Software Implementation and Testing Document

For

Group <5>

Version 2.0

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

1. Programming Languages (5 points)

Typescript:

- A great language that's based on JavaScript with strong typing. Very easy to learn, the syntaxes are simple and straightforward.
- Used with Angular, our front-end server for data processing.
- The reason we choose this language, or the Angular instead of Angular JS is because the component separation makes everything look clean and good for managing datas.

Python:

- It serves as a web server, and we used it to interact with the database.
    - We choose to use this because it provides us with Flask, which is a micro web framework that is easy to learn in a short period of time compared to other existing frameworks.
    - We also used it to temporarily flash messages to alert the users when they entered incorrect username or password or both (this is planned to be done using a pop-up window in later increments).
        - This is great for testing because it is easy to implement but allows us to test the correctness of logic for checking invalid input by the users.

HTML/CSS: This is used to implement how the website look

- The reason we used HTML/CSS was so we could better improve the website. It also makes the website look more professional.

2. Platforms, APIs, Databases, and other technologies used (5 points)

Angular (version 6):

- Angular, as a total MVC pattern front-end server interface, allowed us to separate the view from the back end, and only concentrate on the view components, handling most data processing to the back-end server Flask.
- Integrate angular with Flask Python over the HTTP Service protocol by calling REST APIs.

SQLite3 & SQLAlchemy:

- Python also came with a database called SQLite3 that could be easily implemented & works perfect with the ORM that we are using called SQLAlchemy. This database stores information locally in a file. However, it's possible that we may switch to a more powerful & mainstream database later.

3. Execution-based Functional Testing (10 points)

- To test the functionalities of the login page, we first put some temporary user data into the database. In the Angular app Login page, we tested all cases including entering wrong username, wrong password, both wrong username and wrong password, as well as not inputting anything and clicking on the login button. Different alert messages flashed back for different cases. Lastly, we entered the username and password stored in the database, and the application allowed us to login successfully and we were redirected to the homepage. Besides this, we put console.log statements in our ts file and checked if correct information (alert messages or success message) is returned from flask. We also tried to access the login page after we were already logged in, and the application redirected us to the homepage. Also, we checked the console regularly to make sure that no error message pops up when we reload the page or click login. On the Flask side, we made sure that the page is responsive at all times during the login process.
- Redirect from /login to /index: We test this after implementing the session tracking for login status, then directly access the /login url to see if we got redirected. The user should not be allowed to access the login page if they have already logged on.
- To test the search feature, we entered keywords into the search bar from different pages and checked whether the textbooks with the keyword entered are displayed. Since this time we did search on the Angular side and added sort and filter features, we also made sure that these will not conflict with search and correct textbooks will be displayed at all times.

- To test the booklist page, we originally had 5 books in our database and went to the website to check whether they were all displayed correctly (same thing was done for the homepage). Then we clicked on buttons for sorting alphabetically and filtering by college to test whether correct textbooks are displayed. Then we entered more textbooks (more than 16) into the database to test the page navigation feature. After additional books were added, we went through the previous steps again to make sure everything works fine. We also paid attention to whether each feature would conflict with other features, such as we made sure that after doing sort, filter and search, the page navigation still works fine.
- The About Us and Commonly Asked Q&A pages are not yet being implemented, so we are going to test these features in the next increment.
- When we clicked on the logo located in the header from the booklist, post, and register pages, we were redirected to the homepage.
- To test the post book feature, we entered information to the input fields on the Angular side to test the functionality. First, we entered only numbers in the title and author input fields and checked if the alert message telling the user that the input for these fields cannot be just numbers. Then, we entered characters that are not numbers to the price input field and checked if a different alert message showed up on the page. For testing the drop down menu, we just clicked on it and checked whether all options showed up in the menu. Lastly, we entered correct information to all fields and clicked submit. To test the description input field, we made sure that we can submit the data with or without inputting anything. For each test, we checked whether the data was successfully sent to the Flask backend. If no inappropriate data is sent and all the required fields are not empty, the data should be stored into the database. On the flask side, we added print statements to make sure that correct information is sent from the frontend. Also, we console.log the message sent back from Flask in Angular to make sure that the correct message is sent back from Flask.
- To test the register page, we first tried all combinations of invalid inputs and made sure that appropriate alert messages are displayed on the page. We then enter valid inputs and see whether we are successfully registered and whether the data is stored into the database. Also, we made sure that if we registered successfully, we are logged in automatically and redirected to the homepage.
- To test the book detail page, we click on different book titles from homepage, booklist page, and profile page to make sure that the right information for that book is displayed.
- To test the profile page, we click on different users in the book detail page to see if the information for that user is displayed. We also made sure that the 8 most recent books posted by that user are displayed under the user information (if the user posted less than 8 books, they will all be displayed).

4. Execution-based Non-Functional Testing (10 points)

- To make sure that no username and password information is provided for the users, we opened the console after we test login and made sure that the information is not printed in the console.
- To make sure app.py did not send sensitive data to the users, we run app.py and ng serve the Angular app. We then navigate to all pages of the Flask server to make sure that no sensitive data is displayed on the page. We then do all the post requests on the Angular side and repeat the process. (Currently our Flask server still displays all information in the database, and we will try to fix it in the next increment).
- To test the search feature, we will do many different searches (at least 20) and make sure that it does not take more than 3 seconds for all searches.
- To make sure that the website keeps user information secure at all times, we navigate to all pages for many times to see if there are any circumstances that the user information is potentially not secure. Same thing is done to make sure that the website will not become unresponsive in certain cases.

- To make sure that the system will not display complicated error messages when something goes wrong, we tested our post, register and login with different invalid inputs to make sure that the alert messages are all easy for the users to understand.
- To make sure not signed-in users can't enter sign-in required pages, and make sure users can't sign-in or register if already signed in, we visit all these pages through the webpage, and through changing the address itself. We were able to be redirected to the home page, or login page with a warning, if we are not permitted to use that specific page.

5. Non-Execution-based Testing (10 points)

Review based:

- We test and ensure these requirements by letting Zack and Yuki review the code and ensure language wise are correct.
- We ran sqlite3 and used the .dump to see the data stored within the database, checking on the content of each table. Yuki has also used "DB Browser for SQLite" to visually check the content of the table. For the textbook posts, we can also test them by using our "/booklist" route, that lists all the books out.