs in using the google account for authentication, he or she is able to access the application. Users are also able to control the content creation process by applying certain filters, for example content type, industry, content type, audience target age etc. Filters are essential as it aids the user in producing content according to their custom tone

by using AI technology. The system has made it possible for users to save both generated content and preferred filters for use at a later date with ease. The application provides useful features where users can manage their content and also manage the content they want to expand on by looking into the previously created versions of the content. Content Performance is one of the most important and useful features where users can link their Reddit accounts for self-service platforms bypassing where they can add, edit or delete active posts from the application. The users can now also measure their content performance in terms of the activity of the stored content such as comments and up votes that have been made on that content. The system also offers an embedded demo for new users which covers the basic features and processes within the system making sure there are no issues during registration and helping them understand how to use the system.

Evaluation

1. User Authentication and Authorization

Test Overview:

1. Navigate to the application and then try signing in into the application using Google OAuth.
2. After the sign in, try to navigate protected routes (e.g., /home, /history) etc.
3. Refresh the browser, you should still remain logged in.
4. Log out and navigate to protected routes again.

Results:

A successful application of Google OAuth login was designed. Logging in with Google accounts was possible for the users. After signing in, users satisfied with the home audience could move to such features as protected routes. After a successful logout from the application, protected routes posting redirects to the home page were blocked.

Assessment:

- Strengths:
    - The use of Google OAuth enhances the security of the application by enabling secure login of users in an easy way.
    - The sessions of the users are well managed with the use of JWT.

- Weaknesses:
    - The only method of logging into the application available at the moment is Google OAuth and this may be a drawback for those who do not wish to use Google Oauth.

Next Steps:

- Add alternative ways of logging in (e.g. email and password) as more services should be made available to users.

2. Content Generation

Test Overview:

1. Log in to the application.
2. Go to the home page.
3. Using various filter options, fill the content generation form with the required information.
4. Click on the ‘Generate’ button to generate content.
5. Check the screen for the content that has been generated.

Results:

The content generation functionality was developed as planned based on the Google Generative AI API and that’s the end of the story. Users, for their part, were able to specify different filters, and the content would be generated in accordance with these inputs. The content which was generated was projected on the screen and could be retrieved or modified.

Assessment:

- Strengths:
    - The incorporation of Google Generative AI ensures quality and relevant content for the users trying to generate contextual and high-quality content.
    - The filter options are used to generate content by targeting particular areas of concern.

- Weaknesses:

- Generating content gets slow especially in case of longer prompts.
- We may get rate-limited after sometime since it's a free API.

Next Steps:

- Think of implementing some more free fallback API if gemini api get's rate limited to ensure availability of the application or simply buy the API Active.

3. Content Management (History and Favorites)

Test Overview:

1. Make use of the content generation feature to multiple content.
2. Go to the History page pick and confirm that all the content generated is present.
3. Mark certain pieces of content as favorites.
4. Go to the Favorites page and confirm that those items that have been marked are present.
5. Delete some content from History and Favorites and refresh the tab to verify their deletion.

Results:

In the History and Favorites sections, everything was done quite successfully. The content created was shown on the History page as it should have been. Users could select content as favorite and it was displayed on the Favorites page. The delete feature performed without issues from the History and the Favorites pages as well.

Assessment:

- Strengths:
  - The system operates appropriately in terms of preservation and retrieval of user content.
  - Setting content as favorite provides an avenue for the users to categorize their best content effectively.
  - The pages also have search and filtering options for content that would come in handy when users have a lot of content.

- Weaknesses:
  - The History and Favorites pages also have large portions of the codes that are not uniquely designed and this could pose a risk of complications when updating the system.

Next Steps:

- Refactor the History and Favorites components to use a generic component to reduce code duplication.

4. Reddit Integration

Test Overview:

1. Proceed to the Content Performance page.
2. Try to link your Reddit account.
3. Post/Edit/Delete any of the generated content to Reddit.
4. Verify that posted content is updated on Reddit.
5. Check the Content Performance page for updated metrics (upvotes, comments).

Results:

Integration with Reddit was done. Users were able to hyperlink their accounts to Reddit in order to publish/edit/delete posts to the platform.

Assessment:

- Strengths:
  - The process of linking users' reddit accounts is implemented correctly.
  - Publishing, moderating and removing content directly from the application for reddit works seamlessly for the users.

- Weaknesses:
  - Currently the only integration option available is Reddit which is a disadvantage to those who like to use other social media.

Next Steps:

- Begin work towards the inclusion of more social media networks into the application further enhancing the application's attractiveness.