



# PromptLibrary

CS4624: Multimedia, Hypertext, and Information Access

Virginia Tech

Blacksburg, VA 24061

05/06/2024

Authors: Ryan Wang, Brandon Hoang, Daniel Shin, Gabriel Shin

Client: Yinlin Chen

Instructor: Edward A. Fox

# Table of Contents

<b>Table of Figures</b> .....	<b>3</b>
<b>Table of Tables</b> .....	<b>4</b>
<b>1. Abstract</b> .....	<b>5</b>
<b>2. Introduction</b> .....	<b>6</b>
2.1 Problem Statement.....	6
2.2 Motivation.....	6
2.3 General Approach.....	6
<b>3. Requirements</b> .....	<b>8</b>
3.1 Functional Requirements.....	8
3.2 Non-Functional Requirements.....	8
3.3 Deliverables.....	9
<b>4. Design</b> .....	<b>10</b>
4.1 Frontend Design.....	10
4.2 Backend Design.....	11
<b>5. Implementation</b> .....	<b>12</b>
5.1 Website Overview.....	12
5.2 Technology Stack.....	12
<b>6. Testing</b> .....	<b>16</b>
<b>7. Users' Manual</b> .....	<b>17</b>
7.1 Home Page.....	18

7.2 Sign in and Authentication.....	19
7.3 Prompt Exploration and Export.....	21
7.4 Prompt Review.....	22
7.5 Prompt Submission.....	23
<b>8. Developer’s Manual.....</b>	<b>24</b>
8.1 Project Overall Dynamic and Interaction.....	24
8.2 Frontend.....	25
8.3 Backend.....	27
8.4 Development Description.....	29
<b>9. Lessons Learned.....</b>	<b>34</b>
<b>10. Timeline/Schedule.....</b>	<b>36</b>
<b>11. Future Plans.....</b>	<b>38</b>
11.1 Authentication.....	38
11.2 Incorporation of a Rating System.....	38
11.3 More Efficient Full-Text Searching.....	39
11.4 Scale up the Database.....	39
11.5 Direct Integration of Accounts.....	39
11.6 Continuation of Frontend Accessibility Improvements.....	40
11.7 UTF-8 Encoding.....	40
<b>12. Acknowledgements.....</b>	<b>41</b>
<b>13. References.....</b>	<b>43</b>



# Table of Figures

<b>Figure 1: Entity Relationship diagram for the PromptLibrary database.....</b>	<b>14</b>
<b>Figure 2: Relational Table for the PromptLibrary database.....</b>	<b>14</b>
<b>Figure 3: Screenshot of the home page.....</b>	<b>19</b>
<b>Figure 4: Screenshot of the sign in page.....</b>	<b>20</b>
<b>Figure 5: Screenshot of database page for normal users.....</b>	<b>20</b>
<b>Figure 6: Screenshot of the details page for normal users.....</b>	<b>21</b>
<b>Figure 7: Screenshot of the database page for administrators.....</b>	<b>22</b>
<b>Figure 8: Screenshot of the details page for administrators.....</b>	<b>22</b>
<b>Figure 9: Screenshot of the prompt submission page.....</b>	<b>23</b>
<b>Figure 10: Project UML Diagram.....</b>	<b>24</b>
<b>Figure 11: Project frontend directory structure.....</b>	<b>29</b>
<b>Figure 12: Project backend directory structure.....</b>	<b>32</b>

# Table of Tables

**Table 1: Milestone Timeline and Schedule..... 36**

# 1. Abstract

The rapid advancement and integration of Large Language Models (LLMs) in academic research and the broader technological ecosystem underscores the critical need for specialized, context-rich training datasets. The PromptLibrary project is designed to address this gap by establishing a comprehensive library of high quality prompts. This initiative aims to amass a wide array of prompts, thereby forming an essential instruction dataset to enhance the precision and relevance of LLMs. By encapsulating real-world inquiries and scenarios, this dataset is poised to significantly improve LLMs' learning capabilities and adaptability. The project features a web-based, searchable repository that allows for the submission and retrieval of high-quality prompts, and a quality assurance mechanism to maintain prompt quality. This repository not only serves as a critical resource for fine-tuning LLMs but also fosters a collaborative environment for librarians, educators, and researchers, thereby advancing the utility of LLMs in academic settings. For the semester of Spring 2024, the team was able to produce a proof of concept as the client requested, and was able to create a project prototype that met client specifications. The current version of the project supports functionalities such as prompt submission, database searching, prompt retrieval, user authentication, and exporting one, several, or all prompts.

## 2. Introduction

### 2.1 Problem Statement

The proliferation of LLMs in academic research has highlighted a significant gap in the availability of specialized, context-rich instruction datasets. Traditional datasets often lack the specificity and diversity required to effectively train LLMs for academic applications, leading to models that may not fully comprehend or accurately respond to complex, domain-specific queries. This limitation not only hampers the potential of LLMs in academic settings but also constrains the scope of research and inquiry that can be supported by these advanced artificial intelligence (AI) tools.

### 2.2 Motivation

The motivation behind the PromptLibrary project stems from the critical need to enhance the performance and applicability of Large Language Models (LLMs) within academic libraries. By creating a repository of high-quality, academically relevant prompts, the project aims to provide a foundational dataset that can significantly improve the learning capabilities of LLMs, making them more responsive to the nuanced demands of academic research. Furthermore, the project seeks to foster a culture of collaboration and knowledge sharing among librarians, educators, and researchers, thereby enriching the academic community's collective resources and methodologies.

## 2.3 General Approach

The project's approach involves the development of a web-based repository that enables the submission, categorization, and retrieval of instructional prompts tailored for academic use. This platform will incorporate a rigorous quality assurance mechanism to ensure the relevance and effectiveness of the prompts, thereby creating a valuable resource for training and fine-tuning LLMs. In addition to serving the immediate needs of academic libraries, the repository is envisioned as a model for community-driven AI resource development, demonstrating the potential for collaborative efforts to address complex challenges in the realm of AI and education.

## 3. Requirements

These requirements were gathered through meetings with Dr. Yinlin Chen. We acknowledge that seeking out these requirements continues to be an ongoing process as new features and preferences are discovered. Below is an outline of the requirements that we have discovered from our meetings, as well as our own findings.

### 3.1 Functional Requirements

- Prompt Submission
- Prompt Retrieval and Export
- Prompt Searching, Filtering, and Tagging
- Prompt Review and Approval
- User Authentication

The PromptLibrary will be able to store manually input prompts from authenticated users and place the prompts into a queue, where a super-user will review and approve or deny the prompt. Super-users can also remove prompts that are outdated or that are not highly rated. Prompts that have been approved should appear to all users. Users can search for prompts using a text matching search and a tagging system. Results can be filtered based on tags and ratings. Users can select prompts to export in a JSON format or export the entire library as a JSON file.

### 3.2 Non-Functional Requirements

- Performance
- Scalability

- Security
- Available for free

PromptLibrary will be fast, lightweight, and space efficient. This app will be dockerized [16] and flexible in its deployment, which will become expensive if the size of the library becomes too large. The library will also be free to use and available to anyone, even without an account.

### 3.3 Deliverables

- A web-based searchable repository
- A quality assurance mechanism
- A data export function

The final product is a web-based repository that supports searching, access to high-quality prompts, and a prompt submission mechanism that has a review process for super-users to approve prompts. A function to export the library or parts of the PromptLibrary in JSON format is also necessary.

