Interactive AI-Powered Database Query Web App

ChatDB 42: Project Final Report

Rongzi Xie, Yihao Wang, Yixiu Wang

Github Link:
https://github.com/Yihao-8bit/Dsci551-Project

Introduction

Our goal in this project was to apply AI techniques and database knowledge acquired during the semester to develop an application capable of handling basic queries for both SQL and NoSQL databases. The application can translate natural language requests into executable queries, perform real-time updates, and present database details to the user.

Throughout the project, we reinforced our understanding of database querying, front-end and back-end development, and learned how to integrate AI techniques into data analytics applications.

Planned Implementation (From Project Proposal)

AI Chatbot

We planned to use free & open source text generation models to convert user's queries to SQL or Nosql commands. For now, one of our options is to use python to implement the Meta Llama model, and we also found some useful resources to help us complete the implementation:https://github.com/abetlen/llama-cpp-python.

Due to the limitation of hardware, we also consider small and fast models, converting the user's query shouldn't be difficult for AI agents, and here are some small models we found from hugging face that might be useful.

We consider the new DeepSeek & Llama AI model.

https://huggingface.co/deepseek-ai/DeepSeek-R1-Distill-Llama-70B

Also, Mistral models seems to be extremely small but outperforms other model in many tests, so we also consider using this one:https://mistral.ai/en/news/announcing-mistral-7b

We want to compare the performance of each model based on their difficulty level of implementation, ability to convert user queries and efficiency. At last we chose the deepseek model due to its cheap price and extraordinary understanding of natural languages.

SQL&NoSQL Database Exploration and Operation

Our AI chatbot needs to convert natural language into database queries, so the choice of database should depend on the type of data to be stored and queried, so far according to the lookup, MongoDB, Hadoop, and Spark may be the potential databases we will select

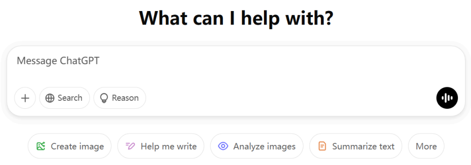
Meanwhile, we also learned that a vector database called Pinecone is mainly used for similarity search, embedded search, and large-scale AI applications, and can provide low-latency, high-performance vector retrieval. It can improve the performance of our chatbot in the direction of semantic query enhancement, storing and optimizing query history, and natural language to SQL conversion optimization.

- Database Construction: choose MySQL or MongoDB, choose whether to introduce Pinecone for vector data processing, and design the data table/document structure.
- Develop Backend API: Create/query interface using django paths, and return SQL/NoSQL queries. Also, explore whether using the Pinecone database can improve the accuracy and speed of responses.
- Integrate AI Models: Implement natural language to SQL conversion with DeepSeek.
- Build Links to Front-end Pages: Design links to front-end pages and implement features required by front-end pages. Implement login page, SQL query page and non-SQL query page
- Deployment and Optimization: Containerization with Docker, Redis caching of queries, monitoring database performance.

Frontend UI Design

We plan to design a frontend page similar to the ChatGPT style, aiming to create a clean, user-friendly, and highly interactive interface.

For the main search area, we intend to design a large text input field where users can enter natural language queries. The input field will be simple, clear, and always visible, with placeholder text such as "Ask me about information in the database..." to guide users and encourage them to ask questions or give commands. When clicked, these buttons will open a brief form where users can fill in the relevant data. An example is shown below.



Each query and its response will be displayed in a table like format, clearly distinguishing between user input and system responses. To enhance interactivity with users, we plan to design dynamic responses. When users enter queries, the interface will display a "loading" message to inform users that the system is processing the query. After the query is entered, the system will immediately process the natural language and display the results. If there is an error or any issues with the query, the system will provide clear and friendly error messages, similar to ChatGPT's error prompts, allowing users to correct the query. If the query results are too long, the chat window should smoothly scroll, and users can click to load more results, similar to how ChatGPT loads more messages.

