# 4 Testing

Testing is an **extremely** important component of most projects, whether it involves a circuit, a process, power system, or software.

The testing plan should connect the requirements and the design to the adopting test strategy and instruments. In this overarching introduction, given an overview of the testing strategy. Emphasize any unique challenges to testing for your system/design.

For our team we aim to test a multitude of different scenarios that connect with our use cases and requirements. This will cover not only the database and the external api's, but also the backend connectivity as a whole. These cases will allow us to be sure that as a whole the backend for the system is functioning properly and is able to run the algorithm and generate routes from given data both from the api's and the database. While these backend unit and system tests are occurring the frontend mobile and desktop applications will also be tested. These tests will align with use cases regarding the various types of users that our program can have and how they would go about using the program. The challenge for this testing model is testing the connectivity of the frontend and backend. Since this project has only one backend that is going to be deployed to two different frontend platforms, the challenge will be coming up with a comprehensive amount of tests that are able to be used to verify use cases in both the mobile and desktop application.

## 4.1 Unit Testing

There individual units that exist within our program are the customer interface, the dispatcher interface, the driver interface, the communication service, the database, and also the use of external api that covers the algorithm for the program. These interfaces contain units within them that will all be individually tested to ensure that the use cases for each user are working correctly. This will be done through different ways though depending on which part of the program we are dealing with. For frontend testing we are looking at using Selenium or Appium in order to simulate a specific user running and working within the frontend. This will allow for tests to be run that follow the information pathway that is developed for that specific use case and see if the workflow that the user would follow in fact works within the scope of the frontend. Additionally for individual frontend pages field placement and style will be checked to ensure that text fits within input fields and additionally that pages have proper connectivity to each other within both frontend applications. For the backend of the project the database will be tested in a few different ways. Firstly we will run a series of queries and check the results in an attempt to see what commonly used queries do to the data within the database. These queries will follow the flow of use cases that were developed and will simulate the data that would be needed in

order to run the assignment algorithm. These tests and queries can either be run through a program such as MySQL workbench or through a terminal. For the backend algorithm code and the api's these will be tested through the use of following use cases with static data numbers. Standard expectations for the algorithm will be developed such that we can have expected results from these static data numbers and use them to verify use cases for different assignment scenarios. Additionally the api's will be tested to ensure that we are able to reliably use them to collect data and information that is needed for the assignment algorithm. These unit tests will follow the standard unit testing framework and will be done in Junit or the equivalent testing environment.

## 4.2 Interface Testing

The interfaces present in the current iteration of the application architecture are the customer interface, dispatcher interface and driver interface. The customer interface will be used by the customer to interact with the application functions and interact with order dispatchers via the communication service. The dispatcher interface will be used by the dispatcher to interact with application functions and interact with the customers and drivers associated with the orders. The driver interface will be used by drivers to interact with application functions and interact with dispatchers associated with orders.

In the current iteration of the architecture, interfaces are being treated as separate components, and access to the components are determined by user role. Shared interfaces such as user login and registration are shared across all users, but other interface components are user role dependent. Ex: Customer users can only interact with the customer interface and will be limited to interaction with the customer interface, and not the dispatcher and driver interface.

Access to specific API services (specific application function) is also dependent on user role and subsequently user interface. Ex: Http calls to the new order service are only allowed from the customer interface.

Interface testing will mostly be based on the black box testing model. Black box testing is the most suited model for interface testing as the goal of this testing method is not to dig deep into the code, going through the application's internal functioning, but to interact with the UI, test the end user functionality, and make sure that every input and output of the system meets the specified requirements.

The tools under consideration to be used for the mobile UI testing are Robotium and Espresso.

Robotium is an android Testing framework to automate test cases for native and hybrid applications. Robotium can be used to create strong automatic GUI testing cases for Android applications.

Espresso is a UI test framework that can be used to create automated UI tests. Espresso tests run on an actual device or emulator and behave as if an actual user is using the app (i.e. if a particular view is off screen, the test won't be able to interact with it).

Customer Interface testing scenarios :

- Customer mobile UI will actively allow user touch input such as swipes or clicks, in order to test functional status of UI components such as button on click listeners and swipe listeners.
- Customer web UI will actively allow user input such as mouse clicks and scrolls, in order to test functional status of UI components such as buttons on click listeners and scroll listeners.
- Customer mobile UI will redirect to the correct page on user input such as swipe or click, in order to test navigation between screens/pages.
- New order form will allow users to input data into form fields, and run UI side validations on the form field data.
- Messaging page  will actively update and persist message data between user and order dispatchers.
- Messaging page will correctly redirect to user messages on user click input,  in order to test messaging page navigation.
- Order tracking page/screen will respond to user order information requests with correct data, in order to test backend service functionality and frontend json response handling and formatting.
- Truck allocation page will display updated and accurate truck information on new user order input. Automated page navigation between new order form and truck allocation page will be tested.

