4 Testing

4.1 Unit Testing

Frontend:

- We will unit test our React frontend using Jest (it is already built into React) and possibly mocha or ava. These are recommended by the React framework for unit testing.
- Every feature that we add should be unit tested for each of its functionalities. For example:
    - Page switching
    - User submissions
    - User login
    - Error outputs

Backend:

- We will be using Mockito to mock objects as part of automated unit tests.
    - Mocking objects created through mockito such as getAccountID to test the functionality of the controller and repository methods.
- We will also test the service part of our unit testing.

4.2 Interface Testing

We will be using Postman for simulating the connection between backend and frontend. This is critical to make sure that speedy development continues while both front and backend are being built initially. Along with that, JMeter and SoapUI are two other tools that have the ability to test API and other interfaces. If we need more tools for testing the interfaces, these two may help us out in the future.

When testing the combination of multiple interfaces, error handling and mockito will be very important. These will allow us to see many of the problems that arise while connecting the interfaces together. Debugging becomes much easier if well made mockito tests are used.

4.3 Integration Testing

Frontend to backend. Round-trip testing.

- Round-trip testing is easy to tell if it is working, as we will have error handling that will catch anything that goes wrong.
  - For example, if there is a wrong email address, there will be an error returned. Or, if the frontend cannot get information from the database, there will be an error as well.
  - Another example would be any CRUD features that we implement.
- The IDE debugger will be used a lot during the testing phase so that we can see exactly where there are issues from the connection.
- This step is critical as if there are problems getting information from the backend, the frontend cannot continue making features for the project.

Frontend:

- We will use interface testing for the frontend so that we can mock fake users to sign into that do not directly come from the database. This way, if there are any problems with the round-trip testing, the frontend can still develop features without losing too much time.
- This step is critical since each page needs to be working correctly and displaying the given information, so having a mock user will speed up the process significantly.

Backend:

- For the integration testing, we will be testing our Spring Boot Application.
  - For instance, we will need to create an account database for the login information of the user. In the account database, we will be testing the user type which includes the id, email, phone, name, and role.
- Part of the testing we will need to do is making sure the entities and tables that we intended to have relationships actually do and they work as intended.
- The tools we will be using are Spring Boot Test and JUnit testing

4.4 System Testing

Puppeteer, Cypress, and PlayWright are all tools that React recommends for automated end-to-end testing of applications. Because they are recommended, and come from reliable developers, we will be utilizing at least one of these applications to finalize our testing process once the application is complete.

We will also be utilizing manual testing with volunteers in order to get responses that a computer will not be able to give. Such as how the flow of the program feels or whether the program looks good or not. These are very important criticisms that will help give us a more polished, and good looking program overall.

4.5 Regression Testing

Part of our requirements for merging branches with new features is that they will need to pass all prior unit tests developed. This can be achieved using CI/CD on GitLab. Along with that, as stated in a previous document, we will be merging all changes into a develop branch before merging the final changes into master. This way, the master branch will always be working and we won't need to back track our progress if an unexpected error occurs. In particular, we need to test the security of our system and the accuracy of data being stored.

4.6 Acceptance Testing

We plan to work closely with our client to ensure that both functional and non-functional requirements are being met. We will verify our testing with our client on a regular basis to ensure that our testing meets our client's standards. We will also regularly perform regression testing after making significant changes to HandRaise to ensure it does not lose any desired functionality.

We will also check with our client to see if he has any additional requirements or recommendations, and include these in our testing as well.

4.7 Security Testing (if applicable)

Security testing will be a large part of our project, since students will be logging into the program using their netid and password. We will write algorithms to break our encryption algorithm and ensure that no essential data is viewable to users.

We will also plan to use fuzzing with our testing to ensure that unexpected inputs, whether to the login page or otherwise, do not compromise the security of the system.

We will also implement JUnit tests for our Spring Boot application that make sure users can only hit the API endpoints they are authenticated for and they don't have access to data they shouldn't.

4.8 Results

The results of our testing will be a stable and secure application that fills the requirements of our client. We have multiple developers working on this project and need to maintain regressive tests to make sure that new features do not break old ones. This can be done through our unit, integration, and end-to-end tests. Software industry standards recommend at least 70% code coverage so we believe if we reach this number or beyond, we will have a stable application. Frequent acceptance testing will ensure that when we develop new functionality, it will be serving the client’s needs for the application. Finally, we will include security testing to make sure that unforeseen events or bad actors cannot hurt our application or its users. Overall, all of the testing types and categories that we have outlined in this document will ensure that our application will fulfil our client’s and user’s needs, as well as provide a smooth experience.