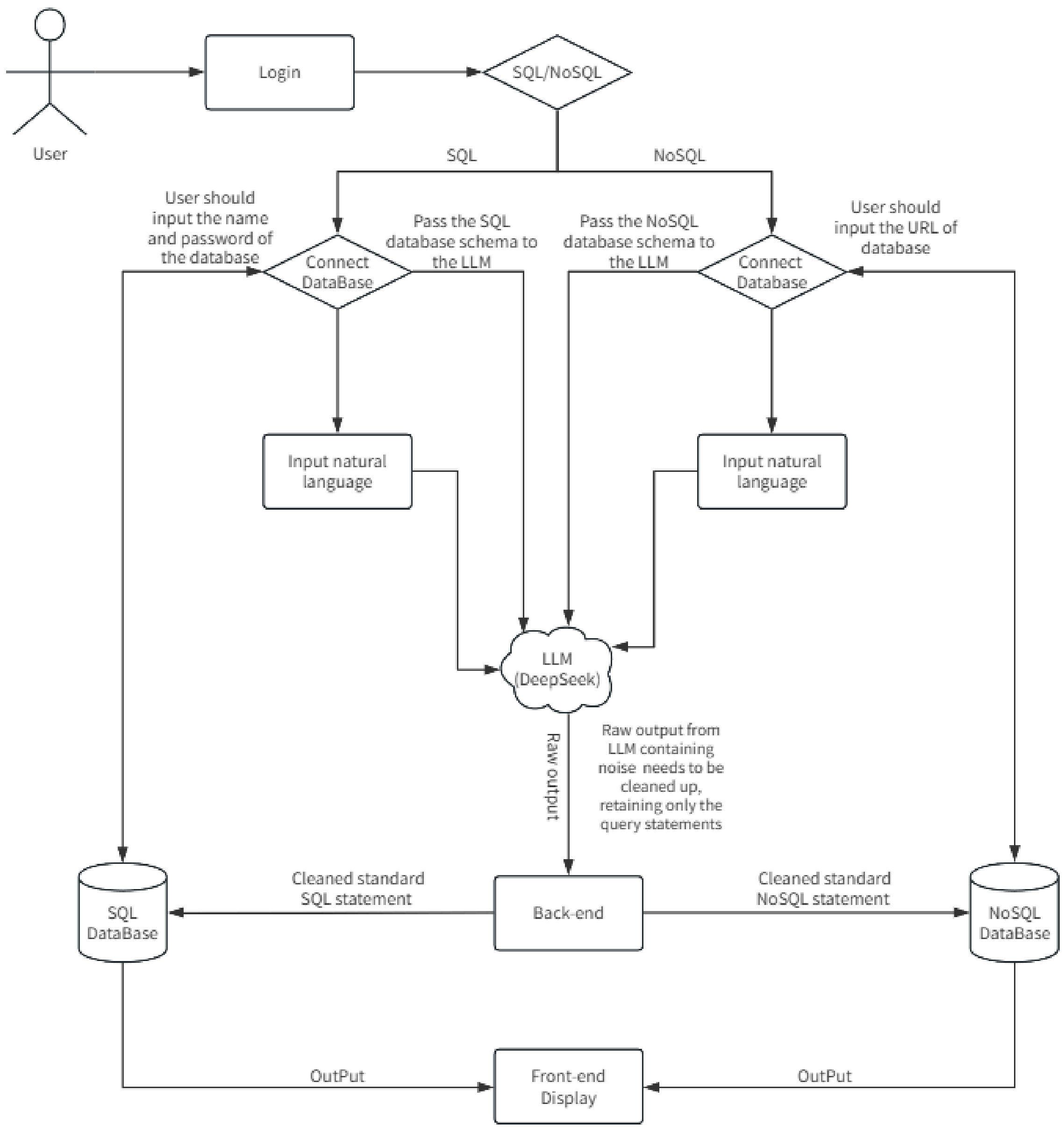
Test

We plan to implement some real world dataset that has connections to each other to test our model's performance on join, search, updating, deletion and other common database query,

We planned to use a database from our homework and lab, which surely meet the requirements.

For MongoDB, we used the actor and film dataset from homework4, and for mysql database, we used the customer order dataset.

