Software Test Description (STD)

CMSC 447 Group 4

Project Vesta

1 Scope

1.1 Identification

This document refers to the software package known as Vesta.

1.2 System overview

Vesta is a web based application that will be utilized by an average internet user. This web application is intended to allow users to acquire potential living locations based on abstract specifications regarding the area. To garner said information, the user will specify quality of life attributes desired within a defined radius. These characters may include meteorological data (ie average temperature, average weather conditions), geographical information, etc. The application will compare the inputted information to specified open source databases to determine the locations that fit the criteria. Once the locations have been determined by the software on the back-end, they will be displayed to the user.

1.3 Document overview

This document will specify the process and specifications in which Vesta will be tested.

2 Referenced documents

This documents references the requirements outlined in the Software Requirements Specification.

3 Test preparations

There are no special hardware or software preparations required to perform the following tests.

4 Test descriptions

4.1 Unit Tests

In order to verify that both the front end and the backend work as intended, small unit tests will be written and run against each CSU. They will be written and retested throughout development.

4.1.1 Front End Unit Tests

Each button, box, slider, page, and front end web component will be tested with a small, manual unit test to verify that it is working correctly.

4.1.1.1 Requirements addressed

These tests will address front end functionality, which is a prerequisite for many other requirements. It also addresses hardware and software requirements, indicating that the website is successfully working for our chosen hardware and browsers.

4.1.1.2 Prerequisite conditions

There are no prerequisites for these tests.

4.1.1.3 Test inputs

The test input will be selected manually by a tester. They will select whatever HTML, JavaScript, webpage, or front end CSU they are currently developing.

4.1.1.4 Expected test results

Each CSU selected or interacted with should produce its desired result on the webpage.

4.1.1.5 Criteria for evaluating results

The results of these tests will be visually evaluated by hand. The tester will confirm that each CSU produces the correct result when interacted with.

4.1.1.6 Test procedure

1. Interact with CSU.
2. Observe result.

4.1.2 Backend Unit Tests

In order to verify that each component of the backend works correctly, small unit tests in python will be written to verify that each function, library, API, object, and class works correctly.

4.1.2.1 Requirements addressed

These tests will address backend functionality, which is a prerequisite for many other requirements. It also addresses hardware and software requirements, indicating that the code is successfully working for our chosen hardware.

4.1.2.2 Prerequisite conditions

There are no prerequisites for these tests.

4.1.2.3 Test inputs

The test input will be selected manually by the unit test writer. Test input may include simulated user requests, city information, HTML queries, etc.

4.1.2.4 Expected test results

Each CSU tested should produce its desired result. Results may include database queries, city information, a list of cities, etc.

4.1.2.5 Criteria for evaluating results

The results of these tests can be evaluated by hand or with a script, depending on the CSU. Each script should compare its results to a known good value to evaluate success.

4.1.2.6 Test procedure

3. Run test script.
4. Compare results.

4.2 Test User Inputs

In order to verify that the web interface and backend database poller work, each individual user input parameter must be tested by choosing a variety of ranges and confirming that the returned cities match the inputted numbers.

The following user input tests will address requirements 3.2.1.I and 3.2.1.II from the SRS.

4.2.1 Test Population, Political Color, Climate, and Cost of Living Buttons

In order to verify that each parameter/button works, all buttons in all categories (population, political color, climate, and cost of living) will be pressed and submitted, and the resulting cities will be verified to match the chosen input. In addition, a variety of combinations of buttons will be tested to verify that the resulting cities match the chosen combinations.

4.2.1.1 Requirements addressed

This test will address requirement 3.2.1.I.

4.2.1.2 Prerequisite conditions

Before this test can be run, the front end, middle end, and back end must all be tested, working, and integrated.

4.2.1.3 Test inputs

The test input will be selected manually by a tester. They will select from a number of buttons for each parameter. The only valid inputs to the backend are the specified values associated with each button, so no edge cases or invalid type tests are required.

4.2.1.4 Expected test results

For each button or combination of buttons selected, only the appropriately matched cities should be returned. If an invalid or impossible combination of buttons are pressed, a “Sorry, no cities match your criteria” message should be returned and displayed.

4.2.1.5 Criteria for evaluating results

The results of these tests will be evaluated by hand. If the resulting cities have parameters that match the selected buttons, then the tests has passed. If any of the cities have parameters that fall outside of the specified range, then the test has failed. The back end needs to be re-examined to determine why it is returning cities that do not match the input parameters.

4.2.1.6 Test procedure

5. Select parameter button(s).
6. Press the submit button.
7. Visually verify that the returned cities match the selected button(s).

