Testing Report

SENG3011 | prOve_it | Version 1.1

2016 - 5 - 3

Table of content

1. Overview of Test Data
2. Testing Environment
3. Comparing another team’s API module
4. Usage of our testing software

1. Overview of Test Data

This is a summary of all these tests that were run for our API. You can run these tests yourself and verify the results from the attached files.

Test ID	Test name	Test type	Result	Data used/input parameters
1	Upload normally	Upload	200, returns token	event_char.csv stock_price.csv
2	Upload with no files	Upload	404 Not Found	N/A
3	Upload incorrectly formatted files	Upload	422 unprocessable entity	random1.csv random2.csv
4	Not enough files sent on upload	Upload	404 Not Found	event_char.csv
5	Uploading files with invalid data	Upload	404 Not Found	broken_event_char.csv broken_stock_price.csv
6	Uploading empty files	Upload	422 unprocessable entity	empty1.csv empty2.csv
7	Upload files with missing fields	Upload	404 Not found	edited_event_char.csv edited_stock_price.csv
8	Send request with valid input	Cum_return	200, returns cumulative returns of all companies	upper Window:5 lower Window:-5 variable Name: Cash Rate variable Upper Limit: 1.5 variable Lower Limit: -1.5 token:diwvTsq7xBGGxGqtFJkT
9	Send request with no input	Cum_return	Invalid parameters	N/A
10	Send request with no token	Cum_return	Invalid parameters	upper Window:5 lower Window:-5 variable Name: Cash Rate variable Upper Limit: 1.5 variable Lower Limit: -1.5 token:diwvTsq7xBGGxGqtFJkT
11	Send request with invalid token	Cum_return	Invalid token	upper Window:5 lower Window:-5 variable Name: Cash Rate variable Upper Limit: 1.5 variable Lower Limit: -1.5 token:ABCDEFG
12	Send request with invalid topic name	Cum_return	Invalid topic name	upper Window:5 lower Window:-5 variable Name: DERP variable Upper Limit: 1.5

				variable Lower Limit: -1.5 token:diwvTsq7xBGGxGqtFJkT
13	Send request where upper_window < lower_window	Cum_return	Invalid parameters	upper Window:-5 lower Window: 5 variable Name: Cash Rate variable Upper Limit: 1.5 variable Lower Limit: -1.5 token:diwvTsq7xBGGxGqtFJkT
14	Send request where variable upper limit < variable lower limit	Cum_return	Invalid parameters	upper Window:5 lower Window:-5 variable Name: Cash Rate variable Upper Limit: -1.5 variable Lower Limit: 1.5 token:diwvTsq7xBGGxGqtFJkT
15	Send request where requested parameters does not match any data	Cum_return	Invalid parameters	upper Window:5 lower Window:-5 variable Name: Cash Rate variable Upper Limit: 0.1 variable Lower Limit: 0.2 token:diwvTsq7xBGGxGqtFJkT

2. Testing Environment

Our manual testing during development was conducted using two tools, **Postman** for Google Chrome, and **curl** for the bash command line. These tools were chosen because most teams (including us) this semester are using REST API's, which these two tools are designed for. Postman offers a simple-to-use interface that allows us to review the results easily, where curl is much faster to use but requires deciphering the output. However since not all people have access to Postman, we have always retained support of using curl in our documentation, giving users the option to do what they think is easier.

When we had established that our API was working, we moved on to more automated testing by using the "Frisby" javascript library. Frisby allows the user to write scripts to automatically make requests to an API server and compare the output to an expected output. Similarly to curl, it is executed from the command line.

Due to the nature of REST API's requiring the user to upload files to the server, we did not run tests on overly large inputs such as those of size 1GB or greater for our own application. Furthermore, we did not test the efficiency of our algorithm as we simply depend on the

calculations already provided by the data. However, this means that if calculations inside the given data are wrong, the output for our application will also be incorrect.

Additionally, we do not run any specific test on the uniqueness of the return token for our file-upload endpoint. However, on creation of a token, the API program will re-generate a new token if the generated one already exists in our database. So, via the software construction, we can guarantee that each token will be only correspond to one upload request. Hence, we are confident that without any further testing, our API backend will return a unique and valid token for each files upload request from our user.

For testing, we systematically tested each part of the functionality of our program, starting with the features that are prerequisites to the other features. Obviously in order to analyse data, the data must first be uploaded, so we tested uploading valid files, various types of invalid files, and no files in order to ensure the correct errors and handling occurred. We then tested the actual data returns using various combinations of both valid and invalid parameters (upper window, lower window, variable name, variable upper limit, variable lower limit, token). This ensured that our API could handle all sorts of invalid requests and corner cases.

In order to make sure that we don't break older features as we code the newer features, we ran our program through our set of tests at regular intervals and when milestones of development were reached. We continuously test the older scenarios to ensure nothing is broken along the way. This was done despite utilising a modular design where new features are developed independently of older features, as software systems can be prone to breaking even when older features are not changed.

3. Comparing another team's API module

Amongst all the teams that are doing the Event Study API, we decided to test team "Wolf of Bridge St" as they have a well documented API reference. Comparing their API to ours, uploading files to the API is performed in the same way. Both APIs require two valid csv files in form-data with a POST request in order to upload correctly. However, there is a significant difference between the parameters in the second endpoint of both APIs. The Wolf of Bridge St API allows the user to specify the event date, a list of RICs, and multiple event

types. Instead, our API can only specify a particular event type, along with the upper and lower range for that event type.

Comparatively speaking, the Wolf of Bridge Street teams' API gives a little more control to the user, e.g by allowing them to specify multiple RICs. However, this comes at a cost of requiring higher technical knowledge as well as a more complex mode of interaction. We believe that given our target user-base (Investors, Traders) a simpler interface will be more beneficial. Additionally, if a power user wants to use our API and get more specific outputs, they can easily filter the resulting JSON object on their own by piping the output to a JavaScript variable. Moreover, since we provide all the data, this allows users to manipulate the output however they wish.

4. Usage of our testing software

We use the Frisby javascript package to test our software. To run our test scripts, please install node.js and the required packages (jasmine-node, frisby)

`Install jasmine-node:` `$ sudo npm install -g jasmine-node`

`Install Frisby:`
`$ sudo npm install -g frisby`
`Or`
`$sudo npm install frisby --save-dev`

Run the test files with jasmine-node that end with `_spec.js`
For example: `$ jasmine-node abc_spec.js` will run the test file `abc_spec.js` with `jasmine-node`

Our test files are: `upload_spec.js, cum_return_spec.js and wolfofbridgestAPI_spec.js`

`upload_spec.js` is a JavaScript testing script for testing our upload files endpoint

`cum_return_spec.js` is a JavaScript testing script to test our cumulative return endpoint

`wolfofbridgestAPI_spec.js` is a script for our tests on the team Wolf Of Bridge Street, we tested both uploading files and a parameterized cumulative return API call.