Detailed Test Plan

Test Cases

ORGANIZATION

The frontend backend and integration levels will all be divided into 7 categories of testing . These include login, register, profile page, sell, update, buy, and logout and are based on the requirements given.

ORDER OF TESTING

All of the individual specifications given that fall into each of these categories will be tested in the following order as this follows a logical chronological order of how a user could potentially use this application:

1. Frontend:
    a. Login
    b. Register
    c. Profile page
    d. Sell
    e. Update
    f. Buy
    g. Logout
2. Backend:
    a. Login
    b. Register
    c. Profile page
    d. Sell
    e. Update
    f. Buy
    g. Logout
3. Integration:
    a. Login
    b. Register
    c. Profile page
    d. Sell
    e. Update
    f. Buy
    g. Logout

Techniques and Tools

In order to execute effective systematic testing of Seet Geek, several techniques and tools will be required.

Step 1: Choose and Create Test Cases

- **Requirement Partitioning/ Functionality Testing**: used to separate the specifications into subgroups of testing levels (frontend, backend and integration) and then further into seven sub groups. The initial requirements were also given pre-partitioned.
- **Mocking**: used to help develop the test cases to isolate the behavior of a particular unit, and develop the corresponding test cases for that attribute.

Step 2: Execute Tests and Document Results

- **Functionality Testing:** will be used when executing the tests, as they are divided into all possible subgroups of requirements.
- **Robustness Testing:** will be used while executing tests to ensure the program does not crash no matter what the input. This will include boundary value testing.
- **Output Coverage Testing:** Integrating components of output coverage will be useful since there are an infinite amount of inputs in this application, considering the different expected outputs and what inputs will cause each will be useful as there is a limited number of outputs of this program.
- **Documenting Test Results:** The output of test execution will be put into a results file and summarized in a readable report.
- **Black Box Testing:** Used since the individual specifications given will be tested.
- **White Box Testing:** Used since the software's code will impact the results of the specifications.
- **Unit Testing:** When each partition is tested in the frontend and back end groups shown in the previous section.
- **Integration Testing:** When each requirement that connects the back end to the front end is tested
- **Mocking:** used to isolate certain test cases.

Step 3: Evaluate Results

- All test results will be evaluated automatically and each test will be repeated until error free and

- Every time a test fails, both the tests and software will be analyzed to decide which is at fault.
- **Acceptance Testing:** Used in order to validate if the software meets the user and clients intentions overall.

Step 4: Decide When Done

- **Black Box Testing:** will help indicate when testing is done, as we will have covered all the given requirement partitions.
- **System Testing:** Used in order to verify testing is complete and meets all functional specifications.

Environments

Testing environments of this software include Windows and Mac on local machines and Linux(Ubuntu) on the cloud.

Roles and Responsibility

Name	**Requirements**
Adam Hirani	Login and Register (R1 and R2)
Derek Moore	Buy, Profile Page, Logout, 404 Error (R3, R6, R7, R8)
Emily Gibbons	Sell, Update (R4, R5)

In case of failure, initially we each will have one person to contact, and the next step will be to have a group discussion about the failure. Emily will consult Derek, as her responsibilities are similar to his. Derek will contact Adam as the Login and Register sections will have some similarities as Buy, Profile Page and Logout, and Adam will contact Emily initially to solve any problems as some of his requirements will overlap with hers.

Budget Management

In order to minimize our CI minutes, we are testing on local machines as much as possible. The option for the program to automatically run on CI when the master code base is updated is also disabled to further minimize the usage of minutes. The cloud environment is selected as Ubuntu as that is a more efficient use of our minutes, as Mac and Windows runners consume minutes 2 and 10 times the rate that Ubuntu will. Emily will be responsible of keeping track of CI minutes and periodically updating other team members of the usage as she is the owner of the repository.