DSCI551: Final Report

Meowdom: A Resource Exchange and Second-hand Trading Platform

Kexin Sheng, Liujia Yu, Jiawen Zhang

Apr 26 2024

Introduction

In this big data era, the ability to store and manage large amounts of data matters in business and operations. Data storage and management efficiency are crucial to those operations, earning an edge for business across sectors. We are aiming to build distributed databases to enhance organizing data in daily usage, in particular, second-hand trading information. Second-hand trading typically acquires different types of information from both sellers and buyers which engage vast and varied personal information as well as product data. Given the complexity and great amount of user data, a robust database and a website for data collecting and searching become necessary. In this report, the details of our second-hand trading website design and database system will be thoroughly explained, which includes front-end and back-end implementation and data manager system operation.

Planned Implementation

Considering our motivation and purpose, we brainstormed and formed three different topics in mind: a second-hand trading website, a course information searching website, and a pet boarding website. As a result, we plan to combine the secondhand trading and pet boarding websites into one, as they share similarities in terms of user interactions. Specifically, they cater to two distinct user groups with different needs, such as sellers and buyers in the secondhand items marketplace. This project aims to help and empower users to access comprehensive information tailored to their specific needs, contributing to a more streamlined lifestyle.

For the details of our implementation, we are going to build a distributed database system that can save information on items that belong to the users and store it in different databases according to their hash value. We decided to use SQL as our database language since it is easy to extract useful information and it can represent the relationship between tables clearly. For the secondhand transaction database, we would separate our data into three tables to store information about users, items, and orders respectively. Specifically, the user table stores personal information about users, including sellers and buyers, containing user ID, name, address, phone number, and email. The order table records the transactions between sellers and buyers and it has attributes such as item ID, ordered date, quantity, seller and user's ID, price, etc. In addition, information such as brand, price, and condition of items that people are willing to sell would be stored in the item table.

Since we choose SQL as our main database language, we plan to use MySQL to build our database system. MySQL is open source and has a robust database system that performs efficiently. We can also connect it to Python which makes our implementation and management much easier.

We planned to collect our second-hand items dataset through web crawling from websites like eBay, Vinted, Craigslist, or Facebook Marketplace. Our website also allows clients to create their own accounts and add data (the items they want to sell) to the database through the website we built. Our plan B would be collecting data using our own items and information to be posted as second-hand products. Moreover, we can conduct surveys to gather information about used items that want to be sold by our friends, and peers in our local communities. We later sought to plan B, combining data collected and pre-processed from websites like Kaggle, and information gathered from our peers.

For the front-end, we have decided to use ReactJS. We chose this React framework for our web application because it is currently the most popular, industry-standard method for creating a Single Page Application (SPA) and also because we had experience with it. We intend to build a SPA as it allows for new “screens” to be loaded quickly without waiting for a new HTML page to be delivered. This function would improve the user experience drastically. Alternatively, we could have chosen Angular or Vue.js to create the SPA, however, neither of these are as popular as React, nor did we have experience with them. Furthermore, we plan to use Ant Design UI as the component library within our React app, so we do not need to implement the look and feel of all the React components manually. In this way, it would drastically improve the efficiency of development and provide a cohesive and comprehensive design. It also allows users to switch from light mode to dark mode easily, potentially increasing the visibility and readability of some users.

As for the backend, we have decided on using Flask as our framework. Our Python-based backend plan is to utilize the Flask framework for the REST API. We chose this over other alternatives such as Tornado and, notably, FastAPI due to its simplicity, extensive community support, and versatility. With a minimalistic design, Flask is easy to learn, offering flexibility in component selection. Its large community ensures ample resources for developers, while middleware support and wide adoption make it adaptable for different applications. Although Tornado has good performance with asynchronous capabilities, and FastAPI has various modern features, the balance between simplicity and extensibility for Flask is an outstanding advantage that benefits a wide range of projects. We also planned to use Postman to create API tests.