## 4. Design

### 4.1 Frontend Design

The frontend is a website that allows the user to interface with the database backend. It uses the Vue framework [1] to generate reactive web pages. The client has emphasized that frontend work should focus on being functional. There are five desired web pages specified by the client: landing page, sign in page, prompt submission page, library showcase page, and prompt details page. Some pages may have variations of the page to support features such as prompt deletion. Each page shares a header and a footer. The header contains the website title and sign in button when not authenticated and a log out button when authenticated. The landing page is minimal and supports text searching and redirecting to the library showcase. The sign in dropdown supports user authentication via Google OAuth 2 [3] through Firebase [2], and redirection to an internal sign-in page. The prompt submission page supports manual input of prompt information and tagging. The library showcase page supports exploration of prompts via scrolling, text searching, and tag searching. It will also contain the export function as a button, and includes management functionality if an administrator signed in. Administrators will be able to see modified versions of all the pages with additional permissions. This is done through a wrapper view that, based on the user's account status, displays the appropriate view. Prompt management is available only to administrators and super-users and supports prompt approval and deletion.

## 4.2 Backend Design

The backend design of this project consists of the server-side functionality of PromptLibrary. The main parts consist of the database and user authentication. The database will store and manage all the necessary information pertaining to the library such as prompts, users, and the corresponding metadata. The structure of prompt entries within the database is inspired by an existing library [14] given by Dr. Chen in order to ensure the storage of all required attributes of a prompt. This also includes other functionality such as fetching any data requested by the user and exporting the database in the form of a file for fine-tuning as requested by the client. Authentication will be done to allow users to sign in and create prompts. Having users sign in to add prompts is necessary for quality control purposes. Authentication functionality will also be synced to the database to allow for the storage of user specific data. The database can be queried either through text-match searching, tag filtering, or both. Text-match searching returns prompts which best match the text within the query, while tag filtering simply only displays prompts with the specified tags. This allows for multiple ways for users to search and find their desired prompts.

# 5. Implementation

## 5.1 Website Overview

The PromptLibrary project's main purpose is to be a fully digital library used to store and record useful and interesting prompts tailored for LLMs like ChatGPT [4] and Google Gemini (previously Bard) [5]. PromptLibrary will also support an export feature to assist in the fine-tuning process for LLMs.

## 5.2 Technology Stack

Backend: Firebase for authentication and Prisma ORM [6] with PostgreSQL [7]

Frontend: Vue [1]

Docker [16] for flexibility in hosting

### 1. Users and Authentication

Users are classified into three categories within the system: unregistered users, registered users, and administrators. Unregistered users are users who have not registered with the library and are given the least access to the system. They will only be allowed to view and search the library. Registered users are users who have signed in through Google [2]. These users can view the library but also add prompts, add tags, upvote prompts, and leave reviews. Registered users are expected to represent most users. Finally, administrators are registered users who have been given administrator status. This enables the user to remove and approve submitted prompts.

Authentication was planned to be handled through Amazon Cognito [8] and AWS Amplify [9], but due to difficulties with using AWS Amplify, authentication was run through Firebase instead. This allows users to sign in through a Google account.

## **2. Data Storage and Collection:**

Prompt entries will be submitted by users who possess a registered account.

Submissions will be done through a form in which the user must fill out details such as the prompt, the response, and the LLM the prompt utilizes. Any submissions must be approved by an administrator or super-user to ensure quality for the library. However, unregistered users can still view and search the database.

In terms of storage, a combination of Prisma ORM [6] and PostgreSQL [7] will be used as a database for the project's backend. A relational database format was chosen as the data is highly structured, as the same data is required for each user and prompt entry. Minimal variety and change in data is expected. The Prisma schema follows the drafted Entity Relationship diagram and Relational Table in Figures 1 and 2, respectively. The database schema for the PromptLibrary is normalized and there are no anomalies when adding, deleting, or changing data. As for searching and querying, this was done through the use of Prisma ORM only.

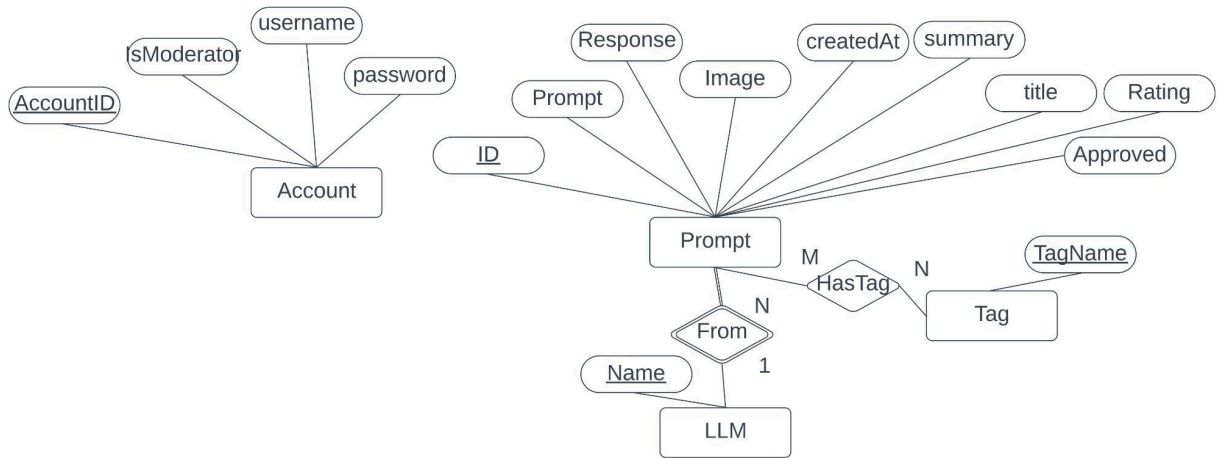


Figure 1: Entity Relationship diagram for the PromptLibrary database.

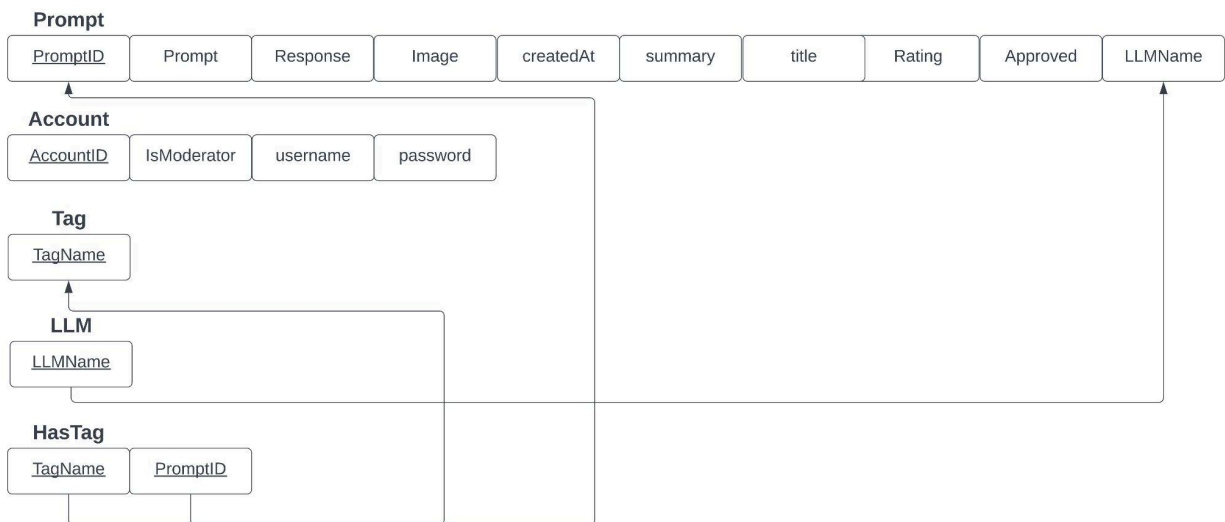


Figure 2: Relational Table for the PromptLibrary database.

