Testing Specifications

Real-time Geospatial Data Processor and Visualiser

Client: Werner Raath

COEUS

Members:
Nsovo Baloyi 12163262
Maluleki Nyuswa 13040686
Keletso Molefe 14222583
Kamogelo Tswene 12163555

09 September 2016
v0.2

Contents

Document History **2**

1 Introduction **3**

1.1 Purpose . . . 3
1.2 Scope . . . 4
1.3 Test Environment . . . 4
1.4 Assumptions and Dependencies . . . 4
1.4.1 Assumptions . . . 4
1.4.2 Dependencies . . . 5

2 Test Items **5**

3 Functional Features to be Tested **5**

3.1 Features Tested . . . 5
3.1.1 Disasters: back-end . . . 5
3.1.2 Weather: back-end . . . 5

4 Test Cases **6**

4.1 Test Case 1: Query All Earthquakes . . . 6
4.1.1 Condition 1: the back-end server must be running . . . 6
4.1.2 Condition 2: the correct route must be called (/earthquakes) . . . 6
4.1.3 Objective: . . . 6
4.1.4 Input . . . 6
4.1.5 Outcome . . . 6
4.2 Test Case 2: Time Query - In between Earthquakes . . . 6
4.2.1 Condition 1: the back-end server must be running . . . 6
4.2.2 Condition 2: the correct route must be called (/inbetween_earthquakes:first/:second) . . . 6
4.2.3 Condition 3: the route parameters (:first, :second)must be in epoch time, as a integer . . . 6
4.2.4 Objective: . . . 6
4.2.5 Input . . . 7
4.2.6 Outcome . . . 7
4.3 Test Case 3: Time Query - Less than Earthquakes . . . 7
4.3.1 Condition 1: the back-end server must be running . . . 7
4.3.2 Condition 2: the correct route must be called (/lessthan_earthquakes/:first) 7

4.3.3 Condition 3: the route parameter (:first) must be in epoch time, as a integer 7
4.3.4 Objective: 7
4.3.5 Outcome 7
4.4 Test Case 4: Time Query - Greator than Earthquakes 7
4.4.1 Condition 1: the back-end server must be running . . . 7
4.4.2 Condition 2: the correct route must be called (/greatorthan_earthquakes/:first)
4.4.3 Condition 3: the route parameter (:first) must be in epoch time, as a integer 7
4.4.4 Objective: 7
4.4.5 Outcome 7
4.5 Test Case 5: Query All Fires 7
4.5.1 Condition 1: the back-end server must be running . . . 7
4.5.2 Condition 2: the correct route must be called (/fires) . 7
4.5.3 Objective: 7
4.5.4 Input 8
4.5.5 Outcome 8
4.6 Test Case 6: Time Query - In between Fires 8
4.6.1 Condition 1: the back-end server must be running . . . 8
4.6.2 Condition 2: the correct route must be called (/inBetweenfires/:first/:second) 8
4.6.3 Condition 3: the route parameters (:first, :second)must be in epoch time, as a integer 8
4.6.4 Objective: 8
4.6.5 Input 8
4.6.6 Outcome 8
4.7 Test Case 7: Time Query - Less than Fires 8
4.7.1 Condition 1: the back-end server must be running . . . 8
4.7.2 Condition 2: the correct route must be called (/lessThanFires/:first) 8
4.7.3 Condition 3: the route parameter (:first) must be in epoch time, as a integer 8
4.7.4 Objective: 8
4.7.5 Outcome 9
4.8 Test Case 8: Time Query - Greator than Fires 9
4.8.1 Condition 1: the back-end server must be running . . . 9
4.8.2 Condition 2: the correct route must be called (/greaterThanFires/:first) 9
4.8.3 Condition 3: the route parameter (:first) must be in epoch time, as a integer 9
4.8.4 Objective: 9
4.8.5 Outcome 9
4.9 Test Case 9: Query All Weather Data 9

4.9.1 Condition 1: the back-end server must be running . . . 9
4.9.2 Condition 2: the correct route must be called (/getweather) 9
4.9.3 Objective: 9
4.9.4 Outcome 9