Architecture Design

Our project design includes four parts: back-end, front-end, database, and database management system. Users can register an account, post items they are willing to sell, or view items on the Home page from the front-end. The back-end defines classes, creates Flaskforms, and builds four routes so that it constructs a connection between front-end and the database. In detail, it can extract data from the database and upload them to the front-end such as displaying all posts on

the Home page. On the other hand, it can download data entered by users from the web page and store it in the database. Also, the database manager would have a data management system to get access to the database directly via a Python file using command lines. More details about our implementation and the content of each part will be presented in later parts.

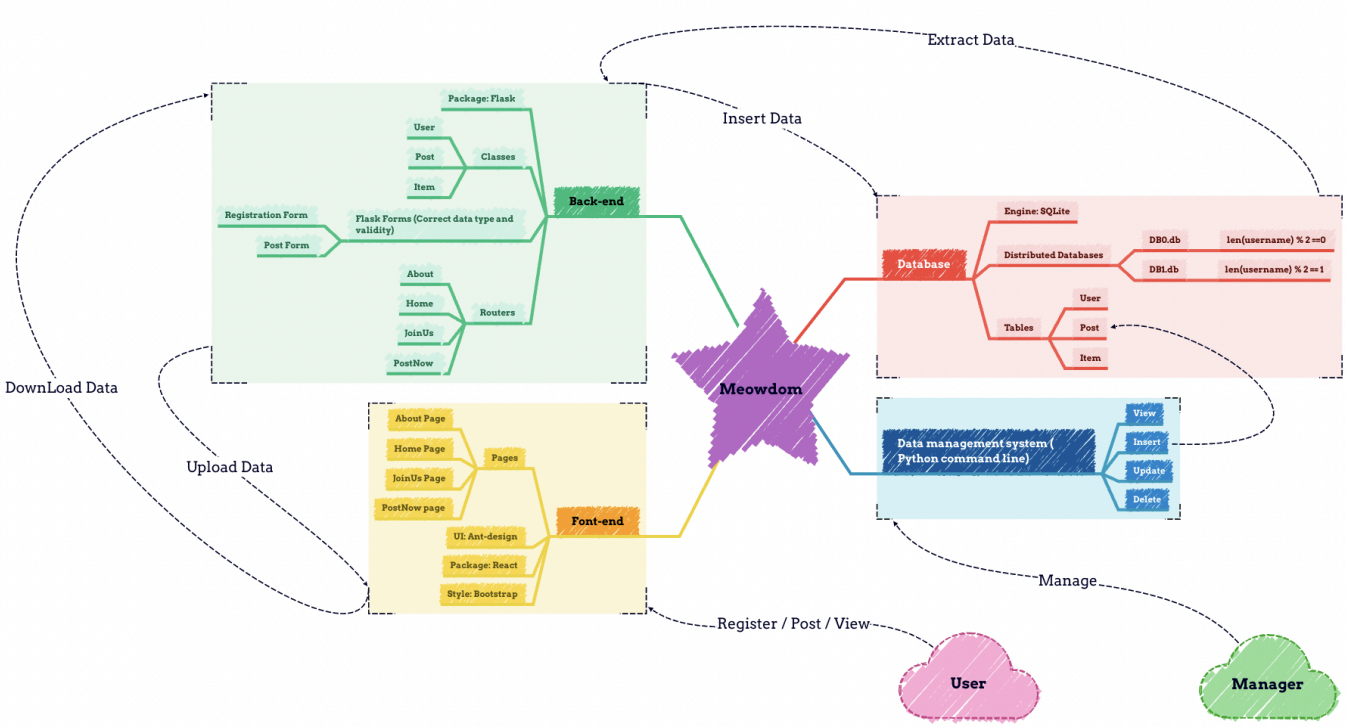

Figure 1: Flow Diagram

Implementation

Functionalities

For database managers, our Python script supports viewing, editing, deleting, and creating all data stored in the database, including posts and user data. In the app designed for end-users, customers can **register** for a new account, **post** the items, and **view** the list of items being posted on the website on the home page. The JoinUs page enables users to enter user info including username, contact info, and address. The PostNow page enables registered users to input the item info they want to sell, including category, original price and selling price, description, used condition, etc. All information will be stored in the database safely for future use.