### **3. Export Functionality**

The main function desired by the client is the ability to export the data from the library for use in fine-tuning LLMs. This functionality will be accomplished through PostgreSQL functionality and will allow the user to select specific prompts or tags and export the prompts in the form of a file. This allows a user to have flexibility in exporting specific or many prompts at a time.

### **4. Docker Container**

In order to easily run the project without manually installing any dependencies, the entire project can be run utilizing Docker containers. Both the frontend and backend possess their own docker-compose files to run each individually for testing. In order to run the entire project, there is another docker-compose file within the project directory which can be used to serve both the frontend and backend simultaneously.

## 6. Testing

Testing focused primarily on two attributes: usability and query functionality – with a heavy emphasis on the former. The purpose of this project is to provide an easy-to-use library that most users can comfortably access, thus an emphasis on user comfort is vital. For this, the team utilized basic internal systems tests to ensure the functionality of the product. The client specified that further user testing would be conducted by a separate team after the delivery of the project.

As for query functionality, tests were conducted to examine the basic functionality and the general stress the system can withstand. After creating an initial seed for the database containing various prompts populated by the team, basic tests were conducted to examine the basic functionality of the database. Basic tests include adding an entry, reviewing and approving an entry, deleting an entry, adding and removing tags from an entry, account sign in, account log out, and account registration. Query requests and responses were sent and received utilizing Postman [19]. The basic tests were then followed by a second round of more advanced tests in order to measure the success and accuracy of the implemented search functionality including filtering by tags and the best match search. First, each query was tested utilizing increasingly complex queries and timed in order to evaluate efficiency. Each result was then manually examined in order to measure the accuracy and effectiveness of the query. If the time would be too high or the result were to include inaccuracies, an update would be developed and delivered for further testing.

## 7. Users' Manual

PromptLibrary is a specialized repository designed for prompt storage. Its primary function is to collect and share instructional prompts that enhance the training of LLMs. The platform provides a user interface for searching, submitting, and reviewing prompts to ensure a comprehensive collection of high-quality, academically relevant instructional data.

### **User environment requirements:**

- Basic knowledge of how to use a computer
- Network connection
- A web browser

The PromptLibrary is engineered to offload the bulk of computational tasks to our dedicated backend infrastructure, thus alleviating the need for users to have specific technical environments on their local machines. The only requirements for users are a stable network connection to facilitate data exchange with our server, and a contemporary web browser capable of handling HTTP protocols, HTML, JavaScript, and CSS to ensure accurate data representation. Users should possess basic digital literacy skills, including the ability to navigate and operate their computer to interact effectively with our digital library. Tasks supported by the PromptLibrary include:

- Conducting searches within the prompt repository using keywords, context, and other relevant criteria.

- Browsing through various categories and exploring a wide array of instructional prompts.
- Submitting new prompts to the library for inclusion in the repository, following a quality assurance process.
- Removing existing prompts, a feature available to users with administrative privileges.
- Exporting selected prompts in convenient formats such as JSON for use in external applications or for further analysis.
- Utilizing the PromptLibrary's resources for the purpose of research, teaching, or enhancing LLMs within academic settings.

## 7.1 Home Page

- The home page presents users with a search bar for prompt inquiries as shown in Figure 3. Users can enter keywords or phrases to locate specific prompts and click "Search" to submit a query.
- An "Explore" button is available for users to browse through a collection of prompts without a specific query.

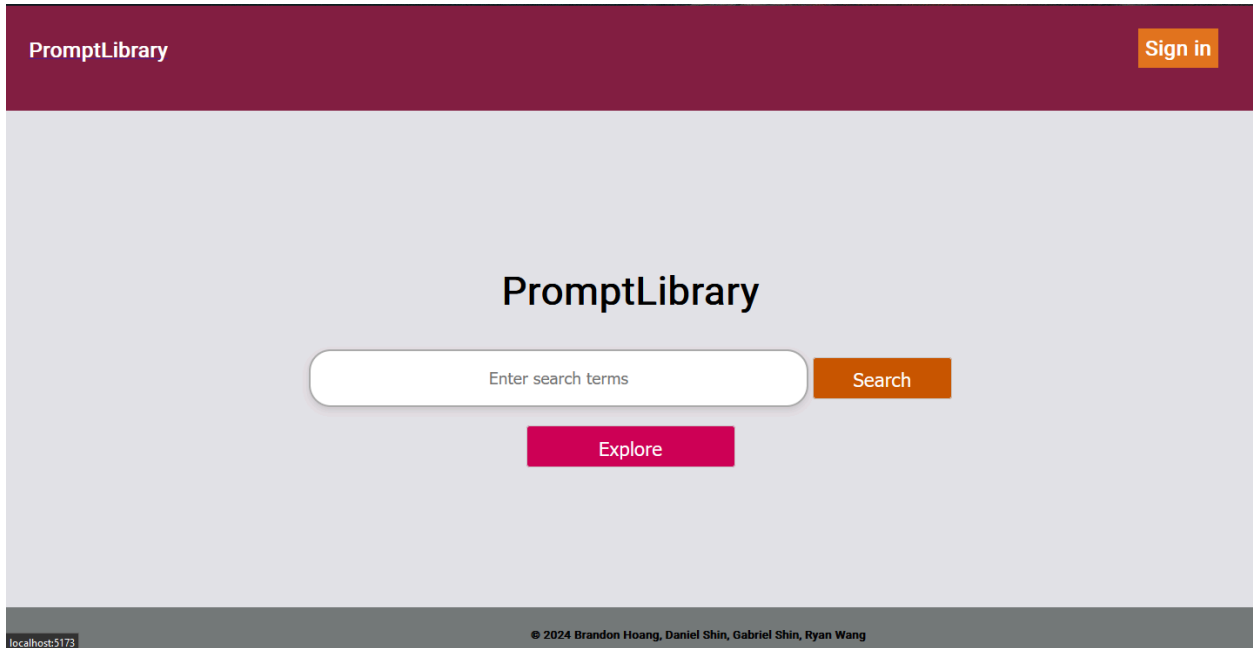


Figure 3: Screenshot of the home page

## 7.2 Sign in and Authentication

- For submission or review of prompts, users must sign in. The sign in button is in the upper right corner of the header.
- Users can employ the integrated "Sign in with Gmail" feature for a streamlined authentication process. Administrators can sign in by entering their registered username and password after clicking the "Sign in as Administrator" button as shown in Figure 4.