5 Item Pass/Fail Criteria 9

6 Test Deliverables 10

7 Detailed Test Results 10
7.1 Overview of Test Results 10
7.2 Functional Requirements Test Results 11
7.2.1 Test Case 1 (TC 4.1) 11
7.2.2 Test Case 2 (TC 4.2) 11
7.2.3 Test Case 3 (TC 4.3) 11
7.2.4 Test Case 4 (TC 4.4) 12
7.2.5 Test Case 5 (TC 4.5) 12
7.2.6 Test Case 6 (TC 4.6) 12
7.2.7 Test Case 7 (TC 4.7) 12
7.2.8 Test Case 8 (TC 4.8) 12
7.2.9 Test Case 9 (TC 4.9) 13

8 Other 13

9 Conclusions and Recommendations 13

Document History

Version	Date	Changed By	Summary
v0.1	29 July 2016	Nsovo Baloyi Maluleki Nyuswa Keletso Molefe Kamogelo Tswene	First Draft
v0.2	9 September 2016	Nsovo Baloyi Maluleki Nyuswa Keletso Molefe Kamogelo Tswene	Second Draft

1 Introduction

This documentation is the testing documentation for the Geospatial Data Processor and Visualiser project. It outlines the entire testing plan and it documentation process. The documentation firstly establishes the scope, thereafter the testing environment is discussed including all the relevant assumptions and dependencies made during the testing process. The test items, functional features that were tested and the individual test cases are then discussed. Test conducted are then specified according to whether they passed or failed. Test deliverables are then clearly tabulated followed by detailed test results, and finally conclusions and recommendations are noted.

1.1 Purpose

This document combines the unit test plan and report into a single coherent artefact. The Geospatial Data Processor and Visualiser system aims to collects geospatial data from third party API's, persist the data on a database and thereafter visualise such in real-time through a web-interface. The system focuses mainly on natural disaster and weather visualisation.

Software testing forms an integral part of software design, intended to empirically verify whether the software being developed conforms to specifications. Unit testing test a piece of code in isolation against requirements and when done constructively it contributes to code flexibility and and reusability. Black-box testing was used as the tests were developed to contract specification. Test-driven devolopment(TTD) approach was followed during the project as it forms part of the agile development technique.

The benefits of unit testing include:

1 Reduced system failure risk.

2 Rapid feedback on developed components.

3 Reduced cost due to

- less time spent on bug fixes
- reduced integration problems, and
- lower manual testing costs

4 Improved Maintainability due to unit testing

5 Improved Reusability leading to less code being developed and maintained

1.2 Scope

The scope of this document is structured as follows. The features that are considered for testing are listed in section 3. Individual tests that were identified from the requirements are discussed in detail in section 4. Furthermore, this document outlines the test environment and the risks involved in the testing approaches that were followed. Assumptions and dependencies of this test plan will also be mentioned. Section 7.1 and 9 outlines, discusses and concludes on the results of the tests, respectively.

1.3 Test Environment

This section of the document outlines the environment that existed during the unit testing.

- Programming Languages: AngularJS was primarily used during the development stage on the front-end, and NodeJS and ExpressJS on the back-end.
- Testing Frameworks: Mocha was the primary testing framework used on the back-end. It was chosen for it reach features and ease on testing NodeJS applications. Chai is a TDD assertion library for node and the browser, and was used for it delightfulness for pairing with any testing framework. SinnonJS was used to create mock objects and testing environment.
- Coding Environment IntelliJ version 9.0 Ultimatum Edition by Jetbrains was the coding IDEA of chose for the project development.
- Operating System All COEUS team members had the Windows 10 operating system by Microsoft installed on their laptops during the project development stage.
- Internet Browsers The web-interface was tested to execute accurately on GOOGLE Chrome, Mozilla Firefox and Internet Explorer web browser.

1.4 Assumptions and Dependencies

1.4.1 Assumptions

One of the few assumptions made during the testing was the internet download speed of not less than 5Mb/s. This was important for testing the performance requirements of the system.

1.4.2 Dependencies

2 Test Items

Black box testing was implemented for all items tested due to it simplicity and ability for requirements testing.

