4 Testing

As a part of the Agile development process, testing will be built out and executed as different system components are developed. While minor tests can be done throughout the development process, following the Agile methodology, the main portion of our testing will be done as the final step of that sprint's cycle. This may include, or be followed by a code review by another team member. This system should allow for us to stay organized and systematic, as it is a well-defined and documented process.

A majority of the project-specific challenges we will face throughout the testing process will likely revolve around the Google Location API. Testing this portion of the application will likely require the team to read a large amount of Google's documentation, and to familiarize themselves with the system. We may also then need to mock the API's features in order to test them thoroughly.

Throughout the rest of the document, we will be referring to the system architecture *(Figure 1)* below.

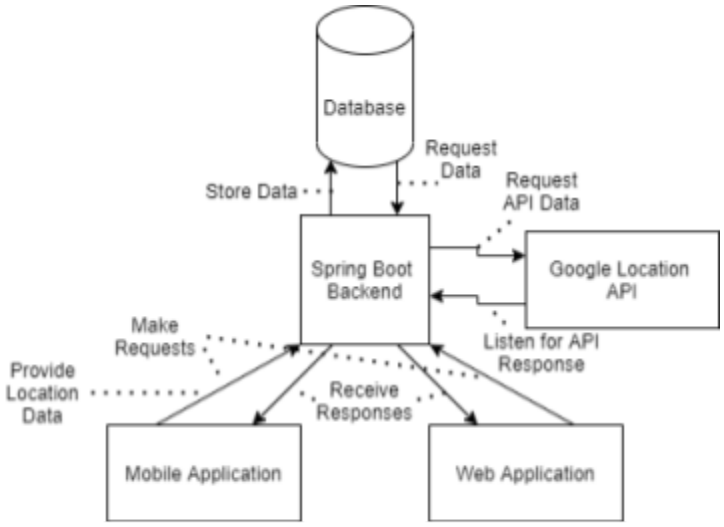

Figure 1: Diagram of Technical Layout

4.1 Unit Testing

Our system can be broadly classified into five major modules, i.e Database, Web Application, Mobile Application, Spring Boot Backend, and Google Location API,. For testing purposes, we plan to evaluate each module's functionality and features both separately and collectively. Referring back to figure 1, within each module there will be different functional units that will collaborate. A few examples of such units are as follow:

- Database module testing.
    - Updates (insertion/deletion)
    - Query procedures.
- API Testing
    - Load Testing using Jmeter
- Google Location API Testing

        - Mock all functionalities needed by the application
        - Use Jest matchers to confirm that the results from the mocks are as expected
        - For these examples, we will have mock transactions to test the correctness of the answer and their overall effectiveness.
- UI testing
    - React user event/Event response
    - Run test in different browsers
    - Accessibility Testing using AMP
    - Run test in different screen sizes

Some of the tools we will utilize for unit testing include:

1. JUnit
2. Mockito
3. TestNG
4. HTML Unit
5. Jest
6. JMeter
7. Mocha
8. AMP
9. Embunit

4.2 Interface Testing

We have two basic modes, mobile and desktop. The basic test for our front end UI will be conducted by running a questionnaire and/or survey with our peers or classmates. The survey will ask participants to identify potential issues with the appearance, functionality, and usability of the UI.

Our primary interfacing will involve interfacing the Google Location API with our Spring Boot backend application. This will involve making calls to the API from our Spring Boot application and returning spatial information relating to the data that was sent. The data may need to be transformed in a certain way by the application in order to make a proper API call. Our primary method of testing the API calls will be sending specific data of a known, specified location.

Tools:

Postman
Mockito

4.3 Integration Testing

Integration testing within the scope of our project will involve CI/CD, or Continuous Integration, Continuous Development. The purpose of our tests, and part of the need for CI/CD, will be to

ensure that implementation of new code or software does not negatively impact the working environment. This will involve writing integration tests each time a new piece of software is added, which also ensures that older code will still be tested when new functionalities are added. Most of the critical paths that will need to be maintained involve interaction between each unit. One of the most critical integration paths will involve an end-to-end connection between front and back end; this would entail storing and retrieving task and worker data from the back end per request of the front end. Another critical path will be the connection between the UI map and the location API; translating front end data to data that can be processed with API calls, as well as translating the returned values from the API back into data that can be sent to the front end.

We will check...

Examples:

- How insertion of a new worker and its location is reflected on the map.
- How a report is sent to the server and the corresponding algorithms handle said report.
- Retrieval and storage of incomplete and complete tasks.