Tech Stack

Our project is a flask-react web-based app. For the backend, we used Python Flask and its dependencies: Flask-Login, SQLAlchemy, and Flask-WTF (WTForms). Also, SQL and SQLite are utilized as our database language and database engine respectively. For the front-end, we used create-react-app as a start to build the react web app, and Ant-Design UI as component elements. The tech stacks that we actually used did not deviate much from our planned implementation, in which part includes more detail about our considerations and weighing in choosing the above tech stack.

Database

We created a distributed database in SQlite with two databases, DB0 and DB1, and we connected it with Python to insert and extract user or item information by importing the package SQlite3. According to the structure of our website, a user should first register an account and then post the items they want to sell which means that we would first insert the account information for each user before inserting the post and item information into the database. Thus, we partition our data into two databases based on the hash value of the username. Based on the length of the username, if it is even, then all information about the user including their personal information and the information about the items that they posted would be stored in DB0. Otherwise, if the length of the username is odd, all information of that user would be saved into DB1. As the schema (Figure 2) shows, each database includes three tables: post table, user table, and item table. The user table contains the account information of each user including an ID as primary key, username, email, phone number, and address. The post table contains the title, post date, and content of the posts posted by each user and we have post_id, and username as the primary key and foreign key respectively. Finally, all item information, including item name, brand, condition, and etc., corresponding to the posts would be stored in the item table. Item_id and username are the primary key and foreign key for the item table respectively.

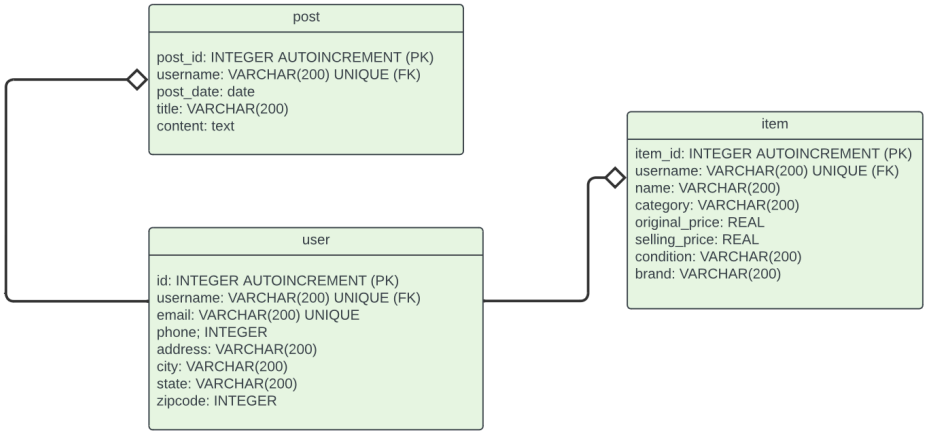

Figure 2: Database Schema

Following the database design, all data collected and stored to our databases should maintain the same data type as stated in the SQL schema. In terms of data type, all id-related attributes and numeric attributes, such as phone and zip code are designed to be integers. Considering price can be a float number we set the data type as real numbers. For contexts where users may enter longer notes, we use the text data type. All other data are set to VARCHAR(200) to ensure a consistent and efficient data structure.

Back-End

We mainly used Python Flask for the backend construction of our website Meowdom. Since we do not have much experience in web development, we learned how to use Python Flask and how to build the backend structures by watching a tutorial given by Corey Schafer on YouTube. The following directory (Figure 3) shows the structure of our project. Specifically, the backend implementation contains four Python files, _init_.py, forms.py, models.py, and routes.py, which are included in the flaskblog directory, and a run.py file which is the file that we should run to start the backend.

```
Meowdom
│   README.md
│   Meodom-Demo.pdf: google slides for presentation
│
└──src
│   │   run.py: a file to run the backend code
│   │   requirements.txt
│   │   DB0.db: SQL database 1
│   │   DB1.db: SQL: database 2
│   │   dbmanager.py: an APP for manager to insert, update, and delete data
│   │
│   └──flaskblog (backend)
│   │   │   __init__.py: initialize the APP, create db engines, and define hash function
│   │   │   models.py: build three classes: user, item, and post
│   │   │   forms.py: build two flaskforms(registration and post) to correct the data type from web and check validation
│   │   │   routes.py: construct four web pages: home, about, JoinUs(register), and PostNow(post)
│   │
│   └──client (frontend)
│   │   │   ...
│
└──instance (directory automatically generated by flask APP)
│
│
└──pic (pictures for proposal and web design
    │   ...
```

