Project Title: System Verification and Validation Plan for Greenway

Team #11, Roadkill
Priyansh Shah, shahp36
Utsharga Rozario, rozariou
Jash Mehta, mehtaj8
Bilal Shaikh, shaikb2
Pranay Kotian, kotianp
Sharjil Mohsin, mohsis2

November 7, 2022

1 Revision History

Date	Version	Notes
03-11-2022	1.0	Verification and Validation Plan Version 1

Contents

2   Symbols, Abbreviations and Acronyms

symbol	description
T	Test

This document shows the Verification and Validation plan for our software, which includes both the verification and implementation of the program. This document will show the plans for all critical documentation as well the system tests used for verfication of the code.

3 General Information

3.1 Summary

The goal of Greenway is to create an application that allows users to provide car information and a destination, in return the application supplies the amount of money it will take for the user to reach that destination using the most fuel-efficient route calculated by the app. The application uses real-time gas price data, information on mileage data, and terrain data to calculate and provide the amount of money it would take to reach a destination for users who drive many different types of gas vehicles; this allows them to use this app in many ways, such as, figuring out which gym may provide the most bang for their buck based on distance from their location.

3.2 Objectives

The objective of the document is to test the various subsystems identified in the system requirements and the systems design documents to ensure correctness and demonstrate adequate usability. Often times software may have bugs that is experienced by the end-user. This document will try to mitigate those issues by ensuring the underlying logic for the subsystems are performing as expected. This will be done through a suite of Unit Tests to test the functional and non-functional requirements. The code tested will be the underlying logic functions that interact with the database and send information back to the client since the rest of the system is front-end UI aspects.

3.3 Relevant Documentation

The relevant documents include:

- Software Requirment Specification Document

- Hazard Analysis Document

- Systems Design Document

4   Plan

This section includes the following sections, "Verification and Validation Team", "SRS Verification Plan", "Design Verification Plan", "Implementation Verfication Plan", "Automated Testing and Verification Tools", "Software Validation Plan".

4.1   Verification and Validation Team

The following project members are responsible for all procedures of the verification and validation processes including writing and executing tests:

- Utsharga Rozario

- Jash Mehta

- Priyansh Shah

- Bilal Shaikh

- Sharjil Mohsin

- Pranay Kotian

4.2   SRS Verification Plan

The following plans indicate what our team intends to do for SRS verification:

- **Review by teammates:** This plan involves going through each SRS and verifying whether each SRS is still valid within our given scope.

- **Review by stakeholders:** This plan involves going through each SRS with our stakeholder and how they view them in perspective to your usage.

- **Task based inspection:** This plan involves going through each SRS as a task, generally done by a teammate or stakeholder.

- **Checklist:** This plan involves verifying if each SRS has been met through our previously set checklist in our SRS document.

4.3 Design Verification Plan

The following plans indicate what our team intends to do for Design verification:

- **Review by teammates:** This plan involves going through the design of the project with our teammates through a high-fidelity prototype or a functional prototype, and verifying if the design is meets the expectation stated in the SRS.

- **Review by stakeholders:** This plan involves going through the design of the project with our stakeholders through a high-fidelity prototype or a functional prototype, and verifying if the design is meets the expectation stated in the SRS.

- **Task based inspection:** This plan involves going through each feature as a task in a high-fidelity prototype or a functional prototype, generally done by a teammate or stakeholder.

- **Checklist:** This plan involves verifying if each design requirements have been met through our previously set checklist in our SRS document.

4.4 Implementation Verification Plan

The following plans indicate what our team intends to do for Implementation verification:

- **Code Inspection:** This will be used for the test plans in section 5.1.1.

- **Static Analysis:** This will be used for the test plans in section 5.1.2 and section 5.1.3.

- **Non-functional Testing:** Non-functional Requirements test plans are written in details in section 5.2.

4.5 Automated Testing and Verification Tools

The following Automated Testing Tools:

- **Cypress:** Cypress is an automated testing tool from front-end development and has ability to take screenshots and video recordings of tests, allows developers to do asynchronous testing, and has In-browser debugging environment. Hence will be used to perform front-end testing on React.

- **Mocha:** Mocha is one of the oldest and most well-known testing frameworks for Node.js and hence will be used for our project. It has also evolved with Node.js and the JavaScript language over the years, such as supporting callbacks, promises, and async/await which we intend to use in our back-end.

- **Mongo Orchestration:** Mongo Orchestration is an HTTP server providing a RESTful interface to MongoDB process management running on the same machine and will be used to test our MongoDB database.

