Deliverable 2

Introduction

Since our previous Deliverable, our group decided to switch the HFOSS project we focused on. We decided to switch to OpenAQ because STEM included many problems for our group including:

- Issues finding all packages of the source code.
- Building what little of the source code we actually could obtain.
- Not being able to successfully build the project on multiple machines running Ubuntu Linux.

With a lack of running software to develop or run tests on, our group could not continue making progress within the overall scheme of the project. Therefore, our group has decided to work with OpenAQ as our new #1 candidate HFOSS project. OpenAQ includes:

- All source code needed to properly build from scratch.
- Guides to build the project from the source code.
- Project being written in JavaScript.
- Successfully built and inspected on a few machines already (more on this later).

Accomplishments

During project development for Deliverable 2, our group has accomplished writing a test plan, which includes the hierarchy of how we are to proceed with testing different aspects of OpenAQ throughout the rest of the semester. Our team has also written how we would test certain operations within OpenAQ, 5 out of 25 is what we wrote so far.

Learning Outcomes

Our team has learned how to develop a test plan for a detailed method of testing different parts of the software from bottom to top. The work done helped develop a further understanding of how to write test cases for a complex software which includes multiple components and interfaces.

Problems Experienced

Even though the guides for building OpenAQ appear to be well written, the steps to successfully building OpenAQ sometimes appear to be out of order. At particular steps, a software package would not install because prerequisite software was instructed to be installed afterwards. Installing all software components when needed resolved many of these issues.

Going forward

The next goal is to keep uncovering more information of how the software functions in order to develop further relevant test cases. As our team learns more software engineering principles to

apply, we can further strategize our resources efficiently towards completing future Deliverables for OpenAQ.

Below is the Test Plan for the OpenAQ project.

The Openaq project is an open-source project that gets air quality data from different cities and countries and makes them available to the public. The pollutants Openaq tracks are PM10, PM2.5, sulfur dioxide (SO2), carbon monoxide (CO), nitrogen dioxide (NO2), ozone (O3), and black carbon (BC). It gathers this data from official-level outdoor air quality source. The readings must meet the criteria of Openaq-fetch in order to be added to the database.

This is the test plan for openaq-fetch, the main data ingest pipeline for the OpenAQ project.

The Testing Process

During the Testing of OpenAQ-fetch, an order of phases will occur. The first will be Unit-Testing where individual operations are tested with a set of test cases to guarantee the operations meet required specifications. The second phase will be Component-Testing where multiple operations are tested together using an interface (forming a component) to guarantee that a particular set of criteria is met. The last phase will be the overall System-Testing in which a select set of components are integrated together and then tested for expected conditions to be met from the software.

Requirements Traceability

The software is required to function as expected which entails:

- The software performs under expected conditions
- The software receives all necessary parameters within every operation during any set of tests
- The interactions between the components during system testing must match with what the oracle predicts.
- A report is made after a set of automated tests have completed.

Tested Items

An operation within OpenAQ known as, MakeSourcefromData(), which adds air quality data to the database. The operation uses parameter unit values such as coordinates, dates, value of ppv (particles per volume), and units of measurement.

Testing Schedule

The schedule will be as follows:

- The first set of Unit-Tests will occur during the development/understanding of code for a couple of weeks from October 1st to October 29th.

- The next set of testing will include the development and execution of an automated testing framework for 25 test cases from October 29th to November 12th.
- The final set of testing involves fault testing where components within the framework will be tested with malicious input for expected results from November 12th to November 19th.

Test Recording Procedures

The software will be running tests from a script file that records the results of the tests into text file with a particular format depending on what part of the system is being tested. The results are then compared to the oracle's text file for correctness & expected output. Lastly, a report is made onto an HTML file of the results from testing our project.

Hardware and Software Requirements

Software & Hardware Requirements include:

* Windows, Mac, Linux system

* Approx 200MB of free disk space

* At least 1.8GB free memory

* 2 core CPU is recommended

* node.js >= 8.6

* Either of:

  * docker or

  * postgresql >= 10 with postgis >= 2 extensions.

Constraints

Constraints include:

- **Lack of Time** due to the testing portion of the project being limited to 2 months.
- **Lack of Staff** due to amount of students working on the project being limited to 3 members with diverse tasks.
- **Lack of Knowledge** due to every student working on the project still learning methods for testing.

System Tests

Test cases will include tests ran through scripts upon the operations of OpenAQ-fetch for pass/fail conditions. Tests will be ran on the inputs for several particular operations for stability, and expected output. Tests will be ran on the outputs of several operations for correctness. Tests

will be ran on order of execution for the system components as well as the overall system for expected goal completion.

Each test case is created in javascript and run from the command line using: node index.js --dryrun 'test'

--dryrun, -d Run the fetch process but do not attempt to save to the database and instead print to console, useful for testing.

Test case 1: Input: parameter: 'pm25', unit: 'ppq', //Error: Should be ppm value: 10, location: 'test1', coordinates: { latitude: -20, longitude: 34 }, country: 'US', city: 'Test', sourceName: 'Test', mobile: false, sourceType: 'government', attribution: [{ name: 'test', url: 'http://test.case' }], averagingPeriod: { value: 1, unit: 'hours' }

Expected Output: 'instance.unit is not one of enum values: µg/m³,ppm'

Test case 2: Input: parameter: '01', //Error: Not a valid parameter unit: 'ppm', value: 10, location: 'test2', coordinates: { latitude: -20, longitude: 34 }, country: 'US', city: 'Test', sourceName: 'Test', mobile: false, sourceType: 'government', attribution: [{ name: 'test', url: 'http://test.case' }], averagingPeriod: { value: 1, unit: 'hours' }

Expected Output: 'instance parameter is not one of enum values: pm25,pm10,no2,so2,o3,co,bc'

Test case 3: Input: parameter: 'no2', unit: 'ppm', value: Q, //Error: Value must be a number location: 'test3', coordinates: { latitude: -20, longitude: 34 }, country: 'US', city: 'Test', sourceName: 'Test', mobile: false, sourceType: 'government', attribution: [{ name: 'test', url: 'http://test.case' }], averagingPeriod: { value: 1, unit: 'hours' }

Expected Output: 'instance value is not of a type(s) number'

Test case 4: Input: parameter: 'pm25', //Ideal case unit: 'ppm', value: 10, location: 'test4', coordinates: { latitude: -20, longitude: 34 }, country: 'US', city: 'Test', sourceName: 'Test', mobile: false, sourceType: 'government', attribution: [{ name: 'test', url: 'http://test.case' }], averagingPeriod: { value: 1, unit: 'hours' }

Expected Output: 'Data source accepted.'

Test case 5: Input: parameter: 'pm25', unit: 'ppm', value: 10, location: 'test5', coordinates: { latitude: -20, longitude: 34 }, country: , //Error: Country field required city: 'Test', sourceName: 'Test', mobile: false, sourceType: 'government', attribution: [{ name: 'test', url: 'http://test.case' }], averagingPeriod: { value: 1, unit: 'hours' }

Expected Output: 'instance requires property "country"'