Software Requirements and Design Document

For

Group <5>

Version 2.0

Authors:
Zhixi L
Hanyan Z
Yuanyuan B
Wesley W
Dennis M

1. Overview (5 points)

This website will allow FSU students to easily trade used textbooks among themselves. Trading textbooks is much more economical for students than to have to buy new ones all the time. This website will help students save money and reuse resources, which is also good for the environment.

The website will have all the basic features that websites for trading used books should have, which include but not limited to user system, posting feature, and booklist page. The website will allow FSU students to trade used textbooks of previous semester for textbooks of the classes they are currently taking.

2. Functional Requirements (10 points)

- The login page shall authenticate the user with user-inputted username and password through Post request. If the user entered the wrong password or username, prompt the error message. If user input matches with a username and it's password in the database, then redirect to homepage. Users shall be able to check the box to keep them logged in for 30 days, and if they do so, they shall not be logged out in 30 days.(High)
- If the user has already logged on, any access to the login page shall be redirected to the home page. (Low)
- When a user searches for a textbook, the web app should compare the existing books' names with the keyword. Display all matching books to the user. (Medium)
- The header should display the full name of the user if already signed in, or "offline" if the user is not signed in. (Low)
- In the booklist page, app.py should find all the posts in the post table, order by the time posted, then send it to be displayed on the booklist page. (High)
- When the user accesses the home page, app.py should automatically fetch 12 most recent posts and display it in the home page. (Medium)
- In the booklist page, the users should be able to change the ordering of the books by clicking on different buttons. The default ordering is by the time in which the books are posted. Users will be able to change it to order alphabetically. In addition to this, the booklist page should also support filtering by colleges. There is a drop down menu where users can select from a list of fsu departments. For example, if the users click on filtering by the College of Computer Science, the list should only display books in courses belonging to this college. (High)
- In the homepage, the users should be able to click on About Us and Commonly Asked Q&A to get a general idea of how this web application works and commonly asked questions regarding the uses. (Low)
- The web application shall be able to redirect the users to the homepage from any pages whenever they click on the website logo located in the header. This requirement is derived because not every page has a "back to homepage" link, which may result in difficulties for the users when they are stuck in a certain page and cannot return to the homepage. (Medium)
- In the register page, the users shall be able to enter their information and the web app shall check whether the inputs are valid. If not, alert messages will be displayed. Otherwise, users shall successfully register and log in automatically, and then the web app shall redirect the users to the homepage (High).
- The book detail page shall display the complete information of the correct book after the users click on the book title through homepage, profile page, and book list page.
- The users shall be able to click other users' username in the book detail page to get into the profile page. The profile page shall display user information (not the password) and a list of books that person had posted (8 most recent ones).
- In the post page, users can upload basic information about the textbooks that need to be traded. Such as the name of the textbook, its degree of newness, price, physical photos,

and the classification of the textbook, such as the textbook of which college it belongs to. All listed are required to the user, because those informations will be used for the search and order area. The Angular Post page shall send data user inputted data to Flask backend and the backend will validate user inputted data and send messages back to Angular. If the input is not in the correct format, alert messages will be sent back. Success message will be sent back otherwise. At the same time, after all data are validated, the data will be send to the database on the Flask side.(High)