Figure 3: Back-end File Directory

Firstly, the function of the file _init_.py is to initialize our app and create two database engines to connect to DB0 and DB1. Also, we define our hash function in this file to return the correct database session given the input username. In addition, in the models.py file, we constructed three classes: user, post, and item and the content of each class is consistent with the tables in our database. The importance of these classes is to help us create a user, post, or item object after we get the related information from the website so that we can insert the object into our database easily.

Moreover, forms.py creates two classes RegistrationForm and PostForm and both are Flaskform. The function of these forms is to transfer the data inputted by the users from the websites into the correct data type that is consistent with the datatype in our database and conduct validations to check if the form is valid to submit. In detail, RegistrationForm would ensure that when the user registers a new account, other users should not use the inputted username and email. Otherwise, it would give a message to let the user choose another username or email which guarantees that the username and email are unique in the user table. In addition, when a user wants to create a new post to sell items, we would ask for their username when they fill out the post page. We would make sure that the user has already registered an account with us before posting their items by searching the two databases to check that the username is contained in one of the databases. Otherwise, we would send a message to encourage them to be a member of MeowDom and post items after they have successfully registered a new account. Indeed, the classes in models.py and forms.py would be used in routes.py to help us insert data from the front-end into the database system.

Finally, the file routes.py implements four pages for our websites: Home page, About page, JoinUs, and PostNow. We would import three classes from models.py, two forms from forms.py, and the hash function from the _init_.py. Firstly, the home page would display all the posts by extracting all the information from the post and item page which includes the title of the post, the content of the post, item name, item condition, original and selling price, .etc. We would also have an About page to give a brief introduction to our platform. Then, JoinUs is a page for user registration. After getting the data including username, id, email, and .etc from the registration page of the frontend, we would first create a RegistrationForm to correct the data type of the data entered by the user and then validate if the username and email are not used by other users. Then, we would create a user object based on the data in the RegistrationForm and insert it into the correct database according to the length of the username. Similar to JoinUs, PostNow would transfer the inputted data from the website into the correct type that we need in the database and then check the validation that the user is contained in the user table of one of the databases. Then, we would separate the input data into the item table and post table and finally, we can insert them into one of the two databases according to the value of our hash function based on the length of the username.

Front-End

Our front-end of the project is built from the *create-react-app* building tool, which is an officially supported way to create a single-page React. The structure of this app is shown in the following screenshot:

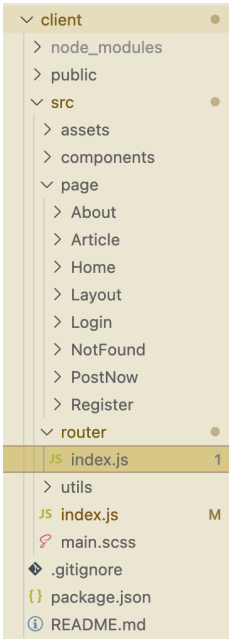

Figure 4: File Directory for Front-end

The SPA is enabled using the react browser router instead of the hash router. The logic of the project's router can be viewed in *client/src/router/index.js*. After successfully starting the app, it compiles the *client/src/index.js* file, and the <RouterProvider> element would render the pages stated by *client/src/router/index.js* file. According to the router file, our app would load *Layout* and the default second-level outlet element, which is */home* page, in one goal (The user would be redirected to the home page with the root URL to */home*.) On this page, items being posted in the PostNow page will be listed one by one.