4.2.2 Test Population, Political Color, Climate, and Cost of Living Boxes and Sliders

In order to verify that each parameter works for a variety of ranges, all advanced input boxes in all categories (population, political color, climate, and cost of living) will be manipulated and submitted, and the resulting cities will be verified to match the chosen input. In addition, a variety of combinations of inputs will be tested to verify that the resulting cities match the chosen combinations.

4.2.2.1 Requirements addressed

This test will address requirement 3.2.1.I.

4.2.2.2 Prerequisite conditions

Before this test can be run, the front end, middle end, and back end must all be tested, working, and integrated.

4.2.2.3 Test inputs

The test input will be selected manually by a tester. They will input a wide range of numbers for each parameter. In addition, they will input edge cases like 0, very large numbers, negative numbers, and invalid inputs to test the error catching capabilities of both the front end and the back end.

4.2.2.4 Expected test results

For each range selected for each parameter or combination of parameters, only the appropriately matched cities should be returned. If an invalid or impossible combination of parameters are selected, a "Sorry, no cities match your criteria" message should be returned and displayed.

4.2.2.5 Criteria for evaluating results

The results of these tests will be evaluated by hand. If the resulting cities have parameters that match the selected buttons, then the tests has passed. If any of the cities have parameters that fall outside of the specified range, then the test has failed. The back end needs to be re-examined to determine why it is returning cities that do not match the input parameters.

4.2.2.6 Test procedure

8. Select parameter(s) ranges.
9. Press the submit button.
10. Visually verify that the returned cities match the selected parameter(s).

4.3 Test Backend Effectiveness and Efficiency

In order to verify that the back end is performing both accurately and at a reasonable speed, two automated tests will be performed on the back end alone.

The following user input tests will address requirements 3.2.1.III and 3.2.1.IV from the SRS.

4.3.1 Time Backend Operations

In order to verify that the back end performs its lookup operations in at least 60 seconds, a script will be run to test the back end with a variety of inputs. Each call of the backend will be timed, and after all tests have been run, the minimum, maximum, and average time the backend took to finish will be reported.

4.3.1.1 Requirements addressed

This test will address requirement 3.2.1.III.

4.3.1.2 Prerequisite conditions

Before this test can be run, all unit tests for the backend must have been run and passed successfully. In addition, the backend must be finished and fully integrated.

4.3.1.3 Test inputs

The test input will be generated randomly by a script. 100 randomly generated user inputs (that fall within acceptable ranges) will be generated and fed into the backend. In addition, edge cases, like very small and very large parameter ranges will be tested and reported separately.

4.3.1.4 Expected test results

Requirement 3.2.1.III states that the system should take no longer than 60 seconds to run. Most of this time will be spent on the backend. If our backend on average takes longer than 60 seconds, we will need to reevaluate our backend algorithms.

4.3.1.5 Criteria for evaluating results

The minimum, maximum, and average time to run the backend will be returned. The average time to run should be less than 60 seconds.

4.3.1.6 Test procedure

11. Run test script.
12. Confirm running times.

4.3.2 Confirm Backend Accuracy

In order to verify that the back end performs its database queries and set operations successfully and accurately, a test script will run a variety of inputs against the backend and confirm the results. Inputs with known outputs will need to be constructed, and the results will need to be stored. The test script will call the backend with each known input, and verify the results with the known output. The test script will report the total number of correct evaluations.

4.3.2.1 Requirements addressed

This test will address requirement 3.2.1.IV.

4.3.2.2 Prerequisite conditions

Before this test can be run, all unit tests for the backend must have been run and passed successfully. In addition, the backend must be finished and fully integrated.

4.3.2.3 Test inputs

Both the test input and output will be constructed by hand and stored in a file. The test inputs will be a variety of ranges of each parameters.

4.3.2.4 Expected test results

The backend should evaluate at 95% accuracy. For any invalid parameter combinations, it should report an invalid combination result. Any resulting accuracies below 95% will require reevaluation of the backend algorithms.

4.3.2.5 Criteria for evaluating results

The test script will return a accuracy percentage. The percentage should be greater than or equal to 95%.

4.3.2.6 Test procedure

13. Run test script.
14. Confirm accuracy.

5 Requirements traceability

Requirements	Test
3.2.1.I	4.2
3.2.1.II	4.2
3.2.1.III	4.3.1
3.2.1.IV	4.3.2
3.3.1	4.1.2
3.9	4.1
3.10.1	4.1
3.10.2	4.1.1, 4.2