5 Testing

The following is an overview of the testing methodology that we will be applying throughout the development of our project. We will create tests for the functional and non-functional requirements, mentioned in section 1, to verify that the functional requirements are working as intended and that the non-functional requirements are meeting the needs presented by our client. An aspect that is unique to our project is that there are no specific cost related requirements as we are developing a proof of concept system. When components of our project, as seen in Figure 2., are implemented we will run them through a series of unit, interface, integration, system, and regression tests. Once these tests have completed, we will discuss with our client the results to find areas that could be improved upon. The subsections below outline when the particular tests will be performed throughout the development cycle.

5.1 Unit Testing

For unit testing, we can test the individual units that make up the total project. For this, we can test the different GUI components and their individual functions. Because we are using React, we could use the Jest framework to test these individual React components to make sure that they work on their own before we integrate them together. We also want to test our backend by unit testing the individual functions on the backend to make sure that our API works correctly. We also want to make sure that the Python Driver works and that the MapBox API works with our functions.

5.2 Interface Testing

The interface testing can be tested through combining multiple units. The combination of units will ensure that the interface of the application is able to successfully implement the interface. Here are a few examples of combined units we will test for interface testing:

Algorithm & Mapbox Database: The algorithm will be tested on its ability to get the correct data from the Mapbox Database.

Algorithm & Interface: The interface will be tested on how well it is able to visualize the algorithm.

Interface & Mapbox Database: The interface and Mapbox database will work together to build the bulk of the visual for the application. Both must be able to work together to display a map on the website and get the necessary data.

Algorithm, Mapbox Database, Interface: The overall interface that the user is interacting with will comprise the all three units— Algorithm, Mapbox, and Interface. We need to test that all components are able to cohesively work together to create a working interface.

5.3 Integration Testing

Integration testing will be done by breaking the system down into different functionalities and then testing these functionalities independently from one another. Integration tests will ensure that all of these individual functionalities are producing expected results. These functionalities combine components that are tested using interface testing. An example of one of these functionalities is selecting a city and having it immediately show up on the map. Another one which would be dependent on the first one already working is selecting a location in the city and having it show up immediately on the map. Once all of these functionalities are tested and working, the entire system will be tested using system testing.

5.4 System Testing

System testing will be done by the succession of multiple integration tests. i.e. System tests will follow multiple long paths and verify that at each step, what we see matches what we expect. These paths, and thus these tests, will stretch across the entire length of the system, from the Mapbox API to the user interface. This will ensure that, if a specific location is provided by the user at the user interface level, that choice will propagate correctly through all levels to the API and the API response will propagate back to the user interface correctly. To do this, corresponding unit, interface, and integration tests should be added together to chain through the system. All critical requirements (that paths are generated correctly, based on the correct algorithm, and begin at or visit the correct locations) should be verified using each of these system tests.

5.5 Regression Testing

To ensure new additions to the system do not break existing functionality, we will run existing unit, interface, and system tests, as well as, manually verifying system functionality to ensure that every existing part of the system still works as expected after implementing new features. In addition, we will also verify that all critical requirements are unchanged after the implementation of new functionality. One critical feature that will be tested and verified is the implementation of new algorithms for the route generation. After the addition of a new algorithm, we will verify that all previous functionality and important system system level functionalities are still working as intended. Additionally, as a stretch goal, after mapping a new city for our system we will verify that all previous cities still function properly and its data has not been modified.

5.6 Acceptance Testing

In order to ensure that the functional requirements of the project are being met, we will develop a set of use cases for functions of the application. The use cases will cover all possible uses of the application. During testing, the use cases will be followed as test routes through the application to ensure that the application is performing as expected.

Non-functional requirements will primarily be tested by demonstrating the application to our client to ensure that project requirements are met. Throughout the development, with each major change or milestone, the client will give the team feedback, and we will revise and improve our product as needed. The client will continue to give feedback until satisfied with the final product.

5.7 Results

We have not tested the design at this point. Next semester is when we will build our website and be able to run these tests.