Users will be directed to the About page for detailed info on our app. It only contains the brief of our project currently. New users can register an account on the */register* page. The form will store user info including username, email, phone number, address, city, state, and zipcode. After filling out the form and clicking the submit button, an alert message saying 'Registration Successful!' will appear. After 3 seconds, users will be redirected to the home page listing all posted items. Our app currently only supports users in the US so the phone number prefix is restricted to option +1 only.

Registered users can post items on the */PostNow* page. The form will store item info including item name, category, condition, brand, original price, ideal selling price, title, and content. Our APP currently only supports transactions in US Dollars so the currency option is restricted to $ only. A */NotFound* page is implemented to handle unexpected user actions. For example, if a URL that does not exist has been entered, the app will direct users to the NotFound page saying "Oops.. Something went wrong", instead of crushing the whole Meowdom app. This is a way to increase user experience.

Database Manager

For better organizing and managing our distributed databases, a robust database management system is essential so we created a database manager system with four functions where data managers can view the existing data, insert new information, update the database, and delete the data entry when necessary. It ensures the efficiency and scalability to support the management of vast amounts of user and product data stored in MeowDom. The primary control of the data manager is handled by “dbmanager.py”, which is directly linked to our databases, DB0.db and DB1.db, through SQLite3 packages. There are mainly three structural steps for implementation design.

The first step is to locate the appropriate database. Username is set to be the primary key for all tables in our database design, making it practical to use “Username” for searching databases by applying a hashing function.

Second, in managing the existing database, it is crucial to verify the identity and data types of the input data. This includes the identification of the username and matching data type of the input data. In other words, we are ensuring two questions: 1. Is the user that we want to make changes in our database? 2. Is the input data valid or not? To incorporate this functionally, we configure the data type for inputs aligned with our SQL schema and apply “check_username” and “check_email” functions to test the validity of user credentials.

Third, we have developed four functionalities for the data manager: view, insert, update and delete. Each function corresponds to a set of Python functions for the three tables within databases. Considering easier control and efficiency, we encoded each command to one numeric value. If the number is not encoded, questions will keep showing up until the data manager enters the correct commands or quits the system. This design ensures stability and increases fault tolerance.

```
Welcome to Meowdom database management system!
Which command do you want to execute? 1. View 2. Insert, 3. Delete, 4. Update, 5.Quit
Please enter the number of the command that you want to execute: 2
Which table do you want to insert into? 1.user, 2. post, 3.item, 4.quit
Please enter a number or quit: 1
Please enter the username: Peter
Please enter the email: peteremail123@gamil.com
Please enter the phone: 1234567890
Please enter the address: 111,west,67st
Please enter the city: Los Angeles
Please enter the state: CA
Please enter the zipcode: 900008
Added the user Peter successfully!
```

Figure 4:Data manager interface (Insert Command)