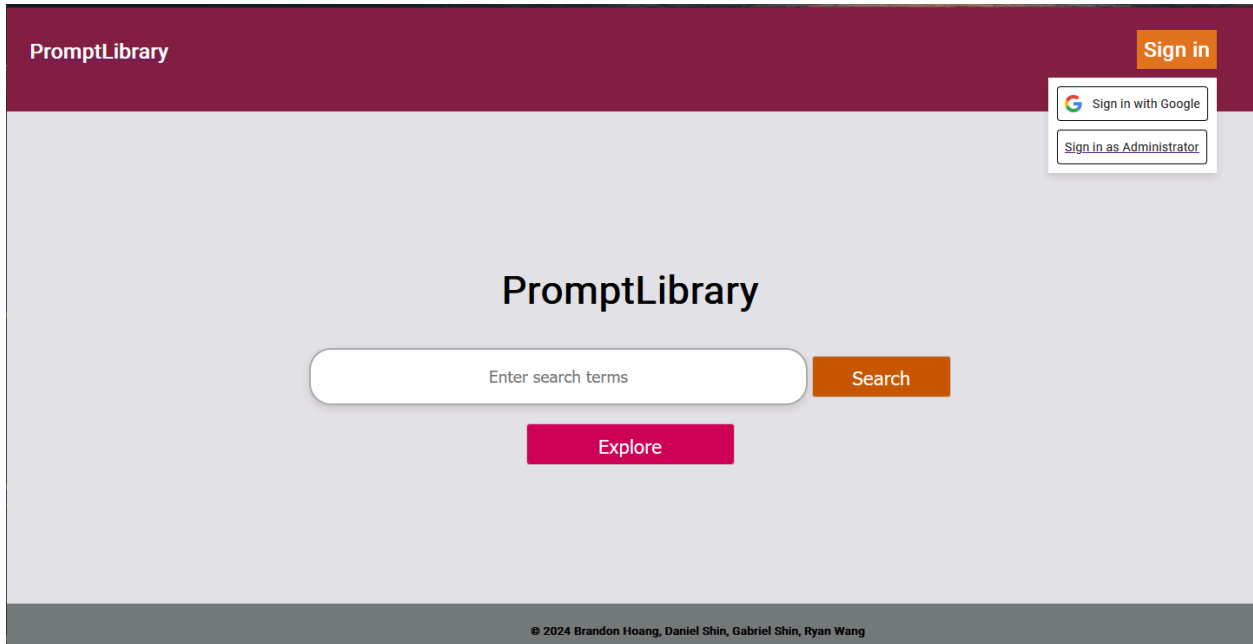


Figure 4: Screenshot of the sign in page

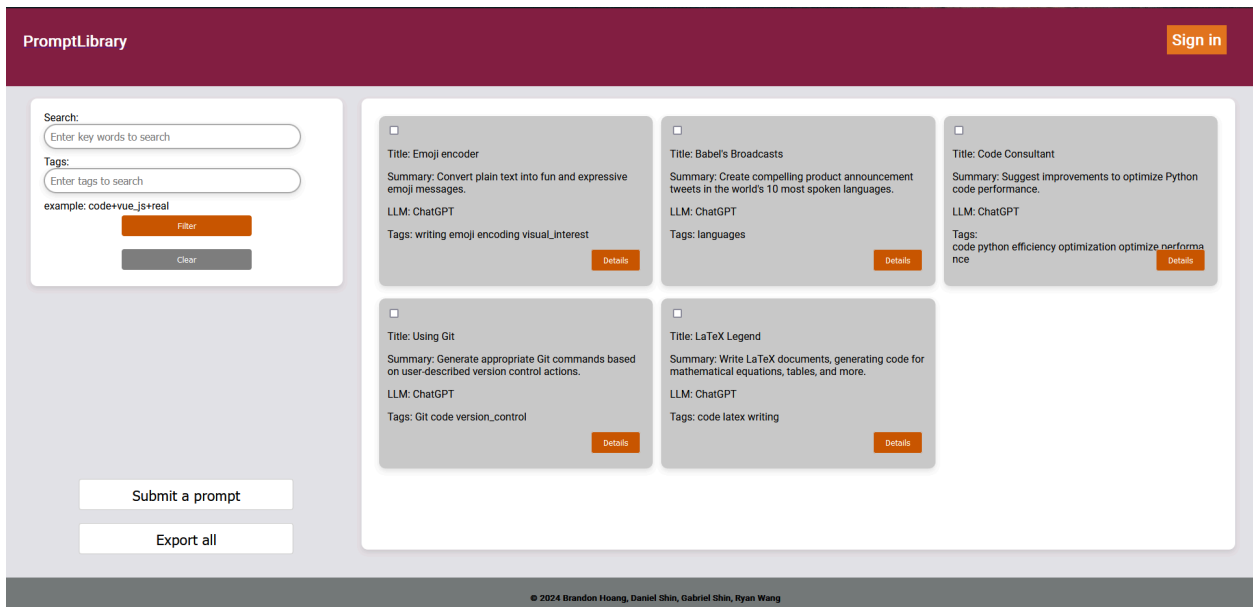


Figure 5: Screenshot of the database page for normal users

## 7.3 Prompt Exploration and Export

- An exploration page for viewing prompts is available for all users, as shown in Figure 5. Prompts are displayed as “cards” and contain basic information about that prompt.
- Filtering by tag is done through the left-hand panel. Users can select one or more tags, or filter by text-match search.
- Prompt cards can be expanded to see more details. Clicking on a card will cause a redirect to a page that contains the prompt’s details, as shown in Figure 6.
- Prompts can be selected for export via checkbox. These checkboxes can be found in the upper left corner of a prompt’s card.
- The export button is at the bottom left corner of the web page. After clicking this button, the selected prompts will be exported into a JSON file and downloaded into the user’s default local directory. When no prompts are selected, the button will by default export all prompts in the database.

**PromptLibrary** Sign in

---

**Code Consultant**

Summary: Suggest improvements to optimize Python code performance.

---

Prompt: Your task is to analyze the provided Python code snippet and suggest improvements to optimize its performance. Identify areas where the code can be made more efficient, faster, or less resource-intensive. Provide specific suggestions for optimization, along with explanations of how these changes can enhance the code's performance. The optimized code should maintain the same functionality as the original code while demonstrating improved efficiency. `def fibonacci(n): if n <= 0: return [] elif n == 1: return [0] elif n == 2: return [0, 1] else: fib = [0, 1] for i in range(2, n): fib.append(fib[-1] + fib[-2]) return fib`

---

Response: The provided code generates a list of Fibonacci numbers up to the nth number. While the code is functional, there are several areas where it can be optimized for better performance: 1. **Use Memoization**: This technique involves storing the results of expensive function calls and returning the cached result when the same inputs occur again. In this case, we can use memoization to avoid recalculating Fibonacci numbers that have already been computed. This can significantly improve the performance, especially for larger values of n. 2. **Reduce Memory Usage**: Instead of storing the entire list of Fibonacci numbers, we can just keep track of the last two numbers. Since each Fibonacci number is calculated based on the previous two numbers, we only need to store the last two numbers in memory. Here's the optimized version of the code incorporating these improvements: `python def fibonacci(n, memo={}): if n <= 0: return [] elif n == 1: return [0] elif n == 2: return [0, 1] elif n in memo: return memo[n] else: fib = [0, 1] for i in range(2, n): fib.append(fib[-1] + fib[-2]) memo[i+1] = fib[-1] return fib` Explanation of Changes: - **Memoization**: We use a dictionary 'memo' to store previously calculated Fibonacci numbers. Before computing a Fibonacci number, we check if it's already in the memo. If it is, we return the cached result. This eliminates redundant computations and improves performance, especially for larger values of n. - **Reduced Memory Usage**: We only keep track of the last two Fibonacci numbers in the list 'fib'. Instead of storing the entire list, we update these two numbers iteratively as we generate Fibonacci numbers. This reduces memory usage and makes the code more efficient. With these optimizations, the code should run faster and use less memory, especially for larger values of n.