Tools:
Github
Postman

4.4 System Testing

Refer to the functional requirements found in section 1.2 of the *“Requirements, Constraints, and Engineering Standards”* portion of the project documentation. The following system-level tests may be performed to ensure that said requirements were met:

- When viewing a map or job status page, current and up to date information should be displayed
- When an unauthorized user tries to access information, they should be rejected and redirected
- When a client posts a task, a visual representation is displayed and their task is stored on the server
- Mocked unit tests on location API to ensure data sent and received is as intended.
- Location tests to ensure the UI properly displays a worker’s location when needed.
- General flow interface tests will be done to ensure that the movement between portions of the app works as expected
- Tests listed in 4.1,4.2, 4.3 will also fall under the scope of system testing.

The tools used throughout this process may include (but are not limited to): Postman, mockito, Jest, and Junit. We will also use manual and peer reviews to test basic operations and user experience.

4.5 Regression Testing

Sometimes newly added functionalities or changes in existing code make the existing features non functional. And to avoid that, a test suite is chosen that covers both the existing features and newly added ones. And the chosen test suite consists of tests that cover different crucial

parts of an application, eg. impact, criticality and most used functionalities. In our case, two such crucial aspects to be tested are scalability and extensibility of the application.

1. We plan on testing the scalability by increasing the traffic on our website and increasing the load on our database testing its impact on the application's overall performance. Apart from load handling, the performance on the user's end will be tested to ensure that the users are getting real-time updates with minimum delays regardless of the fact that the web application or the mobile application is being used.
2. Our program should be extensible and addition of new features in the future should call for bare minimum changes in the existing application. For eg., updating the map information tests its impact on the map services of the app. It has to be ensured that the UI functions properly regardless of the location of the user.

4.6 Acceptance Testing

How will you demonstrate that the design requirements, both functional and non-functional are being met? How would you involve your client in the acceptance testing?

In order to ensure our functional requirements are met, we will use the demonstration with demo cases. Demonstration with demo cases will help us target specific requirements to ensure the detection of any faults.

In order to test out non-functional requirements, we will merely measure the requirements that are measurable and calculate the percentage of the requirements met to the total number of non-functional requirements.

We will involve the client in the acceptance testing. However, we will also have certain criteria for acceptance testing which the system as a whole must satisfy. A few examples include:

1. How visually appealing the overall UI is.
2. How intuitive the UI is.
3. How efficient and effective the reassignment algorithms are.
4. How extensive the design is.
5. How reliable the application is functioning.

4.7 Security Testing (if applicable)

While security is not part of any of the tasks for this project, we recognize its relevancy and we will adhere to it at a system level in terms of password protection, access writes, input sanitization, etc. However, we will not be developing any detailed plans for security testing.

4.8 Results

By the end of this semester, we expect to expand on different ideas for testing in differing contexts. However, we will do so only in terms of qualitative improvements, not quantitative. There will be other technical improvements as we move through the semester next spring.

One major example of testing proving that the design is as intended is that by using thorough testing we will be able to ensure that our application's algorithm contains all of the categories discussed in section three of the project documentation. This both helps achieve intended functionality and meets the client's specifications.

As for a general diagram of our testing structure, for sprints we will be following the Agile methodology. This methodology usually lists testing and reviewing as the 5th and final step of the cycle. For the categories listed in this document, we will be following a process similar to that shown in Figure 2 below.

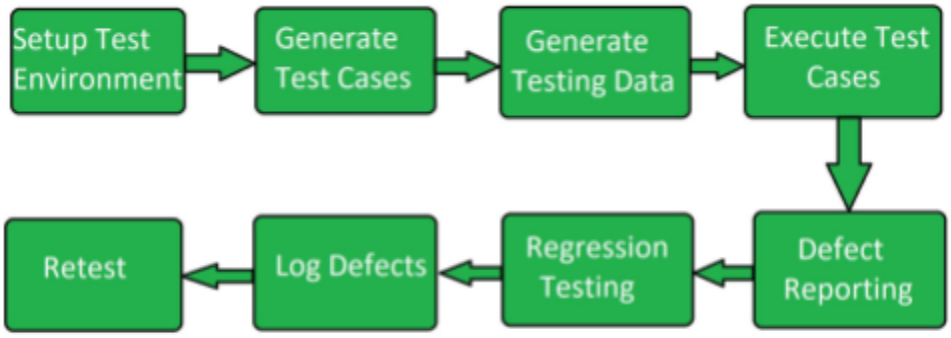

Figure 2: System Testing *(GeeksForGeeks.com)*