DSCI 551 - Project Midterm Progress Report

Team Responsibilities:

Wenjia Wang:

I would pay my effort to cooperate with teammates in developing a web-based application with Flask including tasks such as setting up the framework, defining routes, creating templates, managing requests and responses, and connecting to a database. I will facilitate teamwork by organizing projects into directories, defining routes using decorators, and connecting Flask to a database of choice, such as MySQL or MongoDB.

Zhongtian Ye:

My task mainly focuses on designing the user interface in order to help database managers and end-users easily access the data in the database. I will use Flask, CSS and Javascript to design an easily implemented and high efficiency user interface, create highly interactive visualizations, real time dashboards and sophisticated workflow apps. I would also choose the dataset that our team will be working on, which will be fit with the database that our team will construct and web application functions that we will develop.

Jianfei Xiao:

As the team leader, I will use my skills and experience in developing Convolutional Neural Network-based systems, as well as skills in programming languages such as Java and Python to coordinate the work of the two team members throughout the project and ensure that the project is progressing steadily. In terms of breakdown, I will be leading the back-end development efforts and focusing on the integration of Flask and database management, overseeing the design and implementation of Flask routing, aligning with the distributed database architecture, and designing and implementing user interfaces for database administrators and end-users.

Implementation Questions:

- Which database are you using?

  To date, our project has integrated MySQL, Firebase, and MongoDB. Given the project's requirements for distributed databases capable of managing diverse data types, we are focused on developing a system that incorporates both SQL and NoSQL databases, leveraging the unique strengths of each, we remain open to the possibility of integrating additional databases.

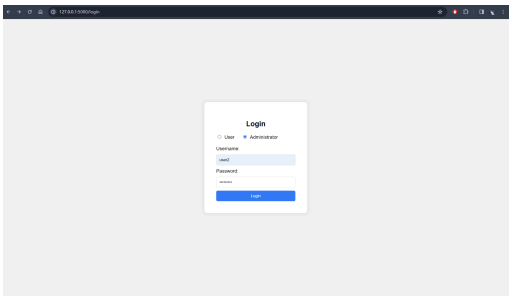
- What is the approach for Distributed scaling of data?

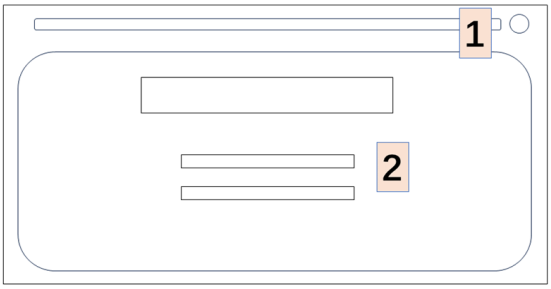
  On the 2nd page of our implemented website, we give users a choice between MySQL for relational data needs and MongoDB or Firebase for NoSQL requirements. Each database shines with different data types: SQL for structured queries and transactions and NoSQL for scalability and flexibility with unstructured data. In the future, we may add text guidance on our website to help the user pick the right one based on their project's unique requirements, making the decision process straightforward and informed.

- Which application did you choose to implement?

  We've developed a comprehensive login system and a user-friendly dashboard that enables database querying, along with full support for various CRUD (Create, Read, Update, Delete) operations. In the future, we may enrich our platform's database interaction capabilities and incorporate advanced data visualization frameworks to elevate the user experience on our website.

Planned Implementation:



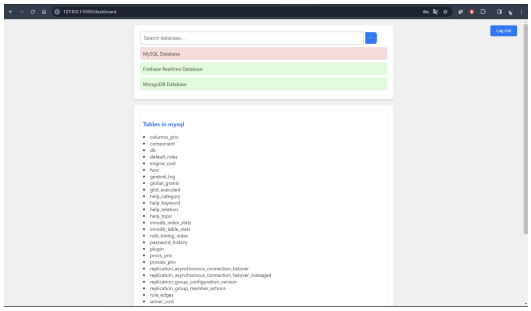


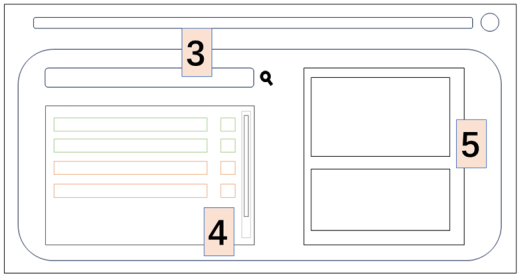
- **Tag 1&2**

  We plan to develop a web-application with Chrome on Windows OS. We selected Flask as our backend framework to ensure the seamless development process.