---

LLM: ChatGPT

Tags: code python efficiency optimization optimize performance

[Back to Library](#)

© 2024 Brandon Hoang, Daniel Shin, Gabriel Shin, Ryan Wang

Figure 6: Screenshot of the details page for normal users

## 7.4 Prompt Review

- Administrators can choose to approve or delete prompts from the database. This can be done through the card view or through the detailed prompt views as shown in Figures 7 and 8.

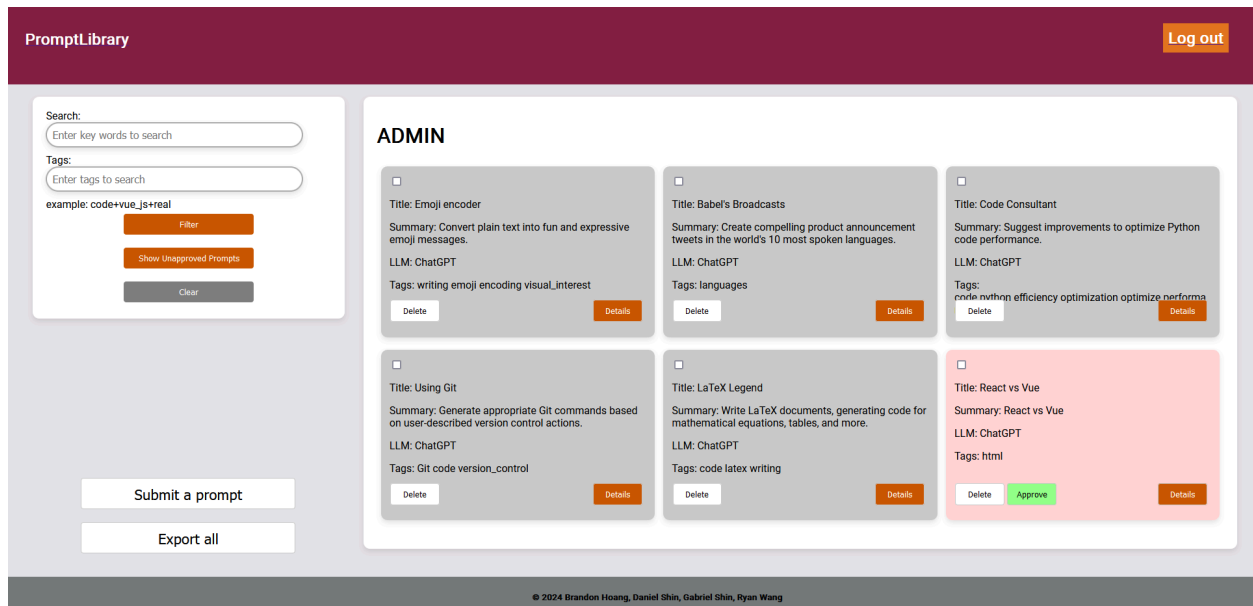


Figure 7: Screenshot of the showcase page for administrators

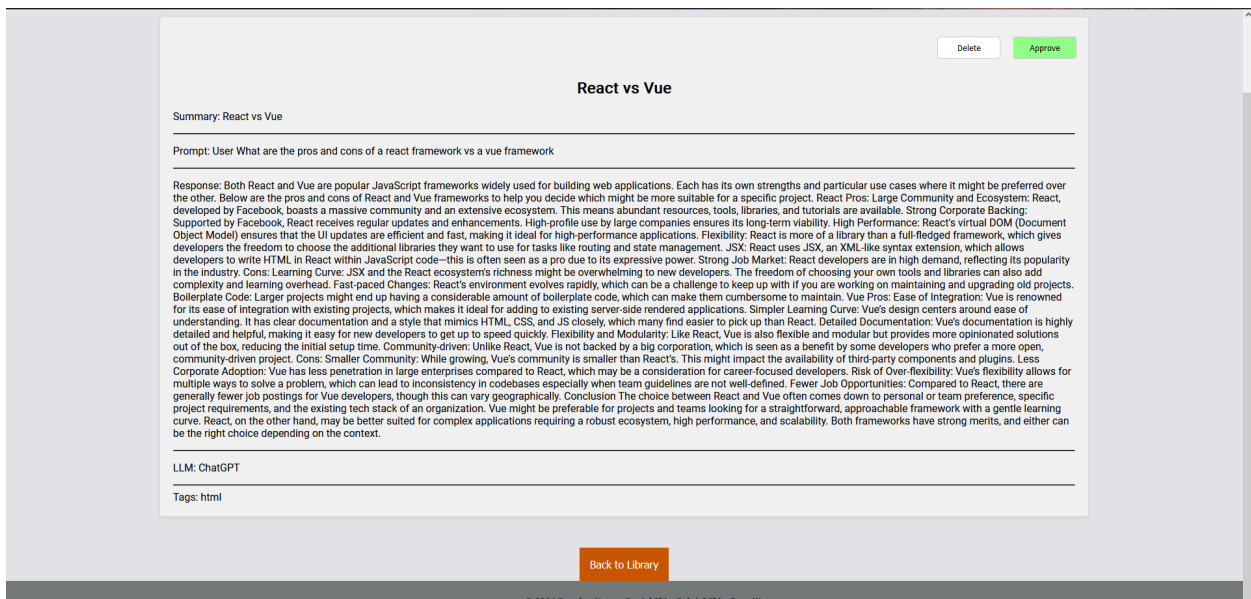


Figure 8: Screenshot of the details page for administrators

## 7.5 Prompt Submission

- Users have the option to submit prompts by selecting "Submit a prompt" from the library showcase as seen in Figure 9.
- The submission form requires users to enter details such as title, summary, prompt, response, tags, and the applicable LLM.
- All submissions undergo a review process to ensure the quality and relevance of the content in the repository.

The screenshot shows the 'Submit a Prompt' form on the PromptLibrary website. The form is titled 'Submit a Prompt' and includes the following fields and instructions:

- Title\***: Enter a one to three word title for the prompt.
- Summary\***: Enter a short one to three sentence summary of the prompt.
- Prompt\***: Enter prompt.
- Response\***: Enter sample response.
- Tags**: Include tags as comma separated values with no spaces between tags.
- Language Model\***: Select a Language Model (dropdown menu).

At the bottom of the form is a large orange 'Submit' button. Below the form is a 'Back to Library' button. The footer of the page contains the copyright notice: © 2024 Brandon Huang, Daniel Shin, Gabriel Shin, Ryan Wang.