Architecture Design (Flow Diagram and its description)


User
Login
SQL/NoSQL
SQL
NoSQL
User should input the name and password of the database
Connect DataBase
Pass the SQL database schema to the LLM
Pass the NoSQL database schema to the LLM
Connect Database
User should input the URL of database
Input natural language
Input natural language
LLM (DeepSeek)
Raw output
Raw output from LLM containing noise needs to be cleaned up, retaining only the query statements
SQL DataBase
Cleaned standard SQL statement
Back-end
Cleaned standard NoSQL statement
NoSQL DataBase
OutPut
Front-end Display
OutPut

Implementation

1. Functionalities

Front End-UI:

The front-end interface is built using HTML, CSS, and a bit of Vue syntax to create an easy-to-use and developer-friendly user interface. The page implements login, registration, SQL and NoSQL queries and integrates the functionality into a switchable template. Users can switch between MySQL and NoSQL modes logically using the clearly positioned toggle buttons at the top of the interface, and the whole interface is simple and modular, with a unified and clean layout style, and a logo drawn for our project using LLM.

The login and registration forms have an intuitive layout that provides real-time feedback, and are able to handle error messages that cannot be handled by the back-end, and are logically self-explanatory, and give information tips on the front-end. When registration is successful or login fails, a message pops up in the center of the screen to notify the user and avoid errors, thus enhancing the overall interactive experience.

For the query interface, each mode provides the function of selecting the linking server, allowing users to input and display relevant queries, and adapting the output format for different cases.

In the query panel, users can：

1. Enter a database and link to it.
2. Enter a natural language query, and the backend will translate the natural language query into a corresponding query statement.
3. Submit queries and view results in a scrollable, dynamic form.
4. View explicit information about whether the query succeeded, failed, or data was not found.

From login and registration to query submission, user actions are reflected on the same page as much as possible to minimize page reloads, thus creating a smooth and responsive user interface.

Front-End UX:

Users can easily switch between NoSQL and SQL databases using a toggle button on the web interface. The entire application runs on a single page, making it intuitive and convenient to operate. Additionally, each user can register with a unique account and password, ensuring privacy and secure access to their data.

Back-End – Query Conversion:

Our chatbot can convert natural language queries into both SQL and NoSQL commands by first retrieving the structure of the connected database. This is done through a custom function that scans the database to extract table and column names (for SQL) or key structures (for NoSQL), enabling accurate query generation.

The application supports both MySQL and MongoDB connections directly through the web interface, allowing users to switch accounts or databases seamlessly.

Back-End – Error Handling:

The application is capable of handling invalid or non-existent queries. For instance, if a user requests data for a record that doesn't exist or tries to delete a non-existent entry, the chatbot will generate a clear, user-friendly message indicating that no matching record was found in the database.

2. Tech Stack

Application Structure

```
DSCI551_PROJECT/
├── dsci551_project/
│   ├── __init__.py
│   ├── asgi.py
│   ├── settings.py
│   ├── urls.py
│   ├── wsgi.py
│   └── manage.py
└── dsci551/
    ├── migrations/
    ├── static/
    │   └── images/
    │       └── logo.png
    ├── templates/
    │   ├── login.html
    │   ├── register.html
    │   ├── select_database.html
    │   ├── query_form.html
    │   ├── query_history.html
    │   ├── nosql_query.html
    │   └── dj_home.html
    ├── __init__.py
    ├── admin.py
    ├── apps.py
    ├── models.py
    ├── tests.py
    └── views.py
```

Package Used

Django==5.0.3

pymongo==4.12.1

PyMySQL==1.1.0

Requests==2.32.3

Front End

Templating: Django Templates (HTML5, CSS3, JavaScript)

Back End

Framework: Django (optionally with Django REST Framework)