Dispatcher Interface testing scenarios :

- Dispatcher mobile UI will actively allow dispatcher touch input such as swipes or clicks, in order to test functional status of UI components such as button on click listeners and swipe listeners.
- Dispatcher web UI will actively allow user input such as mouse clicks and scrolls, in order to test functional status of UI components such as buttons on click listeners and scroll listeners.
- Dispatcher mobile UI will redirect to the correct page on user input such as swipe or click, in order to test navigation between pages.
- Messaging page  will actively update and persist message data between order dispatchers, users and truck drivers.
- Messaging page will correctly redirect to user messages on dispatcher click input,  in order to test messaging page navigation.
- Order tracking page/screen will respond to dispatcher order information requests with correct data, in order to test backend service functionality and frontend json response handling and formatting.
- Route allocation page will display update and accurate route information on the google map component on the page. Calls to the google maps api and page map component updation will be tested.

Driver Interface testing scenarios :

- Driver mobile UI will actively allow user touch input such as swipes or clicks, in order to test functional status of UI components such as button on click listeners and swipe listeners.
- Driver mobile UI will redirect to the correct page on user input such as swipe or click, in order to test navigation between screens/pages.
- New order form will allow users to input data into form fields, and run UI side validations on the form field data.
- Messaging page will actively update and persist message data between order dispatchers and truck drivers.
- Messaging page will correctly redirect to user messages on driver click input,  in order to test messaging page navigation.
- Order tracking page/screen will respond to driver order information requests with correct data, in order to test backend service functionality and frontend json response handling and formatting.
- Route allocation page will display update and accurate route information on the google map component on the page. Calls to the google maps api and page map component updation will be tested.

## 4.3 Integration Testing

One critical path to test will be a customer User correctly see its orders. This will require the user to view their orders correctly in the web application by requesting information from the Order Tracking Service. The Order will be in a database and has to have been set from the Truck and Route Allocation Services. Additionally we will need to test the cycle starting from the Dispatcher Interface. The dispatcher web application will most importantly need driver orders which will be dependent on the Route Allocation Service, then the Route Allocation Service will be using an external API to process traffic and map data. Also, a Driver will use the Driver Interface in the form of a mobile application, this will be dependent on the Communication Service and Order Tracking Service. These three are our critical integration requirements because of our primary use cases. A Customer would need to track their orders, a Dispatcher would need to submit new orders, and a Driver would need to view its orders assigned.

One critical aspect we will assess is the response time from customer order to assignment. This will involve an event from the customer web application which calls the user order service, truck allocation service, route allocation service, map API, and calls our vehicle routing algorithm.

We will test this by performing a new user order from our front end web application, in Chrome Developer Tools we will be able to view the response time of this response which must use all of the services listed previously. .

## 4.4 System Testing

For system testing strategy is to focus on interaction between all parts of the system as much as possible, and individual tests should be less plentiful and more general. Starting with unit tests, they are still important for system level testing but don't need to be as plentiful or specific. For example, a set of unit tests on a file in the customer interface can probably be narrowed down to a single test that covers the general functionality of that file. In terms of interface testing scenarios, I think it's most important to zoom in on scenarios that cover as much of the entire system as possible. An example would be the order tracking scenario in the dispatcher interface, where the page responds to driver order information requests. As stated above, this test is important because it tests both service functionality on the backend and json response handling/formatting on the frontend. And then integration tests are made for interaction between all parts of the stack, so most of those can be kept and considered part of the system-level testing.

No new tools need to be added for these tests, but rather a combination of the same tools that are used for unit and interface and integration testing. Unit test files should still be made in the code editors that we build the app in. Interface tools should use both dev tools on the frontend and postman on the backend. And then integration testing will use a combination of these depending on the functionality being tested.

## 4.5 Regression Testing

The most critical of features to our program is the reassignment algorithm, so ensuring that new changes do not break this will be key when looking at regression testing. The reason this is so important is that the algorithm is what will be the key part of the project that all other components need to interact with, so any requirement for the project is directly tied to the correctness of this algorithm. To ensure that the reassignment algorithm does not break upon changes being made, firstly we will use the standard data set that was developed for unit testing to verify that with new changes the assignment algorithm still outputs the expected results. Once we can verify that the assignment algorithm works as expected then we can move on to automated tests on both the mobile and desktop applications in order to verify that data and requests are correctly being sent to the backend. This will narrow down the scope of any problems that arise to one specific frontend platform rather than having to work through both in the event of any bugs. Additionally these tests will allow us to verify if any problems are occurring with the database if specific known queries are used to retrieve information such that the result of what these queries return is known. The new features or implementations that we foresee that may cause issues is the use of a larger fleet of trucks or warehouses, which might cause issues within the database, or adding additional ui changes. The regression testing in these cases needs to verify the response time of the program as adding data such as this or new ui pages should not alter the response time of assignment in any drastic way. The only thing that would be a drastic change to the core of our project would be a change to the reassignment algorithm itself. This new algorithm would force our regression testing to follow a different method of testing each component of both the frontend and backend with every part except the new algorithm. This would ensure that all of the other components are functioning properly independently and together. From there the connections between the new algorithm

and each individual component would need to be tested and then finally automated tests like the ones mentioned earlier would be used to verify that prior use cases that were satisfied with the program are still functioning properly and as expected.