  As mentioned in the project requirements, the quantity of the dataset has a substantial size. Thus, implementing a Log in System is essential to manage our user permissions effectively. Users will be categorized into “normal users’’ with read-only access and "administrators" with rights to modify and delete. After the user type in the username and the password, we will

check the user permission as we will maintain a User table in a SQL database. Administrators will be provided more rights such as “Delete” or “Modify” the tables in the databases.



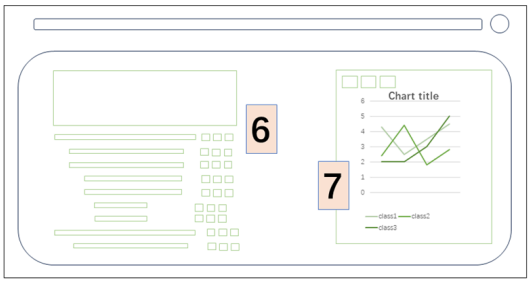


- **Tag 3**

  After logging in to this system, the web will redirect to this interface. The search bar **has the function to look for databases available to read and edit**. The search bar also has a likelihood search feature, which means **the user can type in part of database names without specifying the full name of the database, and uppercase or lowercase letters also do not matter**. Surely the search bar supports the exact search function. Search results will display databases matching the query criteria.

- **Tag 4&5**

  The List of the databases. We may use different colors to distinguish between the SQL(RED) databases and the NoSQL (GREEN) databases. **The user may click on the database that he/she wants to access (clicking either the red bars or green bars)**, then extra information of the database will be shown in figure 5 to offer insights into the data's structure and scale. This information includes the name of all the databases stored in the selected database (MySQL, MongoDB). **Moreover, the user may click on a specific database to check all the tables within that database. The user then may click on a specific table, the page will redirect to the next page to be ready for more interactions from the user and the database.**

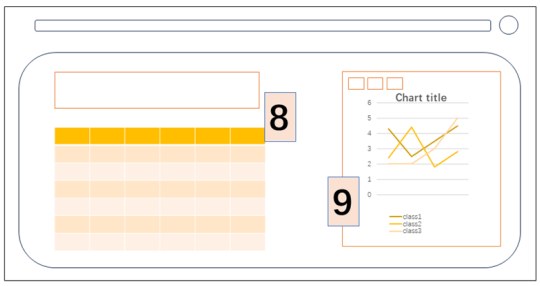


- **Tag 6**

Here we use Green to identify the selected NoSQL database. The JSON data will show line by line, administrators will have access to additional functionality, including options to modify or delete records, highlighted by three buttons alongside each

- **Tag 7**

  We also want to provide some visualization charts and graphs to give the users some intuitive feelings of the data. Thus, we may integrate some data visualization frameworks into the website to generate the figures.



- **Tag 8**

  Similarly, here we use Red to identify the selected SQL database. The tables displayed sequentially, with administrative functions available for record management.

- **Tag 9**

  Same as Tag 7 but possibly develop different types of visual representations to improve user interaction with SQL database information.

Status of the project:

We have implemented a log-in system to manage user permissions (tag 1-2). The user may run login_v2.py and navigate to http://127.0.0.1:5000/login to see the log-in page.

To store the users' information, we created a database in MySQL called users_info and created a table called users_table with columns id (as primary key), user_name, hashed_password and type.

The actual users' information will be stored automatically into the table by running users_info.py once you put a user's information including id, user_name, password, and type (normal or administrator) into the users_info.py. Password hashing will automatically run to improve security and to make sure that the login process can successfully run.