The following Verification Tools:

- **EsLint:** ESLint is a tool for identifying and reporting on patterns found in ECMAScript/JavaScript code, with the goal of making code more consistent and avoiding bugs and hence will used for linting our front-end (React) and back-end (Node.js).

4.6 Software Validation Plan

There are no available external data that can help validate the software. Hence, there are no plans for validation for now.

5 System Test Description

5.1 Tests for Functional Requirements

The areas of testing listed below are the following: Start screen, Map Interactions, Backend processing, Results Display. Start screen covers functional requirements 1-5 from the SRS as all those requirements are related to the inputs and functionality that exists in the program before any processing

or output is shown. Map Interaction covers functional requirements 6-8 as these requirements are concerned with functionality related to the map on display. Backend processing covers functional requirements 9-12 as these requirements are concerned with calculations and, external data collection and processing alike.

5.1.1 Start Screen

Start screen covers functional requirements 1-5 from the SRS as all those requirements are related to the inputs and functionality that exists in the program before any processing or output is shown

Preliminary Information Tests

1. preliminary-information-test-1

   Control: Automatic

   Initial State: No input in the start screen

   Input: Start Location

   Output: Status Message stating if the Location was accepted

   Test Case Derivation: The status message should be accepted the location as FR1 in the SRS requires the system to be able to take that as an input.

   How test will be performed: The testing framework will use a valid start location to input it into the field and ensure the status message is accepted the location.

2. preliminary-information-test-2

   Control: Automatic

   Initial State: No input in the start screen

   Input: Location of the Destination

   Output: Status Message stating if the Location was accepted

Test Case Derivation: The status message should be accepted the location as FR2 in the SRS requires the system to be able to take that as an input.

How test will be performed: The testing framework will use a valid location for a destination to input it into the field and ensure the status message is accepted the location.

3. preliminary-information-test-3

Control: Automatic

Initial State: No input in the start screen

Input: Car Details

Output: Status Message stating if the Car Information was accepted

Test Case Derivation: The status message should be accepted the Car Information as FR3 in the SRS requires the system to be able to take that as an input.

How test will be performed: The testing framework will use valid Car details for details to input it into the field and ensure the status message is accepted the Car Information.

4. preliminary-information-test-4

Control: Automatic

Initial State: Accepted Car details in the start screen

Input: Car mileage/fuel economy information

Output: Status Message stating if the Car mileage/fuel economy information was accepted

Test Case Derivation: The status message should be accepted the Car mileage/fuel economy information as FR4 in the SRS requires the system to be able to take that as an input.

How test will be performed: The testing framework will use valid Car mileage/fuel economy information for details to input it into the field and ensure the status message is accepted the Car mileage/fuel economy information.

5. preliminary-information-test-5

Control: Automatic

Initial State: Accepted Car details in the start screen

Input: Car information

Output: Status Message stating if the Car information was updated

Test Case Derivation: The status message should be accepted the Car information as FR5 in the SRS requires the system to be able to take that as an input.

How test will be performed: The testing framework will use valid Car information for details to input it into the field and ensure the status message is updated the Car information.

5.1.2 Map Interactions

Map Interaction covers functional requirements 6-8 as these requirements are concerned with functionality related to the map on display.

Map Tests

1. map-test-1

Control: Manual

Initial State: Start Screen finished

Input: None

Output: If the Map displays a route from start to end based on start screen input.

Test Case Derivation: The Map should display a route from start to end as specified in the Start screen, as FR6 States.

How test will be performed: The tester will ensure that the map has the correct route showing on it for a given start and end destination with a correct route to compare with.

2. map-test-2

   Control: Manual

   Initial State: Start Screen finished

   Input: None

   Output: If the Map displays all gas stations that it can possibly encounter along the route from a start to end destination.

   Test Case Derivation: The Map should display all gas stations along the route from start to end as specified in the Start screen, as FR7 States.

   How test will be performed: The tester will ensure that the map has all the gas stations along the route it shows and that they are being correctly displayed with a map to reference.

3. map-test-3

   Control: Manual

   Initial State: Start Screen finished

   Input: None

   Output: If correct gas prices come up when gas stations along the displayed route are clicked upon.

   Test Case Derivation: The Map should display all gas prices in the gas stations along the route from start to end as specified in the Start screen, as FR8 States.

   How test will be performed: The tester will click on all gas stations along the route and check that they are displaying gas prices in a range given beforehand to the tester.

5.1.3 Backend Processing

Backend processing covers functional requirements 9-12 as these requirements are concerned with calculations and, external data collection and processing alike.

Backend Tests

1. backend-test-1

   Control: Automatic

   Initial State: Start Screen finished

   Input: Route Details

   Output: If the route returned to the user is one that is optimizing the distance and elevation perfectly to reduce fuel costs.