Implementation Screenshots

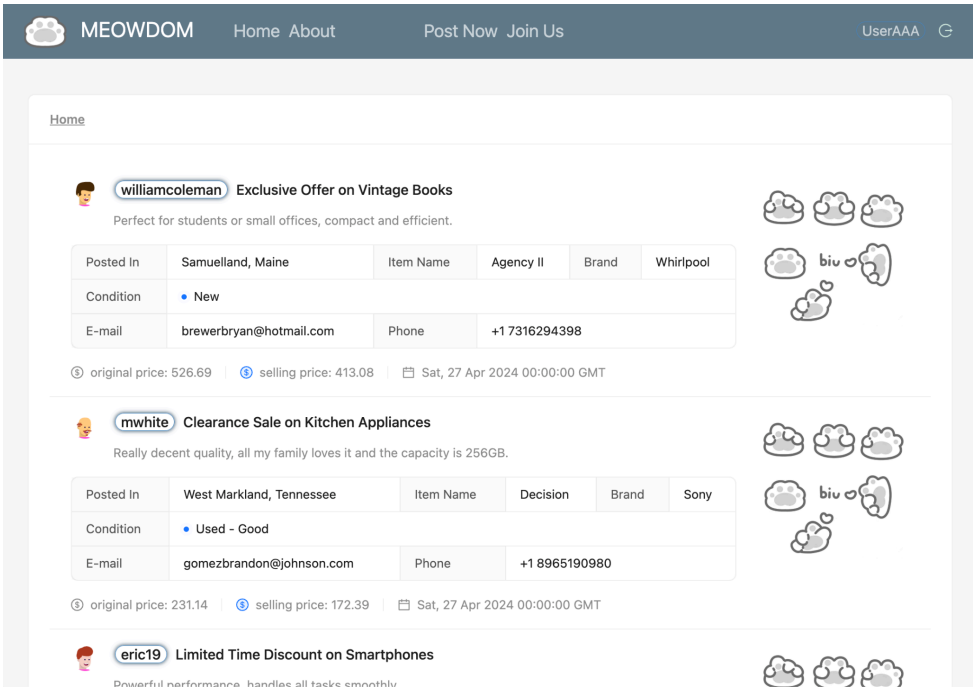

Figure 5: Home Page Part 1

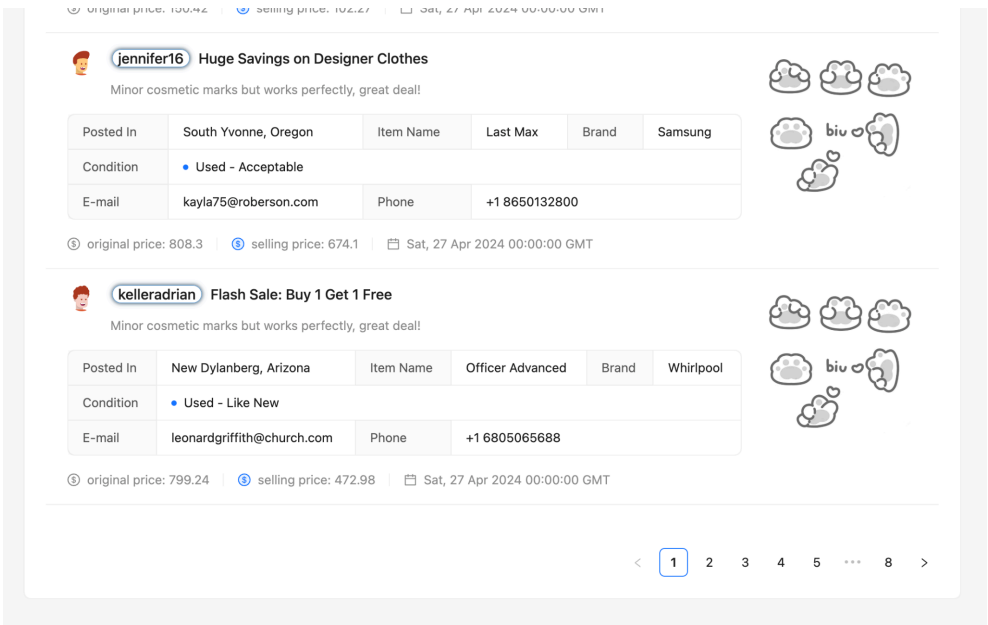

Figure 6: Home Page Part 2

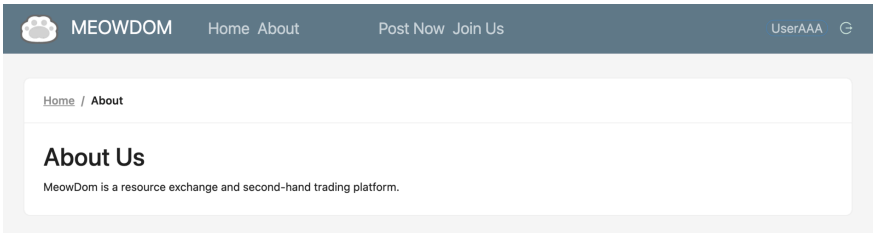

Figure 7: About Page

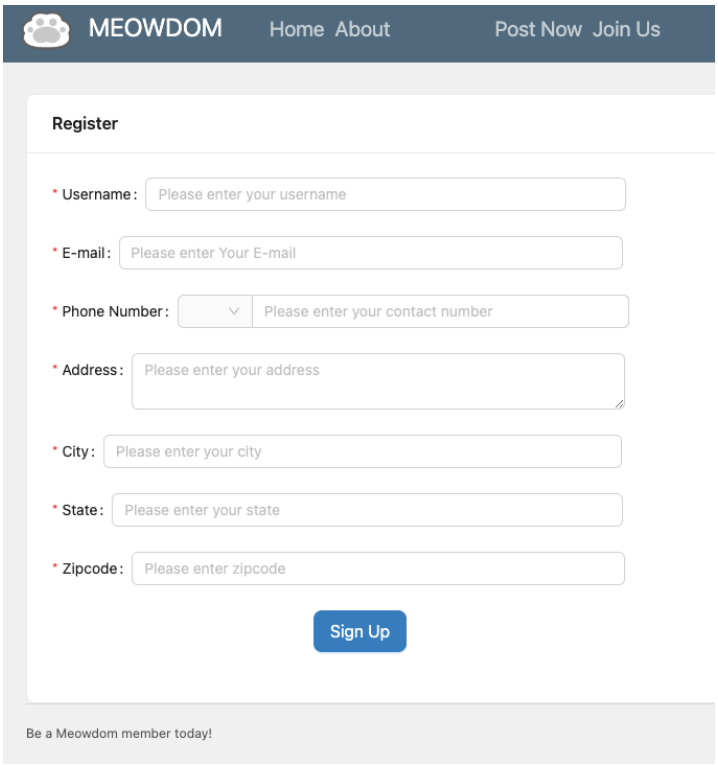

Figure 8: JoinUs

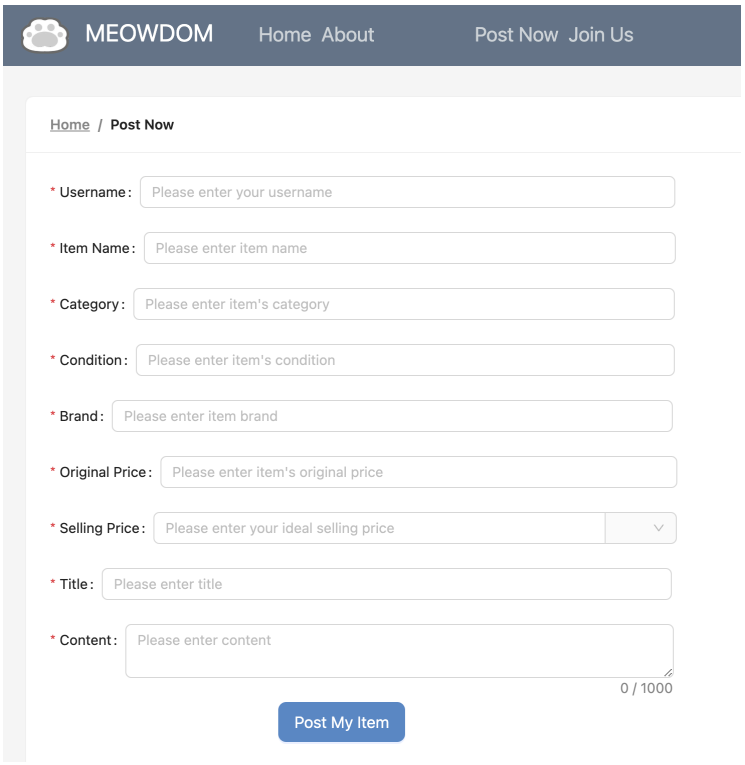

Figure 9: Post Now

Learning Outcomes

From this project, we gained experience in web development. By constructing our second-hand transaction website Meowdom as a group, we learned the way of designing structures and flows

for a website, the method of connecting the backend with front-end and database, and the key features of creating a user-friendly website.