Then the user may type his/her username and password and select his/her user type (either normal user or administrator). By clicking the blue "Login" button, the system will check if the user is valid to reach the dashboard. If a user typed the correct username, password and selected the associated user type, then it will redirect to the dashboard page. If the user

typed incorrect password or username, then the error message is “Invalid username or password”. If the user typed the correct username and password but selected the wrong user type, then the error message is “User type does not match”.

We also implemented a dashboard (tag 3-5) to let the users choose which databases and tables he/she may want to query on. Users can use the search bar on the top to look for SQL or NoSQL databases such as MySQL, Firebase and MongoDB. The search function supports both exact and fuzzy search. The user may type in keywords such as “Fire”, “fire” or “my” to search related databases. As we stated in planned implementation, SQL databases are shown in red and NoSQL databases are shown in green. If the user typed a non-exist database then it says “No valid database selected!” Right now, we only implemented the function to interact with the local MySQL server. The user may click on MySQL Database, then all the MySQL databases on the local MySQL server will be listed in the box below. The user may also click on a specific database in the list, and then all the tables in that database will show as a list. Additionally, there is a logout button on the dashboard. The user may click it to log out from the dashboard and the page will redirect to the login page.

Next, we will try to implement more functions to interact with the database. Basic queries such as selecting a specific database and filter with certain conditions will be implemented. We will try to visualize the table structure once it is selected to improve user experience. For administrators, they will have the right to modify the data in the databases. But for normal users, they only have the basic functions to make queries on the database. After implementing these functions with MySQL, we will implement the functions to interact with MongoDB. If time permits, we will also implement these interactions with Firebase.

Challenges:

We face some restrictions and difficulties with our current static HTML page. The static page is unable to automatically display updated data from the database without requiring a manual refresh is one notable problem. This implies that unless the webpage is manually updated, users might only view old data, after they change and delete some information from the database. Furthermore, we are unable to achieve features like real-time notification. We must manually change the HTML files in order to add new features or provide updated information.

These problems are addressed with JavaScript, which enables dynamic interaction with the Document Object Model (DOM). JavaScript allows us to change the webpage's elements and attributes in real-time without requiring a complete page reload. Some helpful functions like querySelector, queryElementsByClassName, getElementById, and querySelectorAll could be used to select particular HTML elements and apply dynamic updates.By modifying the DOM tree, we may accomplish dynamic updates by incorporating JavaScript into our static HTML page. This makes it possible for us to easily add elements and real-time data, improving usability.

Timeline:

From 2.4 to 4.17, 10 WEEKS

- Front End:
    - Vue.js, d3.js
        - Visualize data in multiple ways
    - Expected Duration: 2 week
- Database:
    - Management:
        - Insert different types of test data into multiple databases
    - Contemplate the utilization of various databases for the storage and processing of specific data.
    - Expected Duration: 1 weeks
- Back End:
    - Project Functionality:
        - Develop methods that enable users to Create, Read, Update, and Delete (CRUD) the data
- Expected Duration: 2 weeks

Timeline specification:

Our project timeline has evolved based on practical insights gained during the development process. Originally, we structured our timeline by sequentially focusing on the front-end, database, and back-end development. However, we've discovered a more efficient strategy is to amalgamate all three components simultaneously and construct the website page by page.

To date, we have successfully crafted three web pages for our prototype and are actively working to finalize the remaining pages. Despite our progress, we have encountered three significant challenges:

For the front-end side, our data visualization needs improvement. The current representation does not convey information intuitively, which requires learning and implementing advanced data visualization techniques. We aim to create dynamic, insightful visualizations that will allow users to interpret data more effectively. These development processes may take approximately one week.

Secondly, while our current testing utilizes databases with limited records, we must prepare for scenarios involving larger datasets to ensure our distributed database infrastructure can handle big data workloads. This will involve rigorous testing with substantial data volumes to validate our system's robustness.

On the back-end side, there are a few minor bugs that require attention. In addition, while our schedule allows, we remain open to the possibility of integrating additional databases. Our overarching design philosophy prioritizes maintainability and scalability, ensuring that future updates and enhancements can be integrated efficiently.