Figure 9: Screenshot of the prompt submission page

# 8. Developer's Manual

## 8.1 Project Overall Dynamic and Interaction

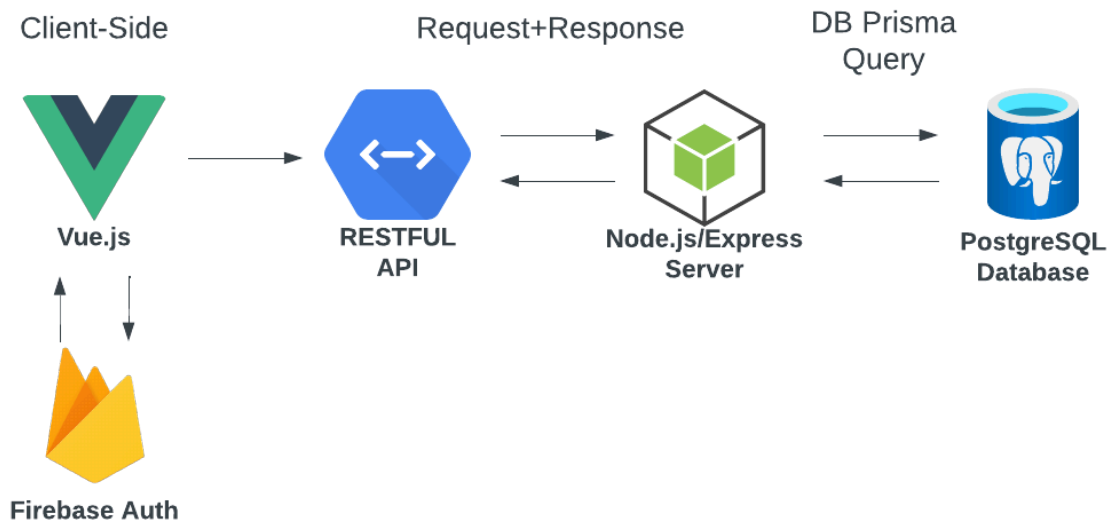


Figure 10: Project UML Diagram

The architecture of the project is composed of several interconnected components, as can be seen in Figure 10. The client-side uses Vue.js [1] for frontend development, utilizing its reactive and component-driven framework to build a dynamic and responsive web interface. Additionally, Firebase Authentication [2] is integrated to manage user authentication processes, ensuring secure and reliable user login and registration functionalities.

On the server side, the application communicates through a RESTful API connected to a Node.js/Express server [10]. This configuration facilitates efficient handling of HTTP requests and delivers data in a structured format. The Node.js/Express server further

interacts with a database management layer utilizing Prisma Query [6], which provides an intuitive and secure interface for database operations. For data storage, the system uses a PostgreSQL database [17] supporting the project's data-intensive operations and ensuring data integrity.

## 8.2 Frontend

The frontend of the project is developed in Vue with Node.js. Below are the instructions the developers should perform before forking the repository.

1. Create Working Directory
  - a. Create a file directory on your computer. This will serve as the central location for all your projects.
2. Install Java
  - a. Ensure Java 17 SDK is installed on your system.
3. Install IntelliJ IDEA Ultimate [20]
  - a. The frontend is developed using IntelliJ IDEA [12].
  - b. Ensure that the development machine follows the minimum specifications for IntelliJ IDEA:
    - i. The latest 64-bit Mac OS, or
    - ii. 64-bit Windows 10, and
    - iii. 8 GB RAM, and
    - iv. An SSD Drive with 5GB of free space, and
    - v. A monitor with 1920x1080 resolution

- c. It's possible to run IntelliJ [12] with less than 8 GB of RAM, but it can be slow. It's possible to run IntelliJ with a regular hard drive, but it takes a while to boot up. We highly recommend satisfying these requirements.
4. Install Node.js [10]
  - a. Download and install the latest LTS version of Node.js from the official site.
5. Install Vite [11]
  - a. Install Vite globally using the command `sudo npm install -g vite`. If you have previous versions, update Node.js [10] and Vite to their latest versions.
6. Install Vue Plugin in IntelliJ [12]
  - a. In IntelliJ, open "Preferences", navigate to "Plugins", search for "vue" in the Marketplace, and install it. Restart IntelliJ if required.
7. Install Vue Devtools Extension [1]
  - a. For Chrome or Firefox, install the Vue.js devtools extension from the browser's extension manager.

You will need to navigate to the repository link:

`https://github.com/narwhalle/PromptLibrary`

and clone it onto your machine. To do this, follow the instructions listed below.

- If you're planning on using your local machine, open a terminal of your choice such as git-bash or zsh and run:

`git clone https://github.com/narwhalle/PromptLibrary.git`

- Or if you're planning on developing remotely using the secure shell (SSH) run:

```
git clone git@github.com:narwhalle/PromptLibrary.git
```

## 8.3 Backend

The backend requires both PostgreSQL [7] and Prisma ORM [6]. The former is the relational database chosen for the PromptLibrary while the latter is an Object Relational Mapper (ORM) to the said database. This ORM functions as a bridge between the backend code and the PSQL database within the `prisma.schema` file. Additionally, the ORM is utilized within a RESTful (representational state transfer) architecture and thus the create, read, update, and delete (CRUD) operations are hosted within an API in the directory.

To run locally, PostgreSQL must be installed, and the user must run `npm install` to install all of the required packages, and the following steps must be followed:

1. Connect the database by inputting its URL into the `.env` file according to the `.env.example`
2. Run `npx prisma db push`
3. Run `npx prisma db seed` to seed the project with the initial values.

Alternatively, the entire project is dockerized within a container utilizing Docker. In order to utilize the Docker container, the following should be done:

1. Run `docker compose up -d --build`
  - a. This command can be run within either the PromptLibrary, Backend, or Frontend directory in order to only build specific sections. Running the

command in the Backend directory will only build the backend database and server. Running it in the Frontend directory will only deploy the frontend website. Running the command in the PromptLibrary directory will deploy the entire project.

2. For the first time building the container, run

**a. `docker exec -it web npx prisma db push`**

**b. `docker exec -it web npx prisma db seed`**

Once the project is running locally or within the Docker container, simply navigate to localhost:8000 and the project can be interacted with in full. The backend can be accessed within localhost:8080.

As for the security of the project, this is handled mostly through Prisma ORM. Prisma is a parameterized query builder and is thus inherently protected against SQL injection attacks. Additionally, the database should be regularly exported through the PostgreSQL CLI (command line interface) and can be thus reimported in the case of server malfunction or ransomware attacks.