Items tested include:

- Front-End Web Interface
- Back-End Servers
  - Disaster
    - Earthquakes
    - Fires
  - Weather
- Rabbit MQ
- User Authentication

3 Functional Features to be Tested

3.1 Features Tested

3.1.1 Disasters: back-end

- Earthquakes
- Fires
- Weather - back-end

3.1.2 Weather: back-end

- weather

The Earthquakes and Fires disasters were tested against requirements to ascertain that the correct data is returned at the specified format (json object)for each function call. The response body should be an array, and the status should be 200. The parameters (start and end date) passed in to the time-query function should be in epoch time represented as an integer. All the test summarized on the table below are located the following folder; `https://github.com/Coeus2016/visualizer-server/test`.

Table 1: Summary to different test cases

Feature ID	RDS Source	Summary	Test Case ID
1	controllers.disasters.earthquakes.test	Passed	E01
2	controllers.disasters.earthquakes.test	Passed	E02
3	controllers.disasters.earthquakes.test	Passed	E03
4	controllers.disasters.earthquakes.test	Passed	E04
5	controllers.disasters.fires.test	Passed	F01
6	controllers.disasters.fires.test	Passed	F02
7	controllers.disasters.fires.test	Passed	F03
8	controllers.disasters.fires.test	Passed	F04
9	weather.test	Passed	W01

4 Test Cases

4.1 Test Case 1: Query All Earthquakes

4.1.1 Condition 1: the back-end server must be running

4.1.2 Condition 2: the correct route must be called (/earthquakes)

4.1.3 Objective:

The purpose of this test is to validate whether all earthquakes data residing on the database is returned as a json object

4.1.4 Input

4.1.5 Outcome

4.2 Test Case 2: Time Query - In between Earthquakes

4.2.1 Condition 1: the back-end server must be running

4.2.2 Condition 2: the correct route must be called (/inbetween_earthquakes:first/:s

4.2.3 Condition 3: the route parameters (:first, :second)must be in epoch time, as a integer

4.2.4 Objective:

This test tests whether all earthquakes that are between specified first and second epochTime are returned, as a json object

4.2.5 Input

4.2.6 Outcome

4.3 Test Case 3: Time Query - Less than Earthquakes

4.3.1 Condition 1: the back-end server must be running

4.3.2 Condition 2: the correct route must be called (/lessthan_earthquakes/:first)

4.3.3 Condition 3: the route parameter (:first) must be in epoch time, as a integer

4.3.4 Objective:

The purpose of this test is to validate whether all earthquakes data before the specified epochTime is returned, as a json object

4.3.5 Outcome

4.4 Test Case 4: Time Query - Greator than Earthquakes

4.4.1 Condition 1: the back-end server must be running

4.4.2 Condition 2: the correct route must be called (/greatorthan_earthquakes/:first

4.4.3 Condition 3: the route parameter (:first) must be in epoch time, as a integer

4.4.4 Objective:

The purpose of this test is to validate whether all earthquakes data before the specified epochTime is returned, as a json object

4.4.5 Outcome

4.5 Test Case 5: Query All Fires

4.5.1 Condition 1: the back-end server must be running

4.5.2 Condition 2: the correct route must be called (/fires)

4.5.3 Objective:

The purpose of this test is to validate whether all fires data residing on the database is returned as a json object

4.5.4 Input

4.5.5 Outcome

4.6 Test Case 6: Time Query - In between Fires

4.6.1 Condition 1: the back-end server must be running

4.6.2 Condition 2: the correct route must be called (/inBetween-fires/:first/:second)

4.6.3 Condition 3: the route parameters (:first, :second)must be in epoch time, as a integer

4.6.4 Objective:

This test tests whether all fires that are between specified first and second epochTime are returned, as a json object

4.6.5 Input

4.6.6 Outcome

4.7 Test Case 7: Time Query - Less than Fires

4.7.1 Condition 1: the back-end server must be running

4.7.2 Condition 2: the correct route must be called (/lessThanFires/:first)

4.7.3 Condition 3: the route parameter (:first) must be in epoch time, as a integer

4.7.4 Objective:

The purpose of this test is to validate whether all fires data before the specified epochTime is returned, as a json object