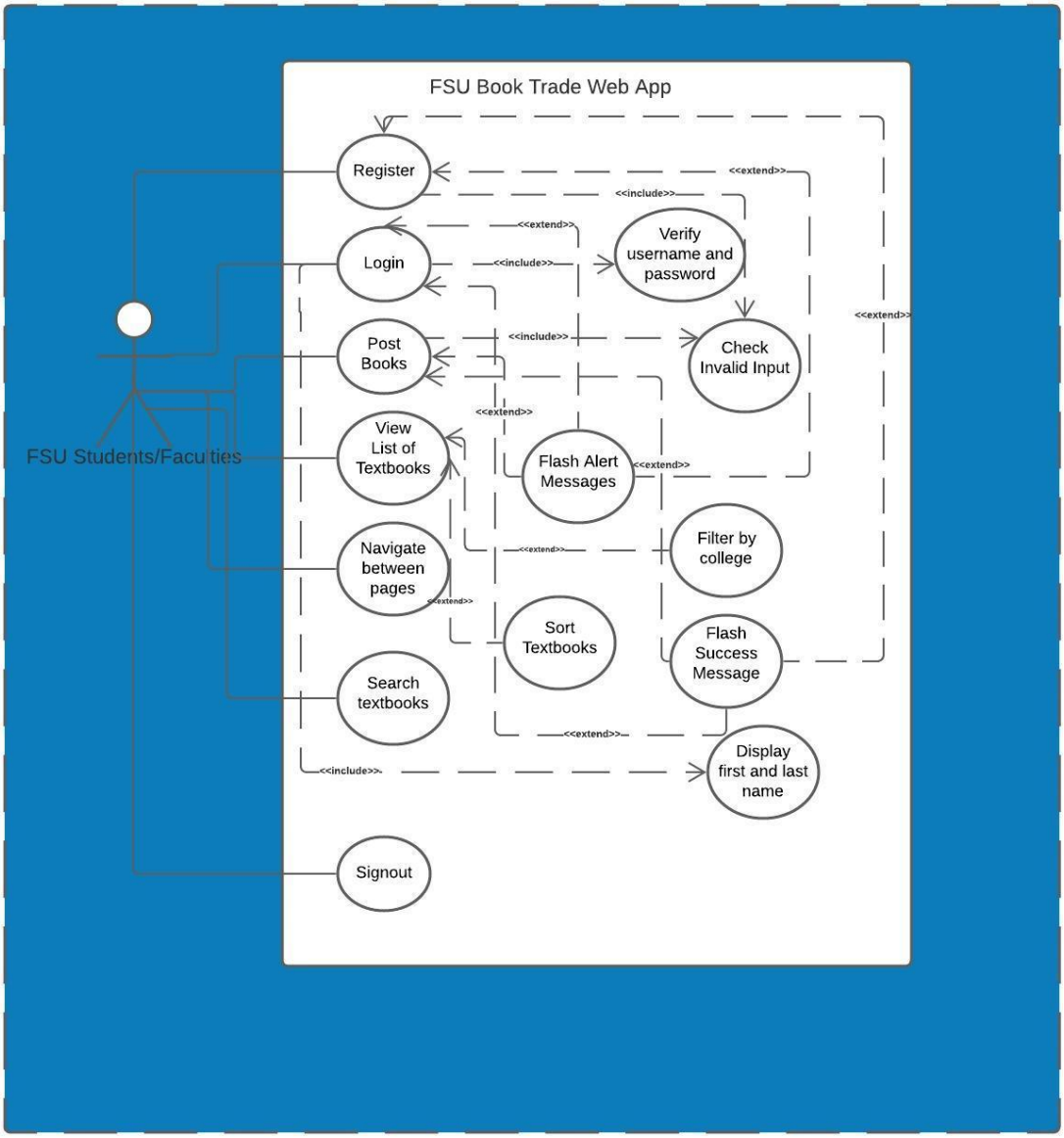
- For the category of the textbook, which is a “drop down menu”. It listed all colleges of department. The user only needs to select one. This category also helps users to search textbooks.(Medium)
- The description area for the user to add more details about the textbooks, but it is not a required.(Low)

3. Non-functional Requirements (10 points)

- Use SQLAlchemy as the ORM to manage the database.
- Use SQLite3 that came with python as the local database to store account & post tables.
- Route Controller app.py should not send sensitive data to users such as the password of a user.
- Search functionality for people to find the posts in 3 seconds.
- System should keep track if the user has signed in or already signed out at all time.
- Use the token-based authentication for keeping track if the user has signed in or not.
- Keep not signed in users out from pages that require sign in, and keep user sign in again if already signed in.
- The system shall keep user information secure all the time.
- The system shall not let users see username or password after they inspect the page.
- The front-end should be implemented using HTML/CSS/JS/TypeScript and Angular.
    - Sorting and any other similar features should be implemented in the front-end using TypeScript and Angular. This requirement is derived because implementing everything on the server side can make the application less responsive due to the round trips the data has to go through.
- The system shall not become unresponsive in any cases and shall be able to restart itself whenever an error occurs.
- The system shall not display any complicated error messages when something goes wrong. Instead, the error message shall be easy for users to understand.
- The system should store user information in the correct database.
- The server-side should be implemented using Python and Flask.
- The client-side should be implemented using Angular with Typescript.
- Using HTTP Request from Angular to connect front end and back end.