## 8.4 Development Description

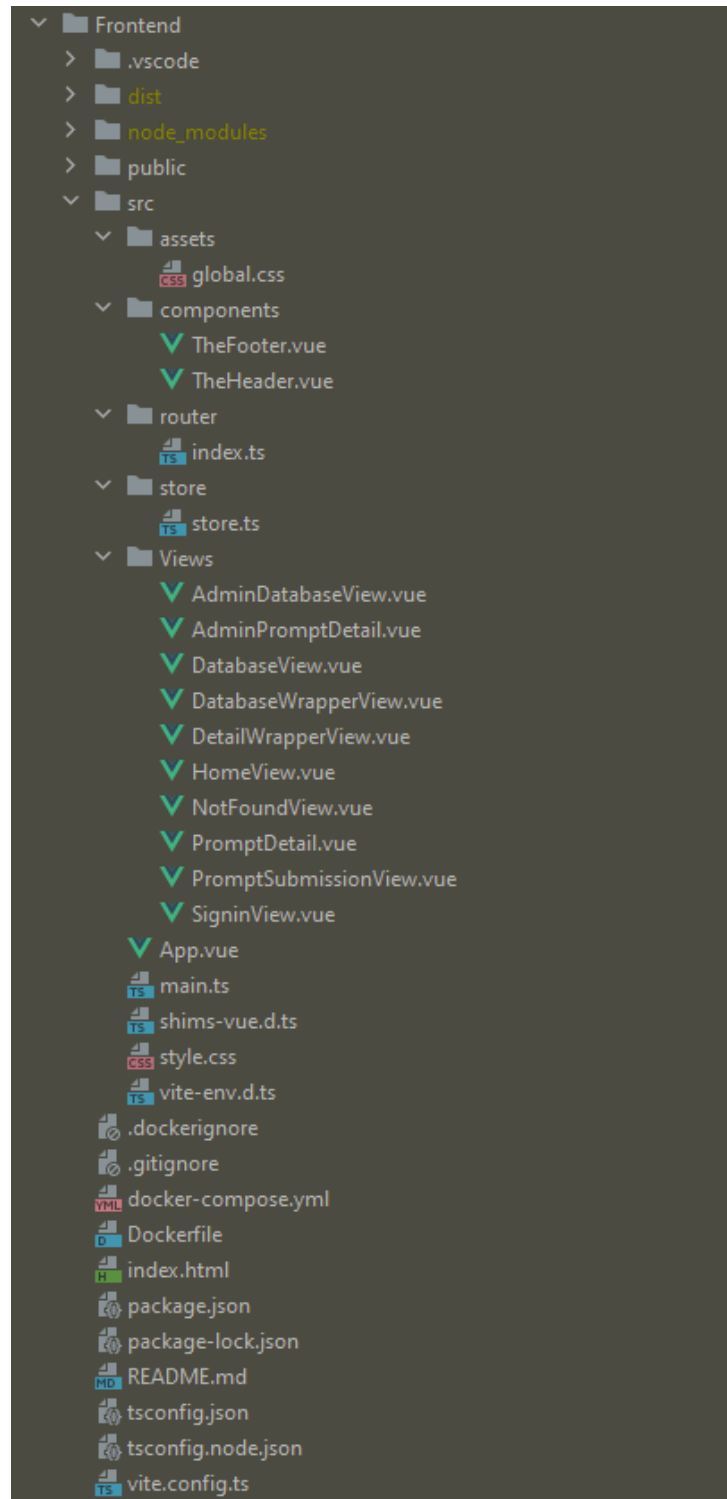


Figure 11: Project frontend directory structure

The project has two parts:

## Frontend

The frontend directory stores all the code for presenting the information stored in the database on the client's side as shown in Figure 11. Most of the code is stored in the **src** directory. Within the **src** directory, here are the description of the sub-directories:

- The **assets** directory stores the **global.css** which stores the global CSS styling configuration for the frontend code.
- The **components** directory stores the view part of the frontend, which includes the header and footer of the website.
- The **router** directory stores the routing information of the frontend information which links each web page together and also safeguards access to restricted views for both administrators and normal users.
- The **stores** directory stores the code which writes information of the project to local storage. In the current implementation, the state of checking if an administrator has logged in is stored in the local storage.
- The **Views** directory stores each individual frontend view. Each view is composed of the UI, the script that stores the function logic, and the styling which is specified in each view.

Overall, the frontend:

- Uses Vue 3
- Uses Firebase Authentication to support Google Authentication

- Uses routers to control and navigate users to different views
- Sends the respective API calls to the backend to update or retrieve information from the backend

## **Backend**

The Backend directory stores all of the code for creating the backend server and API (Application Programming Interface) as shown in Figure 12. The controllers directory stores the code for the fulltext search. The dist directory is created during the build process and contains all of the files converted to JavaScript. The lib directory contains a file for creating a Prisma singleton and the node\_modules directory stores the dependencies for the Backend. The Prisma directory stores the schema and seed file for actually utilizing Prisma. The routes directory stores all of the files for the end points within the API and serves as the bulk of the functionality. The Backend directory also contains various files for a variety of purposes, but the most notable are the dockerfiles for the container, server.ts for running the server, and the package.json files for the packages and scripts.

Overall, the backend:

- Used Prisma ORM
  - Create the schema and map relationships
  - CRUD (CREATE READ UPDATE and DELETE)
  - RESTful API (REpresentful State Transfer, Application Programming Interface)
- migration.sql, created by Prisma, creates the tables within PostgreSQL

- PostgreSQL stores all of the collected data

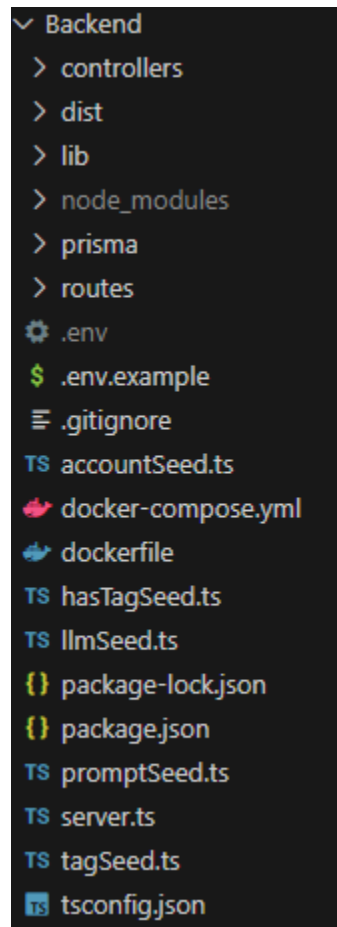


Figure 12: Project backend directory structure

## Docker

- Utilized Docker compose files defining and running multiple containers
  - Each container enabled a standardized development environment, safeguarding against compatibility issues within the team.
- Dockerfiles utilize the node base image for the backend server. The frontend dockerfile utilizes the nginx image for the frontend. The base images serve as a

foundation and environment for the dockerfile containing the project to be built upon.

- Both the frontend and backend have their own compose files, but there is also a project wide compose file for running both simultaneously.

## 9. Lessons Learned

Throughout the project, the team gained experience in full stack development and working as a team to produce a product. To maintain the quality desired by the client, the team participated in biweekly meetings with them and hosted work sessions to improve the product's quality and meet the client's expectations. Through these meetings, the team developed skills in professionalism and cultivated the ability to work cooperatively.

### **Problems and Solutions**

One of the major hurdles the group faced was a lack of experience with AWS Amplify. In order to host the website, the client recommended utilizing AWS Amplify and the toolsets it provides. However, the team had little to no experience with the software and had to learn it over the course of the project. Due to the complex nature of AWS Amplify and the depth of creating a fully functional website, fully utilizing its capabilities proved to be difficult. The first approach the team had was to utilize online resources, primarily the official documentation for AWS Amplify. After the research phase, the team had a foundation in the basics of utilizing AWS services. Another approach the team had was collaborating with other teams within the course. Several other teams, more specifically Team 5, also implemented their projects within AWS Amplify. Thus, working together and sharing information proved to be beneficial for all parties involved. Collaboration not only deepened the team's understanding, but also fostered a sense of community among the class. However, the problem of AWS Amplify was eventually solved by moving away from the Amplify structure. The lack of compatibility with the architecture,

our lack of experience with AWS software, and finally the cost of this technology led the team to eventually switch to a more traditional and cloudless approach with Docker. This allows for more flexibility in the deployment as well.

## 10. Timeline/Schedule

In order to ensure proper communication and understanding, the team met with the client in bi-weekly in-person meetings. Table 1 is the timeline of development with team tasks highlighted in yellow, frontend development in red, and backend in blue.

Completed tasks have a green box in the status column.