ORM: MySQL via Django ORM, MongoDB via PyMongo

LLM Integration: DeepSeek API

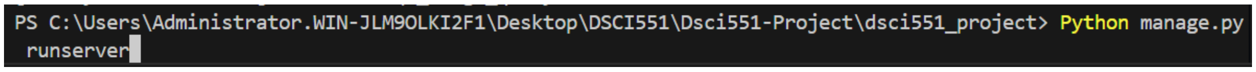
Error Handling

Catch and report LLM timeouts or failures

Handle database execution errors with clear JSON error responses

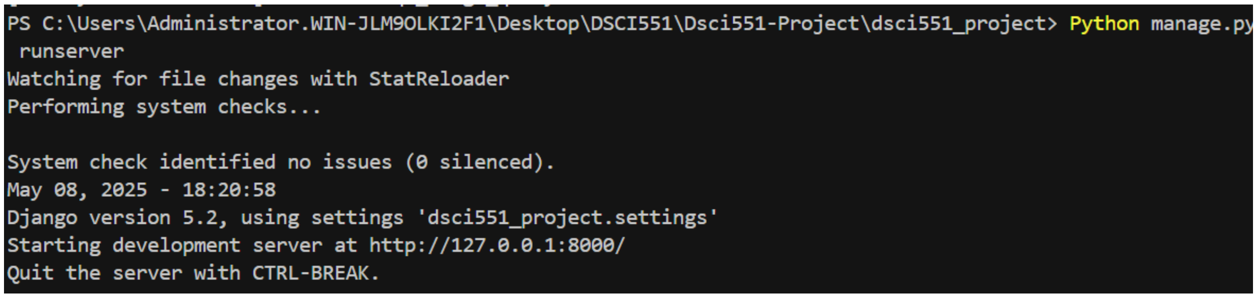
3. Implementation Screenshots

More detailed instructions are in the github readme.



```
PS C:\Users\Administrator.WIN-JLM9OLKI2F1\Desktop\DSCI551\Dsci551-Project\dsci551_project> Python manage.py
 runserver
```

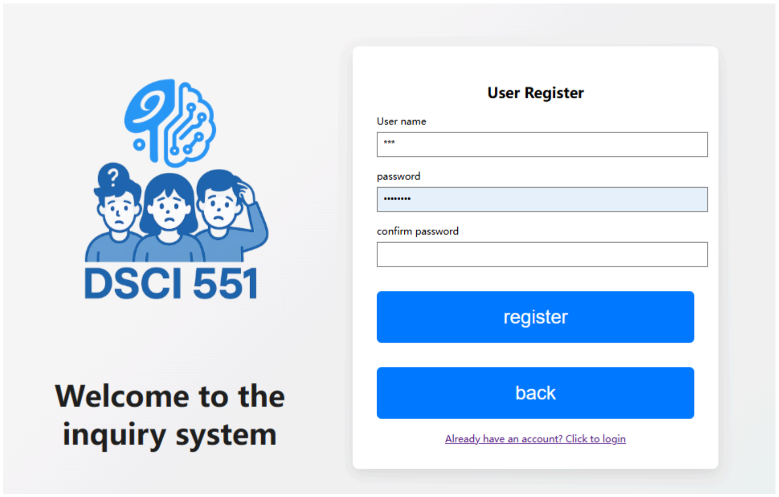
Run the above code under dsci551_project, and wait for the server to be started.



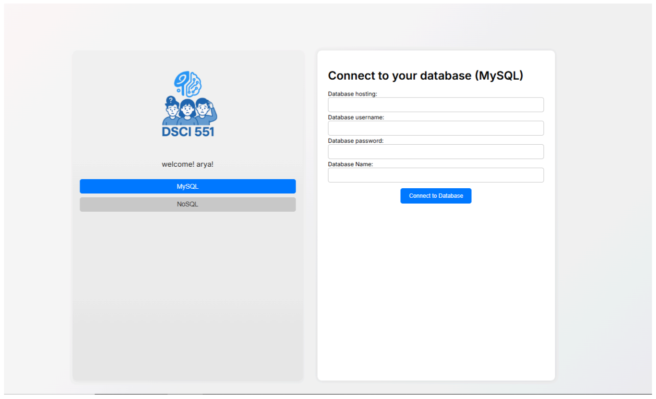
```
PS C:\Users\Administrator.WIN-JLM9OLKI2F1\Desktop\DSCI551\Dsci551-Project\dsci551_project> Python manage.py
 runserver
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).
May 08, 2025 - 18:20:58
Django version 5.2, using settings 'dsci551_project.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
```

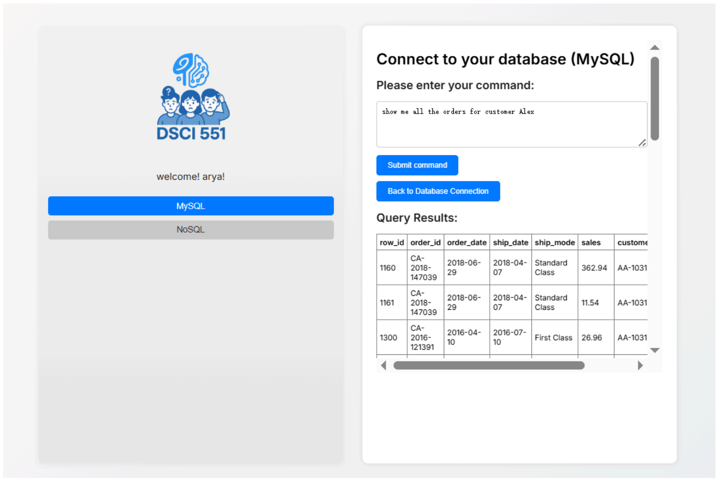
Click to the localhost link to access our application.