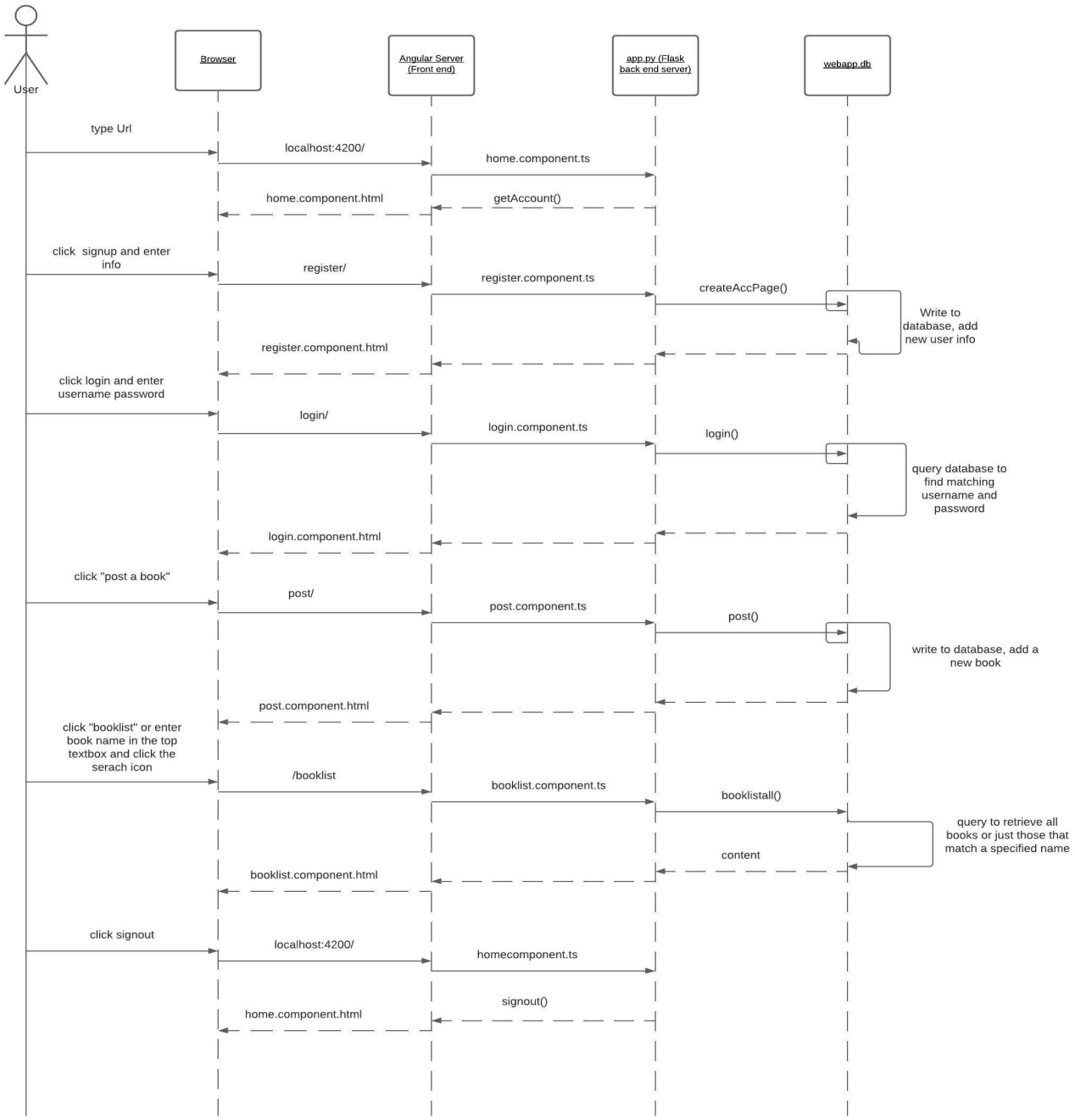
4. Use Case Diagram (10 points)

The register and post books use case will check user input. If the input is invalid, it will display different alert messages according to the situation. If username, email, or fsuid already exists in the system, the app will display username/email/fsuid already exists accordingly. If the input is appropriate, the app will display a success message.
The login use case will verify if the username exists in the system and if the username and password matches. Then it will display alert messages or a success message accordingly.
The view textbooks use case will have features to sort textbooks alphabetically and filter by colleges.
Users are able to navigate between pages and search textbooks from any pages.
Lastly, users are able to sign out of their account.



5. Class Diagram and/or Sequence Diagrams (15 points)

Sequence Diagram



6. Operating Environment (5 points)

For back-end server to operate:
The Windows 10 or MacOS 10.15.7 both with at least python 2.7.18
Pip installation of flask, Flask-Sqlalchemy, flask-cors, pyjwt

For front-end server to operate:
angular/cli 11.2.5
Node: 14.16.0

Run Backend:

- python3 back-flask/app.py

Run Frontend (must be at ./front-angular, and run (npm install) to install necessary node_modules)

- ng serve

For Client to access web app:
This web application should be able to operate on any browsers including Safari and Chrome with the link “localhost:4200”.

7. Assumptions and Dependencies (5 points)

- The Mac/Linux environment is different from the Windows environment, as our team comprises Mac & Windows programmers, these two factors may cause compatibility issues as we go deeper into our web-app development.
- For our front end to run, we rely on the node-modules. However, since it’s generated locally, there may be differences between each of our node-modules, which may cause some problem when running the Angular server in the future.
- We may reuse some image processing components from other open source projects for our image upload, snipping, and attach to our book or user avatar.