Task	Date	Status	Notes
Choose development details	February		Frameworks, communication tools, IDE
Draft implementation	February		Frontend design, database schema
Front: Milestone 1	February		Create the guest page along with headers and footers
Front: Milestone 2	February		Finalize web page theme/style
Back: Milestone 1	February		Create the database and fully implement simple POST and GET endpoints
Front: Milestone 3	March		Complete authentication page and prompt table

Front: Milestone 4	March		Create prompt submission page
Front: Milestone 5	March		Create authentication support and functionality
Front: Milestone 6	March		Implement support for searching and sorting
Back: Milestone 2	March		Implement backend authentication support
Back: Milestone 3	March		Implement prompt ratings and reviews
Back: Milestone 4	March		Create tag and text search functions
Back: Milestone 5	March		Create and finalize the export endpoint
Team: Milestone 1	March		Integrate and connect back and frontends
Team: Milestone 2	March		Dockerize the project
Team: Milestone 3	March		Deploy the project and start testing by March 22nd
Team: Milestone 4	April		Complete user testing
Team: Milestone 5	April		Deliver the final product by April 22

Table 1: Milestone Timeline and Schedule

# 11. Future Plans

As for subsequent semesters, future teams could be improving upon and maintaining the current project. This will include new features that users may request after the current product is delivered. As of now, the main features to be implemented or improved upon are:

- Authentication
- Incorporation of a rating system
- More efficient full-text searching
- Scaling up the database
- Direct integration of accounts
- Continuation of frontend accessibility improvements
- UTF-8 encoding

## 11.1 Authentication

Authentication for the project is currently functioning solely off of Firebase's API. For building a more secure token-based API, this process should pass JWT tokens into the database for security purposes as well as general data collections. "JWTs can be used to carry authentication information securely between microservices without the need to repeatedly verify credentials" [18].

## 11.2 Incorporation of a Rating System

In the future, the project should support a rating system in which each logged in user can give ratings to publicly visible prompts in the database. This will help promote users

to submit quality prompts and allow exceptional prompts in the database to take priority and be more visible in the database.

### 11.3 More Efficient Full-Text Searching

The current implementation of searching is a preliminary attempt for the functionality of searching. In short, it is a working function, but its performance can be improved for returning more accurate search results efficiently, and it needs better handling for edge cases. Additionally, the search should also take the rating system into account and allow searching and filtering based on prompt rating.

### 11.4 Scale up the Database

The current database is fine for its current scale of operations, but as future development continues, the scale of the database for the project should be prepared to handle larger operations. Future teams should consider implementing features such as data compression to save storage space, automatic database backup (the current database only supports manual database backup), and allow the database to handle multimedia content (images, PDF files, code files, etc.).

### 11.5 Direct Integration of Accounts

For future development, the project should enable more user-oriented features throughout the site such as favoriting posts, keep track of prompts each user submitted, check the approval status of their submitted prompts, and rate the prompts.

## 11.6 Continuation of Frontend Accessibility Improvements

As the development of the project continues, the styling and design of the project can be further improved in terms of web accessibility for users with disabilities (users with ADHD, color blindness, etc.) [15]. Certain elements should be improved in terms of styles as the project continues to expand in scale.

## 11.7 UTF-8 Encoding

Many example prompts were included in which various emojis and non-english characters were used. Through normal usage of the site, these special characters caused no issues. However, when dumping the database, a task that should be performed regularly, these characters would not translate properly into the frontend after being imported. This should be investigated and a consistent encoding should be used to support these characters.

## 12. Acknowledgements

Client: Dr. Yinlin Chen

Email: [ylchen@vt.edu](mailto:ylchen@vt.edu)



Dr. Yinlin Chen is the Assistant Director and Assistant Professor for the Center for Digital Research and Scholarship, and holds a B.S. and M.S. in Computer Science from National Tsing Hua University and a Ph.D. in Computer Science & Applications from Virginia Tech.

Professor and mentor: Dr. Edward A. Fox

Email: [fox@vt.edu](mailto:fox@vt.edu)



Dr. Edward Fox is a professor for the Department of Computer Science at Virginia Tech and the Director of the Digital Library Research Laboratory. He holds a B.S. in Electrical Engineering (Computer Science option) from M.I.T. and an M.S. and Ph.D. in Computer Science from Cornell University.

## 13. References

[1] E. You, "Vue.js," Vue.js, 2014. Available: <https://vuejs.org/>. [Accessed: Apr. 30, 2024]

[2] Firebase developers, "Authenticate Using Google with JavaScript," Firebase, Apr. 16, 2024. Available: <https://firebase.google.com/docs/auth/web/google-signin>.

[Accessed: Apr. 30, 2024]

[3] Google Identity developers, "Using OAuth 2.0 to Access Google APIs | Authorization," Google Identity, 2024. Available:

<https://developers.google.com/identity/protocols/oauth2>. [Accessed: Apr 30, 2024]

[4] OpenAI, "ChatGPT," ChatGPT, 2024. Available: <https://openai.com/chatgpt>.

[Accessed: Apr. 30, 2024]

[5] Google Gemini Team, "Gemini," Gemini, 2024. Available: <https://gemini.google.com>.

[Accessed: Apr 30, 2024]

[6] Prisma Data, "Prisma Documentation," Prisma, 2024. Available:

<https://www.prisma.io/docs/>. [Accessed: Apr 30, 2024]

[7] The PostgreSQL Global Development Group, "PostgreSQL 16.2 Documentation," PostgreSQL, Feb. 08, 2024. Available: <https://www.postgresql.org/docs/16/index.html>.

[Accessed: Apr 30, 2024]

[8] Amazon Web Services, "Amazon Cognito Documentation," Amazon Web Services, 2024. Available: <https://docs.aws.amazon.com/cognito/>. [Accessed: Apr 30, 2024]

[9] Amazon Web Services, "AWS Amplify Documentation," Amazon Web Services, 2024. Available: <https://docs.aws.amazon.com/amplify/>. [Accessed: Apr 30, 2024]

[10] OpenJS Foundation, "Node.js v21.7.3 Documentation," Node.js, 2024. Available: <https://nodejs.org/docs/latest/api/>. [Accessed: Apr. 30, 2024]

[11] E. You, "Vite," Getting Started, 2024. Available: <https://vitejs.dev/guide>. [Accessed: Apr. 16, 2024]

[12] JetBrains, "Getting started," IntelliJ IDEA, 2024. Available: <https://www.jetbrains.com/help/idea/getting-started.html>. [Accessed: Apr. 30, 2024]

[13] J. Watmore, "Vue + Fetch - HTTP POST Request Examples," Jason Watmore's Blog, 2020. Available: <https://jasonwatmore.com/post/2020/04/30/vue-fetch-http-post-request-examples>. [Accessed: Apr. 30, 2024]

[14] Anthropic developers, "Prompt library," Anthropic, 2024. Available: <https://docs.anthropic.com/claude/page/prompts>. [Accessed: Apr. 30, 2024]

[15] Virginia Polytechnic Institute and State University, "Web Accessibility," Accessible Technologies, 2024. Available: [https://assist.vt.edu/content/assist\\_vt\\_edu/en/web-accessibility.html](https://assist.vt.edu/content/assist_vt_edu/en/web-accessibility.html). [Accessed: Apr. 30, 2024]

[16] Docker, "Docker overview," Docker Documentation, Apr. 17, 2024. Available: <https://docs.docker.com/get-started/overview/>. [Accessed: Apr. 30, 2024]

[17] The GraphQL Foundation, "Introduction to GraphQL," GraphQL, 2024. Available:  
<https://graphql.org/learn/>. [Accessed: Apr. 30, 2024]

[18] Sanchhaya Education, "Designing Authentication System | System Design,"  
GeeksForGeeks, Jan. 31, 2024. Available:  
<https://www.geeksforgeeks.org/designing-authentication-system-system-design/>.  
[Accessed: Apr. 30, 2024]

[19] Postman, "Postman API Platform," Postman, 2024. Available:  
<https://www.postman.com/>. [Accessed: Apr. 30, 2024]