Project Title: System Verification and Validation Plan for Greenway

Author Name

November 1, 2022

1 Revision History

Date	**Version**	**Notes**
Date 1	1.0	Notes
Date 2	1.1	Notes

Contents

List of Tables

[Remove this section if it isn't needed —SS]

List of Figures

[Remove this section if it isn't needed —SS]

2 Symbols, Abbreviations and Acronyms

symbol	**description**
T	Test

[symbols, abbreviations or acronyms – you can simply reference the SRS (Author, 2019) tables, if appropriate —SS]

This document ... [provide an introductory blurb and roadmap of the Verification and Validation plan —SS]

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
- Systems Design Document
- Hazard Analysis 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

[You, your classmates and the course instructor. Maybe your supervisor. You shoud do more than list names. You should say what each person's role is for the project. A table is a good way to summarize this information. —SS]

4.2 SRS Verification Plan

[List any approaches you intend to use for SRS verification. This may just be ad hoc feedback from reviewers, like your classmates, or you may have something more rigorous/systematic in mind.. —SS]

[Remember you have an SRS checklist —SS]

4.3 Design Verification Plan

[Plans for design verification —SS]

[The review will include reviews by your classmates —SS]

[Remember you have MG and MIS checklists —SS]

4.4 Implementation Verification Plan

[You should at least point to the tests listed in this document and the unit testing plan. —SS]

[In this section you would also give any details of any plans for static verification of the implementation. Potential techniques include code walkthroughs, code inspection, static analyzers, etc. —SS]

4.5 Automated Testing and Verification Tools

[What tools are you using for automated testing. Likely a unit testing framework and maybe a profiling tool, like ValGrind. Other possible tools include a static analyzer, make, continuous integration tools, test coverage tools, etc. Explain your plans for summarizing code coverage metrics. Linters are another important class of tools. For the programming language you select, you should look at the available linters. There may also be tools that verify that coding standards have been respected, like flake9 for Python. —SS]

[The details of this section will likely evolve as you get closer to the implementation. —SS]

4.6 Software Validation Plan

[If there is any external data that can be used for validation, you should point to it here. If there are no plans for validation, you should state that here. —SS]

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. test-id1

   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. test-id2

   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. test-id3

   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. test-id4

   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. test-id5

   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. test-id1

   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. test-id2

   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. test-id3

   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. test-id1

   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. test-id2

   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. test-id3

   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. test-id4

   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

[The nonfunctional requirements for accuracy will likely just reference the appropriate functional tests from above. The test cases should mention reporting the relative error for these tests. —SS]

[Tests related to usability could include conducting a usability test and survey. —SS]

5.2.1 Area of Testing1

Title for Test

1. test-id1

   Type:

   Initial State:

   Input/Condition:

   Output/Result:

   How test will be performed:

2. test-id2

   Type: Functional, Dynamic, Manual, Static etc.

   Initial State:

   Input:

   Output:

   How test will be performed:

5.2.2 Area of Testing2

...

5.3 Traceability Between Test Cases and Requirements

[Provide a table that shows which test cases are supporting which requirements. —SS]

6 Unit Test Description

[Reference your MIS and explain your overall philosophy for test case selection. —SS] [This section should not be filled in until after the MIS has been completed. —SS]

6.1 Unit Testing Scope

[What modules are outside of the scope. If there are modules that are developed by someone else, then you would say here if you aren't planning on verifying them. There may also be modules that are part of your software, but have a lower priority for verification than others. If this is the case, explain your rationale for the ranking of module importance. —SS]

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:

3. ...

6.2.2 Module 2

...

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:

2. test-id2

   Type: Functional, Dynamic, Manual, Static etc.

   Initial State:

   Input:

   Output:

   How test will be performed:

6.3.2 Module ?

...

6.4 Traceability Between Test Cases and Modules

[Provide evidence that all of the modules have been considered. —SS]

References

Author Author. System requirements specification. https://github.com/..., 2019.

7 Appendix

This is where you can place additional information.

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?

[This is a section that would be appropriate for some projects. —SS]

Appendix — Reflection

The information in this section will be used to evaluate the team members on the graduate attribute of Lifelong Learning. Please answer the following questions:

1.

2.