   Test Case Derivation: The route displayed to the user should be the most fuel efficient route, as FR9 States.

   How test will be performed: The testing framework will have correct nodes pertaining to the most fuel efficient route and will compare them with nodes outputted by the back end to ensure that they are correct.

2. backend-test-2

   Control: Automatic

   Initial State: Start Screen finished

   Input: Route Details

   Output: If the database call on the backend returns correct elevation data for a route.

   Test Case Derivation: The most fuel efficient needs accurate elevation data and as such the FR10 is derived from the needs of FR9. And the test is designed to ensure that FR10 is fulfilled in the design.

   How test will be performed: The testing framework will test that the backend is getting the correct elevation data from the database with a reference dataset with correct elevation data.

3. backend-test-3

   Control: Automatic

   Initial State: Start Screen finished

   Input: Route Details

Output: If correct fuel consumption is given by backend for different parts of the route which have different elevation metrics.

Test Case Derivation: The application should be able to calculate accurate fuel consumption in different elevation conditions as distance is not the only metric for determining the most fuel efficient route. This test is made for FR11.

How test will be performed: The testing framework will test that the backend is calculating the correct fuel consumption data from the database with a reference dataset with correct fuel consumption data.

4. backend-test-4

   Control: Automatic

   Initial State: Start Screen finished

   Input: Route Details

   Output: If correct total cost is calculated by the backend for a certain route with a start and end decisions as the route details.

   Test Case Derivation: The application should be able to calculate accurate total cost for a certain route, as FR12 states.

   How test will be performed: The testing framework will test that the backend is to ensure that the a correct total cost is being returned with a correct total cost for a route as reference.

5.2   Tests for Nonfunctional Requirements

5.2.1   User Interface

Usability Tests

1. test-ui-1

   Type: Functional, Manual

   Initial State: N/A

   Input/Condition: N/A

   Output/Result: N/A

Test Case Derivation: As NFR1 states, the system must have a modern, minimalist user interface.

How test will be performed: This test will be verified using responses to Usability Survey Question 1 and 4. An average score of over 7 on Q1 and majority 'Yes' response to Q4 is considered a pass.

2. test-ui-2

   Type: Functional, Manual

   Initial State: N/A

   Input/Condition: N/A

   Output/Result: N/A

   Test Case Derivation: As NFR2 states, the system shall have a consistent style across the different pages (or interfaces) of the platform.

   How test will be performed: This test will be verified using responses to Usability Survey Question 2. A majority 'Yes' response is a pass.

3. test-ui-3

   Type: Functional, Manual

   Initial State: N/A

   Input/Condition: N/A

   Output/Result: N/A

   Test Case Derivation: As NFR3 states, the system shall have a user interface which is intuitive and easy to navigate.

   How test will be performed: This test will be verified using responses to Usability Survey Question 3. An average score of over a 7 is considered a pass.

4. test-ui-4

   Type: Functional, Manual

   Initial State: N/A

Input/Condition: N/A

Output/Result: N/A

Test Case Derivation: As NFR6 states, the system shall hide the details of its implementation from the user.

How test will be performed: This test will be verified using responses to Usability Survey Question 6. A majority 'No' response is a pass.

5. test-ui-5

Type: Functional, Manual

Initial State: N/A

Input/Condition: N/A

Output/Result: N/A

Test Case Derivation: As NFR7 states, the system shall hide the details of its implementation from the user.

How test will be performed: This test will be verified using responses to Usability Survey Question 7. A majority 'Yes' response is a pass.

5.2.2 User Experience

Quality of Life Tests

1. test-qol-1

Type: Functional, Manual

Initial State: N/A

Input/Condition: N/A

Output/Result: N/A

Test Case Derivation: As NFR16 states, the system shall have a simple installation process for all potential users.

How test will be performed: This test will be verified using responses to Usability Survey Question 8. A majority 'Yes' response is a pass.

5.2.3 Performance

Performance and Robustness Tests

1. test-pr-1

    Type: Functional, Manual

    Initial State: N/A

    Input/Condition: Starting location, ending location.

    Output/Result: Route and fuel consumption.

    Test Case Derivation: As NFR8 states, the system shall complete the route and fuel consumption calculation in a reasonable amount of time (<5 seconds).

    How test will be performed: Starting and ending location are entered. Calculate route button is selected. Amount of time to complete calculation is measured.

2. test-pr-2

    Type: Functional, Manual

    Initial State: N/A

    Input/Condition: N/A

    Output/Result: N/A

    Test Case Derivation: As NFR9 states, the system shall calculate fuel consumption as accurately as possible.