After registration, log in with your username and password.



For the SQL part, make sure you enter the correct username and password before clicking the connect to database button.



Connect to your database (MySQL)

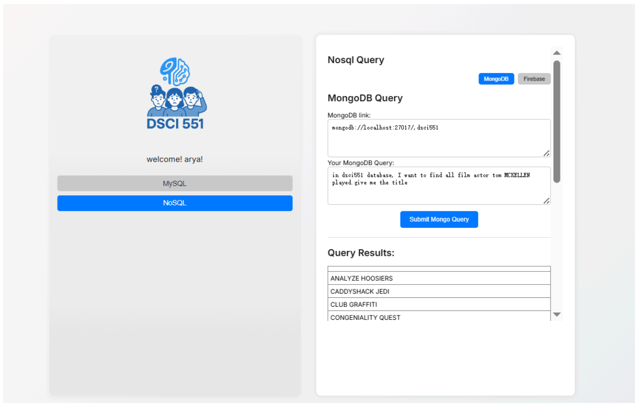
Please enter your command:

```
show me all the orders for customer Alex
```

Query Results:

row_id	order_id	order_date	ship_date	ship_mode	sales	custome
1160	CA-2018-147039	2018-06-29	2018-04-07	Standard Class	362.94	AA-1031
1161	CA-2018-147039	2018-06-29	2018-04-07	Standard Class	11.54	AA-1031
1300	CA-2016-121391	2016-04-10	2016-07-10	First Class	26.96	AA-1031

After that you can enter your query.



Nosql Query

MongoDB Query

MongoDB link:

```
mongodb://localhost:27017/,dsci551
```

Your MongoDB Query:

```
in dsci551 database, I want to find all film actor tom MCKELLEN played.give me the title
```

Query Results:

ANALYZE HOOSIERS
CADDYSHACK JEDI
CLUB GRAFFITI
CONGENIALITY QUEST

Same thing for the mongoDB part, please enter the link and your database name, separated by a comma. For example: “mongodb://localhost:27017/,dsci551”, and make sure there is nothing else, no empty space and newline character

During the process, please do not hit the website’s back and forward button on the upper left, please use our “back” and “logout” button to go back to the last9 page.

Learning Outcomes

1. Front-end design:

Through this experience, we were able to build a responsive page that could interface with the backend in a short period of time using HTML, CSS, and Vue.js within the Django framework.

For the technical part, in addition to using the skills we already knew how to use previously. We learned how to communicate between Django views and Vue components via JSON responses, as well as how to properly build front-end logic and exception handling to run in sync with asynchronous APIs.

In terms of teamwork, we use git for code control and versioning. Since everyone's code style is different, and the front-end interfaces reserved for the back-end are also different, the workload of integrating the back-end code and accessing the front-end becomes larger, which exercises and improves our team's coding ability and communication skills. Meanwhile, using git for code control can help us carry out simple and efficient code version control, merging and updating code modifications, which can be more convenient for programming and save a lot of ineffective communication.

2. AI chatbot

   The AI chatbot is the most critical component of our project, as it determines whether query conversion succeeds. Through this process, we learned how to integrate a large language model (LLM) into our application using API keys and how to guide it with customized prompts to perform specific tasks.

   One of the most valuable takeaways was learning how to craft effective prompts that provide the AI with just enough database context to complete tasks—especially when working with a limited version of the model due to budget constraints. Unlike using full-featured models through platforms like ChatGPT or DeepSeek, we had to be extremely concise and precise in our instructions to avoid unnecessary output and ensure the AI returned clean, executable code.

   At times, the AI produced incorrect results or even refused to generate code, which required continuous refinement of our prompts to achieve the reliability and consistency we needed.

3. MySQL and MongoDB

   We learned how to craft effective prompts to guide the chatbot in establishing connections to databases and enabling generative AI to understand database structures. Throughout this process, we deepened our understanding of database architecture, along with tools such as PyMongo, PySQL, and SQLAlchemy.