Challenges Faced

One of the biggest challenges that we faced was that our login function failed when we used a distributed database system. We used the Loginmanager in Flask to manage the login and log-out of the users. We found that the user authentication in the Loginmanager is true in the backend (route.py) but it returns false in the frontend. We tried to create a login manager by defining our own function such that if the user existed in our database and the password matches, it would let the user login and set the login status as a global variable which equals 1. When they log out, we would set the login status as 0. However, this method does not store the information about the current user and it would be a memory cost to save all the information about the user. In the end, we gave up the login function. We still keep the registration for users. Instead of login, we would ask them to enter their username when they want to post and if they do not have an account with us they need to be a member of us before posting. However, we would explore more about Flask Loginmanager to add the login function in the future.

Another challenge was that we kept getting an error that the registration form was not successfully submitted when we just connected the backend with the front-end to test if we could register as a new user. Initially, we used the validate_on_submit() function which we learned in the flask tutorial in wtforms to check if the registration form is ready to submit but we found that this function does not work for our website and it always returns false. Later, we defined our own validation checking function and called it in the router to solve this problem.

Individual Contribution

Kexin and Jiawen mainly focused on Python Flask backend development and database systems whereas Liujia mainly focused on the front-end because of her experience in front-end web development using ReactJS framework. Although different tasks and parts we worked on, as a team, all the outlines and structures were discussed in groups by holding weekly meetings.

Conclusion

In conclusion, this project aims to enhance the efficiency of data use in daily life, with a focus on second-hand trading websites integrated with a distributed database system. The design encompasses both React front-end structure and Flask as back-end implementation with an operation of a data manager system to effectively manage various user and product data. Our

distributed database design incorporates SQLite database, using SQL to manage the relational table with structural data and “username” as the key for the hash function. This approach not only enhances the performance of data management but also avoids data inconsistency, which is essential for future scalability.

Our implementation design mainly faces two groups of people: users (including the buyers and sellers) and data managers who are responsible for the database management. The Meoedom website allows potential sellers to post their used items through the PostNow page after registering with us and also servers potential buyers who can simply browse second-hand items through the Home page and easily get the contact information from sellers as well as some supportive information to better assist the decision-making process, such as condition and original price. The data manager system is mainly for database management where the manager can view, insert, update, and delete data from the existing databases. In essence, this project not only provides a solution for effective second-hand trading but also addresses the technical challenges of managing various data in the second-hand market.

Future Scope

In future development, we would try to resolve the login issue first, which requires user authentication steps. To achieve this, both front-end and back-end need to add authentication functionality. Our current user-end app only supports viewing and posting item information. It would be better if the users could edit the items they posted. This functionality can only be achieved after successfully resolving the login issue so that users can only edit posts of their own.

The detail pages for the posts would also be constructed if we had more time. The <Article> component is already being established in */client/src/page/Article* directory for developing the detail page. A critical requirement is enabling users to upload, delete, edit, and view photographs. This necessitates an integrated system with a front-end interface for user interactions, a back-end for processing multimedia data, and a database for secure storage and retrieval of photo files. To speed up the initial rendering process, we plan to implement pagination, which retrieves only a certain number of posts from the database (e.g. 20 entries or 50 entries at a time).

In addition, we can launch the function of comments and likes such that users can leave a comment or give a thumbs up to the items they are interested in. Moreover, to make the website look more like a real-world second-hand transaction platform functions like chatting with buyers to ask for more details about the items and creating a page ordering and transactions can be added to our current website. For the database part, we would explore how to store pictures in our database. In this way, sellers do not need to provide a long description of the items and buyers can see the size, condition, and color of the products.

Project Link on Github:

https://github.com/LesleyYu/MeowDom

Reference

1. Flask tutorial: https://www.youtube.com/watch?v=MwZwr5Tvyxo&list=PL-osiE80TeTs4UjLw5MM6OjgkjFeUxCYH
2. Ant-Design: https://ant.design
3. Create React App: https://create-react-app.dev