4.7.5 Outcome

4.8 Test Case 8: Time Query - Greator than Fires

4.8.1 Condition 1: the back-end server must be running

4.8.2 Condition 2: the correct route must be called (/greaterThanFires/:first)

4.8.3 Condition 3: the route parameter (:first) must be in epoch time, as a integer

4.8.4 Objective:

The purpose of this test is to validate whether all fires data before the specified epochTime is returned, as a json object

4.8.5 Outcome

4.9 Test Case 9: Query All Weather Data

4.9.1 Condition 1: the back-end server must be running

4.9.2 Condition 2: the correct route must be called (/getweather)

4.9.3 Objective:

The purpose of this test is to validate whether all weather data queried is returned, as a json object

4.9.4 Outcome

5 Item Pass/Fail Criteria

Each item tested must meet a certain criteria in order to pass. These criteria are as follows:

- Front-End Web Interface
- Back-End Servers
  - The server should start within 4000ms
  - The server should not shut-down on a failed request
  - Disaster
    - Earthquakes

        - Returns a json object
        - The query should execute within 6000ms
      - Fires
        - Returns a json object
        - The query should execute within 6000ms
    - Weather
      - Returns a json object
      - The query should execute within 5000ms

- Rabbit MQ
  - All messages relayed on the channel must persist
- User Authentication
  - Any user can register
  - Registered users should be able to log-in
  - Unregistered user should be prohibited from logging-in

6 Test Deliverables

Artefacts to be produced as part of unit testing include the following:

- Test Plan
- Test Report
- Link to Test Code

7 Detailed Test Results

7.1 Overview of Test Results

Most unit tested conducted thus far are for the back-end server, four unit tests were conducted for each the Earthquakes and Fires disasters and one for the weather. Mocha.js was the testing framework of choice used on the back-end server due to ease of testing nodejs applications, Chai was used as an assertion library due to to it delightfulness in pairing with any javascript testing framework i.e. Mocha.js. Such testing framework assisted for running automated unit tests

7.2 Functional Requirements Test Results

The tests we have created for this module are contained within the file located on Github at the : `https://github.com/Coeus2016/visualizer-server/test`. The following results we re obtained from the tests conducted. The tests to produce the following results have passed and the reasons are stated below

7.2.1 Test Case 1 (TC 4.1)

1. All the earthquakes data were returned.
2. The returned data was in the correct format (json object).

7.2.1.1 Result :

Pass

7.2.2 Test Case 2 (TC 4.2)

1. All the earthquakes data returned was between the specified time period.
2. The returned data was in the correct format (json object).

7.2.2.1 Result :

Pass

7.2.3 Test Case 3 (TC 4.3)

1. All the earthquakes data returned was less than the specified time period.
2. The returned data was in the correct format (json object).

7.2.3.1 Result :

Pass

7.2.4 Test Case 4 (TC 4.4)

1. All the earthquakes data returned was greator than the specified time period.
2. The returned data was in the correct format (json object).

7.2.4.1 Result :

Pass

7.2.5 Test Case 5 (TC 4.5)

1. All the fires data were returned.
2. The returned data was in the correct format (json object).

7.2.5.1 Result :

Pass

7.2.6 Test Case 6 (TC 4.6)

1. All the fires data returned was between the specified time period.
2. The returned data was in the correct format (json object).

7.2.6.1 Result :

Pass

7.2.7 Test Case 7 (TC 4.7)

1. All the fires data returned was less than the specified time period.
2. The returned data was in the correct format (json object).

7.2.7.1 Result :

Pass

7.2.8 Test Case 8 (TC 4.8)

1. All the fires data returned was greator than the specified time period.
2. The returned data was in the correct format (json object).

7.2.8.1 Result :

Pass

7.2.9 Test Case 9 (TC 4.9)

1. All the weather data were returned.
2. The returned data was in the correct format (json object).

7.2.9.1 Result :

Pass

8 Other

1. **NPM build**
2. **Use of contracts and mock objects**
3. **How different use cases allow this application to be deployable**

9 Conclusions and Recommendations

Prior to executing a unit test, the table entries on rethink database had to be deleted and new data be scrapped from third party API, thus the time performance of each test greatly dependant on the internet download speed. Besides the time of execution the test were very much accurate as the expected data persisting on the database was returned at each instance. Further test cases have to be written and executed on the front-end to increase the test coverage.