    How test will be performed: This test will be verified using responses to Usability Survey Question 11. A majority 'Yes' response is a pass.

3. test-pr-3

    Type: Functional, Manual

    Initial State: The application is running.

    Input/Condition: N/A

    Output/Result: N/A

Test Case Derivation: As NFR10 states, the system shall be available for use 24 hours per day, 365 days per year.

How test will be performed: This test will be verified using responses to Usability Survey Question 12. A majority 'No' response is a pass.

4. test-pr-4

Type: Functional, Manual

Initial State: N/A

Input/Condition: Invalid and edge case inputs for input fields.

Output/Result: User is notified that they need to modify input (error handling).

Test Case Derivation: As NFR11 states, the system shall be able to respond to invalid or unconventional inputs.

How test will be performed: User inputs invalid and edge cases inputs. System's response to these inputs shall be recorded.

5. test-pr-5

Type: Functional, Automated

Initial State: N/A

Input/Condition: N/A

Output/Result: N/A

Test Case Derivation: As NFR13 states, the system shall be able to handle an infinite amount of users.

How test will be performed: Script to run several instances of the program will stress test the system. System's response to a large number of users shall be recorded.

6. test-pr-6

Type: Functional, Manual

Initial State: N/A

Input/Condition: N/A

Output/Result: N/A

Test Case Derivation: As NFR14 states, the system shall operate as long as possible.

How test will be performed: This test will be verified using responses to Usability Survey Question 13. A majority 'No' response is a pass.

5.2.4 Technology

Technological Tests

1. test-t-1

   Type: Functional, Manual

   Initial State: N/A

   Input/Condition: N/A

   Output/Result: N/A

   Test Case Derivation: As NFR15 states, the system shall be able to run on all modern browsers.

   How test will be performed: This test will be verified using responses to Usability Survey Question 5. A majority 'Yes' response is a pass.

2. test-t-2

   Type: Functional, Manual

   Initial State: N/A

   Input/Condition: N/A

   Output/Result: N/A

   Test Case Derivation: As NFR17 states, the system shall be available to download for the public via GitHub.

   How test will be performed: This test will be verified using responses to Usability Survey Question 9. A majority 'Yes' response is a pass.

3. test-t-3

   Type: Functional, Manual

   Initial State: N/A

   Input/Condition: N/A

   Output/Result: N/A

   Test Case Derivation: As NFR20 states, the system shall be compatible to run between Windows, Mac, and Linux-based operating systems.

   How test will be performed: This test will be verified using responses to Usability Survey Question 10. A majority 'Yes' response is a pass.

5.2.5 Maintainability

Maintainability Tests

1. test-m-1

   Type: Functional, Manual

   Initial State: N/A

   Input/Condition: N/A

   Output/Result: N/A

   Test Case Derivation: As NFR18 states, he system's code shall be well documented and clear to be easily understandable for any developer maintaining the code.

   How test will be performed: Code inspection will be used to validate that all source code changes are documented and maintainable.

2. test-m-2

   Type: Functional, Manual

   Initial State: N/A

   Input/Condition: N/A

   Output/Result: N/A

Test Case Derivation: As NFR19 states, supporting documentation shall be available on GitHub, which includes a user's guide on how to use the application.

How test will be performed: This test will be verified using responses to Usability Survey Question 14. A majority 'Yes' response is a pass.

5.2.6 Legal

Legal Tests

1. test-l-1

   Type: Functional, Manual

   Initial State: N/A

   Input/Condition: N/A

   Output/Result: N/A

   Test Case Derivation: As NFR21 states, the system should comply with the Google Maps Platform Terms of Service.

   How test will be performed: Use Google Maps API to validate that our system is compliant with Google TOS.

5.3 Traceability Between Test Cases and Requirements

Requirement #	Description	Test ID(s)
FR1	The system shall allow the user a way to input a start location.	preliminary-information-test-1
FR2	The system shall allow the user a way to input a final destination.	preliminary-information-test-2
FR3	The system shall allow the user a way to select any car model and make.	preliminary-information-test-3
FR4	The system shall allow the user a way to input the fuel economy of their car.	preliminary-information-test-4
FR5	The system shall have a way to update available car models from either user input or external sources.	preliminary-information-test-5
FR6	The system shall have a map to show journey from start to final destination.	map-test-1
FR7	The system shall display the gas stations along the route.	map-test-2
FR8	The system shall display gas prices at stations on route to destination.	map-test-3
FR9	The system shall have a way to calculate the most fuel efficient route.	backend-test-1
FR10	The system shall have a way to collect elevation data along the suggested route.	backend-test-2
FR11	The system shall have a way to calculate fuel consumption on different terrain elevations.	backend-test-3
FR12	The system must calculate the total cost of the journey.	backend-test-4