## 4.6 Acceptance Testing

Our approach to acceptance testing is to incorporate it into our sprints as we are using the agile approach. Before every sprint, we will lay out the goals to be met in regard to what the client wants and what the specifications state. At the end of each sprint we will review if these goals were met and if the appropriate functional and non-functional requirements are being met. When possible, we will verify functional requirements are being met through dry end-to-end testing as stated in the requirements section (from the requirements document). As for the rest of the functional and the non-functional requirements, those will be verified through regular meetings with the client. Consistent communication with the client will ensure that we remain on track and the final round of acceptance testing goes smoothly.

## 4.7 Security Testing

While security is not a major concern for this project, we will use basic admin tools for the databases to ensure that a user is registered and has password secured access to the system. Additionally UI fields will be protected so that no code or other malicious user can use the system.

## 4.8 Results

We don't have any results ready since we haven't tested anything yet. But we have a list of unit testing that we will be using in order to test our project.

| Testing units will be used | Use case test |
|---|---|
| JUnit | Testing Api's to unsure that we are able to use them in collecting our data. |
| Selenium or Appium | Used to simulate a specific user running and working within the frontend. |
| MYSQL Workbench or Terminal | Used to simulate the data that would be needed in order to run the assignment algorithm. |

***Figure 4.8.1: Unit testing***

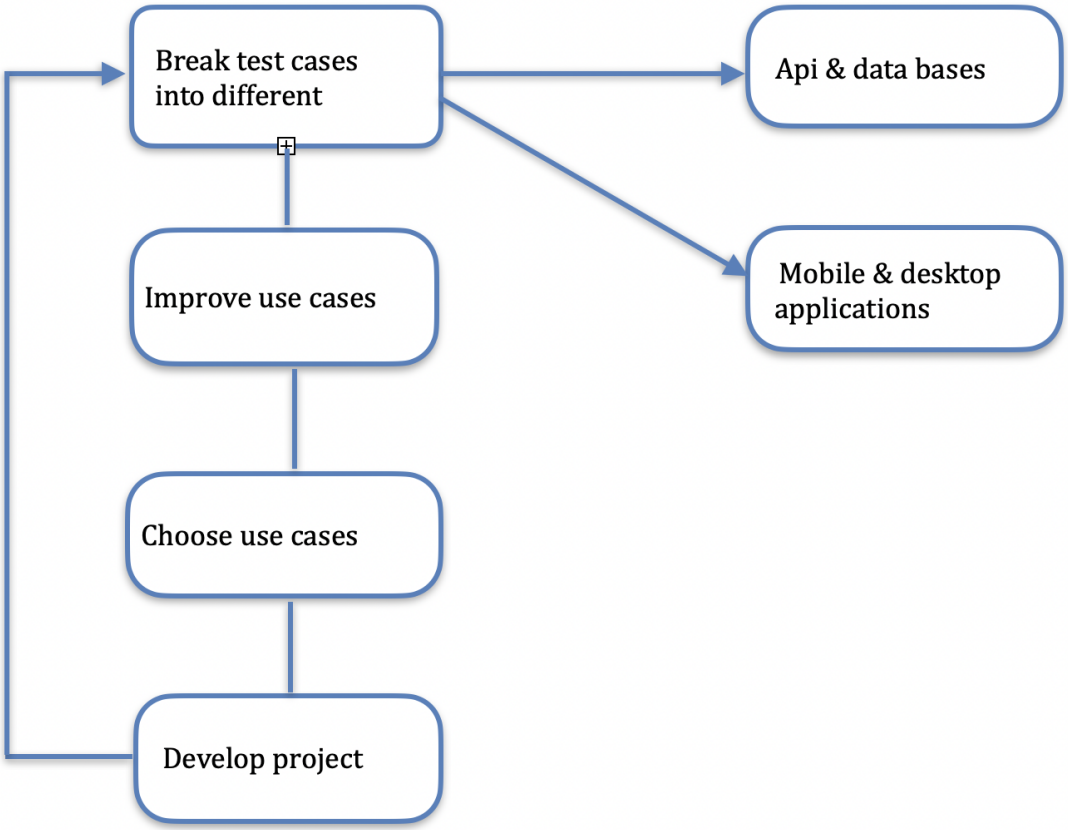


***Figure 4.8.2: Testing plan***

Our testing plan follows the test Driven Development software development process. In this process, requirements for the project are broken down into different individual test cases. This is so that everything that is created so far can be measurable. After this, code is redesigned for each test case so that the test case passes. Once the developer does this for a number of requirements, the tests are refactored and the process is completed again. This is so the tests get more detailed and the code gets much more secure.