In the early stages of the project, we used Django to connect to an SQL database. However, we eventually decided to embed database connectivity directly into the application's functionality. This allows users to connect to their own databases and work with any dataset they choose. Our system, chatDB, dynamically interprets the structure of the connected dataset and generates appropriate responses.

While this design significantly increased the complexity of the project—and introduced the risk of AI instability—we carefully structured our code to ensure robust database connections, accurate parsing of dataset schemas for the AI, and well-defined prompts that guide the AI in executing the correct query logic.

For error handling, we also learned to distinguish between different output types from SQL and NoSQL queries and developed separate handling strategies for each.

Challenges Faced

Connections to Database:

Initially, connecting to an SQL database created a lot of confusion. We had to modify Django settings to ensure it connected to our local databases, and each time a team member pulled the code from GitHub, they had to manually update the username and password in the settings to run the application and cannot switch between accounts and database. To solve this, we switched to using PySQL and PyMongo for database connections and added a front-end interface where users can input their own database credentials. This made the system far more user-friendly and flexible.

AI's Instability:

Since we wanted our AI to support arbitrary databases, it needed to be robust across different data types without relying on prompts tailored to a specific database. Initially, we let the AI infer everything on its own, but the results were poor. The AI often ignored instructions to learn the database structure and started hallucinating tables or fields that didn't exist—an issue common with large language models.

To address this, we wrote code to automatically extract the actual database schema, including table names, column names, and indexes. This structured information is then passed to the AI as context, allowing it to generate more accurate queries. We found that SQL databases produced more stable results, while MongoDB (NoSQL) posed greater challenges due to its flexible and nested data structures. As a result, we had to design more detailed and precise prompts when working with MongoDB to avoid errors.

Integrity of Application Structure:

As a three-person team with each member responsible for different components—frontend, SQL, and NoSQL—miscommunication occasionally led to confusion about the application's current version, integration between the frontend and backend, and discrepancies between GitHub branches. At times, an accidental push of an outdated or incompatible version disrupted the entire system.

However, through a month of collaboration, communication, and iteration, we developed effective strategies to stay aligned. Frequent check-ins, clear documentation, and consistent confirmation after each pull, push, and code update helped us maintain synchronization across the project. This experience greatly improved our teamwork and taught us the importance of coordination in collaborative development.

Individual Contribution

Yihao Wang: In charge of transforming user natural-language queries into optimized MySQL statements, and integrating the large-language-model interface to enable seamless end-to-end query execution and result retrieval.

Yixiu Wang: Responsible for front-end page design, front-end page construction, front-end and back-end code integration, and code version control.

Rongzi Xie: Responsible for managing homework submissions, coordinating team meetings, and developing the code for converting natural language queries into MongoDB commands.

Conclusion

This project successfully demonstrated the integration of AI and database technologies to create a user-friendly web application capable of translating natural language into structured SQL and NoSQL queries. Through challenges such as prompt design, database connectivity, and AI reliability, our team gained valuable experience in full-stack development, natural language processing, and collaborative software engineering. Although there were features we could not fully implement due to time constraints, the core functionality—flexible database access, intelligent query conversion, and responsive user interaction—was achieved. Moving forward, this project lays a solid foundation for future improvements such as multi-device compatibility, enhanced database support, and more robust AI behavior. Overall, it was a highly rewarding experience that strengthened our technical skills and teamwork.

Future Scope

There were a few features we believed would significantly improve user experience but were not completed due to time constraints:

1. Query History:

   Adding a feature to store users' query history would allow them to review previous outputs, reducing the need to repeat questions to the chatbot. It would also help users better understand their interactions with the dataset, including tracking operations like updates or deletions—offering insight into how the dataset has changed over time.

2. Firebase Query Support:

   As part of our goal to support NoSQL databases, we planned to implement natural language query conversion for Firebase by translating user input into cURL-based queries. While we began testing this functionality, we were unable to complete the implementation and validation process.

3. Mobile-Friendly Frontend:

   Our application is web-based and currently optimized for desktop screens. While the front-end design works well on computers, it may be less convenient on mobile devices—especially due to the sidebar navigation for selecting SQL or NoSQL query modes. In the future, adapting the layout responsively for mobile screens would greatly enhance usability and accessibility across different devices.