Requirement #	Description	Test ID(s)
NFR1	The system shall have a modern minimalist user interface.	test-ui-1
NFR2	The system shall have a consistent style across the different pages (or interfaces) of the platform.	test-ui-2
NFR3	The system shall have a user interface which is intuitive and easy to navigate.	test-ui-3
NFR6	The system shall hide the details of its implementation from the user.	test-ui-4
NFR7	The system shall be accessible to users with visual impairments.	test-ui-5
NFR16	The system shall have a simple installation process for all potential users.	test-qol-1
NFR8	The system shall complete the route and fuel consumption calculation in a reasonable amount of time (¡5 seconds).	test-pr-1
NFR9	The system shall calculate fuel consumption as accurately as possible.	test-pr-2
NFR10	The system shall be available for use 24 hours per day, 365 days per year.	test-pr-3
NFR11	The system shall be able to respond to invalid or unconventional inputs.	test-pr-4
NFR13	The system shall be able to handle an infinite amount of users.	test-pr-5
NFR14	The system shall operate as long as possible.	test-pr-6
NFR15	The system shall be able to run on all modern browsers.	test-t-1
NFR17	The system shall be available to download for the public via GitHub.	test-t-2

Requirement #	Description	Test ID(s)
NFR20	The system shall be compatible to run between Windows, Mac, and Linux-based operating systems.	test-t-3
NFR18	The system's code shall be well documented and clear to be easily understandable for any developer maintaining the code.	test-m-1
NFR19	Supporting documentation shall be available on GitHub, which includes a user's guide on how to use the application.	test-m-2
NFR21	The system should comply with the Google Maps Platform Terms of Service.	test-l-1

6 Unit Test Description

This seciton outlines the unit testing for the function and non-functional requirements for Greenway. **This section is temporarily empty until the MIS is created.**

6.1 Unit Test Scope

6.2 Tests for Functional Requirements

[Most of the verification will be through automated unit testing. If appropriate specific modules can be verified by a non-testing based technique. That can also be documented in this section. —SS]

6.2.1 Module 1

[Include a blurb here to explain why the subsections below cover the module. References to the MIS would be good. You will want tests from a black box perspective and from a white box perspective. Explain to the reader how the tests were selected. —SS]

1. test-id1

   Type: [Functional, Dynamic, Manual, Automatic, Static etc. Most will be automatic —SS]

   Initial State:

   Input:

   Output: [The expected result for the given inputs —SS]

   Test Case Derivation: [Justify the expected value given in the Output field —SS]

   How test will be performed:

2. test-id2

   Type: [Functional, Dynamic, Manual, Automatic, Static etc. Most will be automatic —SS]

Initial State:

Input:

Output: [The expected result for the given inputs —SS]

Test Case Derivation: [Justify the expected value given in the Output field —SS]

How test will be performed:

6.3 Tests for Nonfunctional Requirements

[If there is a module that needs to be independently assessed for performance, those test cases can go here. In some projects, planning for nonfunctional tests of units will not be that relevant. —SS]

[These tests may involve collecting performance data from previously mentioned functional tests. —SS]

6.3.1 Module ?

1. test-id1

   Type: [Functional, Dynamic, Manual, Automatic, Static etc. Most will be automatic —SS]

   Initial State:

   Input/Condition:

   Output/Result:

   How test will be performed:

6.4 Traceability Between Test Cases and Modules

7 Appendix

7.1 Symbolic Parameters

The definition of the test cases will call for SYMBOLIC_CONSTANTS. Their values are defined in this section for easy maintenance.

7.2 Usability Survey Questions

1. On a scale of 1 - 10, how aesthetically pleasing is the user interface?

2. Does the user interface have a consistent graphical style throughout the system?

3. On a scale of 1 - 10, how easy to navigate is the user interface?

4. Does the user interface feel minimalist?

5. Does the system run on your favorite web browser?

6. Are details of implementation visible to a user?

7. Is it possible to use the application with a visual impairment?

8. Was the installation process easy?

9. Is the application available to download and install via GitHub.

10. Is the system able to run on your operating system (only answer if you are on a Windows, MacOS, Linux device). Answer 'Yes' if not applicable.

11. Is the system's calculation of fuel consumption accurate?

12. Is the system ever unavailable?

13. Have you experienced any unexpected crashes?

14. Is there a read me file in project Github repo that documents how to use the application?