**Software Implementation and Testing Document**

**For**

**Group <5>**

Version 1.0

**Authors**:

Zhixi L

Hanyan Z

Yuanyuan B

Wesley W

Dennis M

# Programming Languages (5 points)

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.
* 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.

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.

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

What we’re already using:

* Flask - Used for server-side code, route control.
* Flask-SQLAlchemy - ORM for database
* SQLite3 - Database that came with python

Planning to use in the later increment:

* Axios - To make web calls
* AngularJS - For client-side code

# Execution-based Functional Testing (10 points)

1. To test the functionalities of the login page, we first put some temporary user data into the database. We then 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 (requirement 1). We also tried to access the login page after we were already logged in, and the application redirected us to the homepage (2). After we were logged in, we saw that the header displayed the user’s first and last name instead of offline (4).
2. 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.
3. 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(3).
4. To test the booklist page, we first entered 5 books into our database and went to the website to check whether they were all displayed correctly (same thing was done for the homepage). The sorting alphabetically and filtering by college features are not yet being implemented, so we are going to test these features in the next increment(5).
5. 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 (8).
6. When we clicked on the logo located in the header from the booklist, post, and register pages, we were redirected to the homepage (9).
7. To test the post book feature, we entered information to the input fields 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 (10). For testing the drop down menu, we just clicked on it and checked whether all options showed up in the menu (11). Lastly, we entered correct information to all fields and clicked submit to send the data to the database (10). To test the description input field, we made sure that we can submit the data with or without inputting anything (12).
8. To test the registration feature, we entered invalid input into the text fields. The invalid input consisted of either something that was way too long and over the max character count or leaving the text field empty. We then checked if the correct error message was displayed when something wrong was entered. After seeing that errors were being displayed correctly we entered correct information into the text fields and checked to see that the database was updated to include the new account. If the new account was added correctly then we should be able to use the login page to log into our newly created account.

# Execution-based Non-Functional Testing (10 points)

\*All the following numbers refer to the RD document, section 3, Non-Functional Requirements.

Test conducted:

* 3, 4 - We’ve tested these requirements by signing in with a sample account and visiting all pages to ensure no password of the user himself or of other users can be seen, also the user full name is always shown on the header bar.
* 8 - We’ve tested this requirement by constantly visiting pages & going back to the index page for approximately 5 minutes.

Plan to test in future iteration:

* 4 - We haven’t statistically calculated the amount of potential posts going to be posted for our website’s peak. However, so far, we’ve limited the user cap as the amount of students in FSU.

# Non-Execution-based Testing (10 points)

Review based:

* 1, 2, & 10 - We test and ensure these requirements by letting Zack and Yuki review the code and ensure language wise are correct.
* 9 - This has been ensured by running sqlite3 and using 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.