Testing Report

Mock Market

12/10/2021

Dylan Porter

Table of Contents

1. Purpose

The purpose of this document is to outline the unit tests and acceptance tests that were executed to ensure proper system functionality across the requirements. The document goes into detail of the tools that were used for unit testing as well as the associated results of them. In addition, the document analysis the unit testing progress throughout the application's development and tries to also explain the reasoning behind code coverage metrics.

2. Product Scope

The mission of *Mock* Market is to enable those interested in market investing by allowing them to place theoretical trades on the actual market without using real money. The application will provide an interface to conduct trades, including buying and selling of securities, using fake (mocked) funds. The interface will also show what the theoretical gain or loss would be given the individual's portfolio. This application will help with learning about market trading without the risk of monetary loses.

3. Format of the Document

The document outlines the specifics behind the unit tests, the associated results, and an analysis of the unit tests that were ran. Additionally, the document also outlines the specifics behind the user acceptance tests, the associated results, and an analysis of the tests that were ran. Additionally, the document provides traceability to the use cases / requirements that are relevant to the unit and user acceptance tests.

4. Types of Testing

The types of tests utilized in this project were primarily from Junit. Junit tests are generally used to test react-based applications, and so I used the testing framework to verify a lot of the application's component state. Additionally, the User Acceptance tests were utilized by manual inspection and interpretation. They were derived from the general requirements outlined in the system (from the use cases).

For the front-end and back-end, Junit test cases were used. On the back-end server, I verified server API responses from the endpoints I was exposing. On the front-end, I verified application-based component rendering state. In other words, I verified that the application rendered correctly and rendered without errors.

5. History of Unit Tests

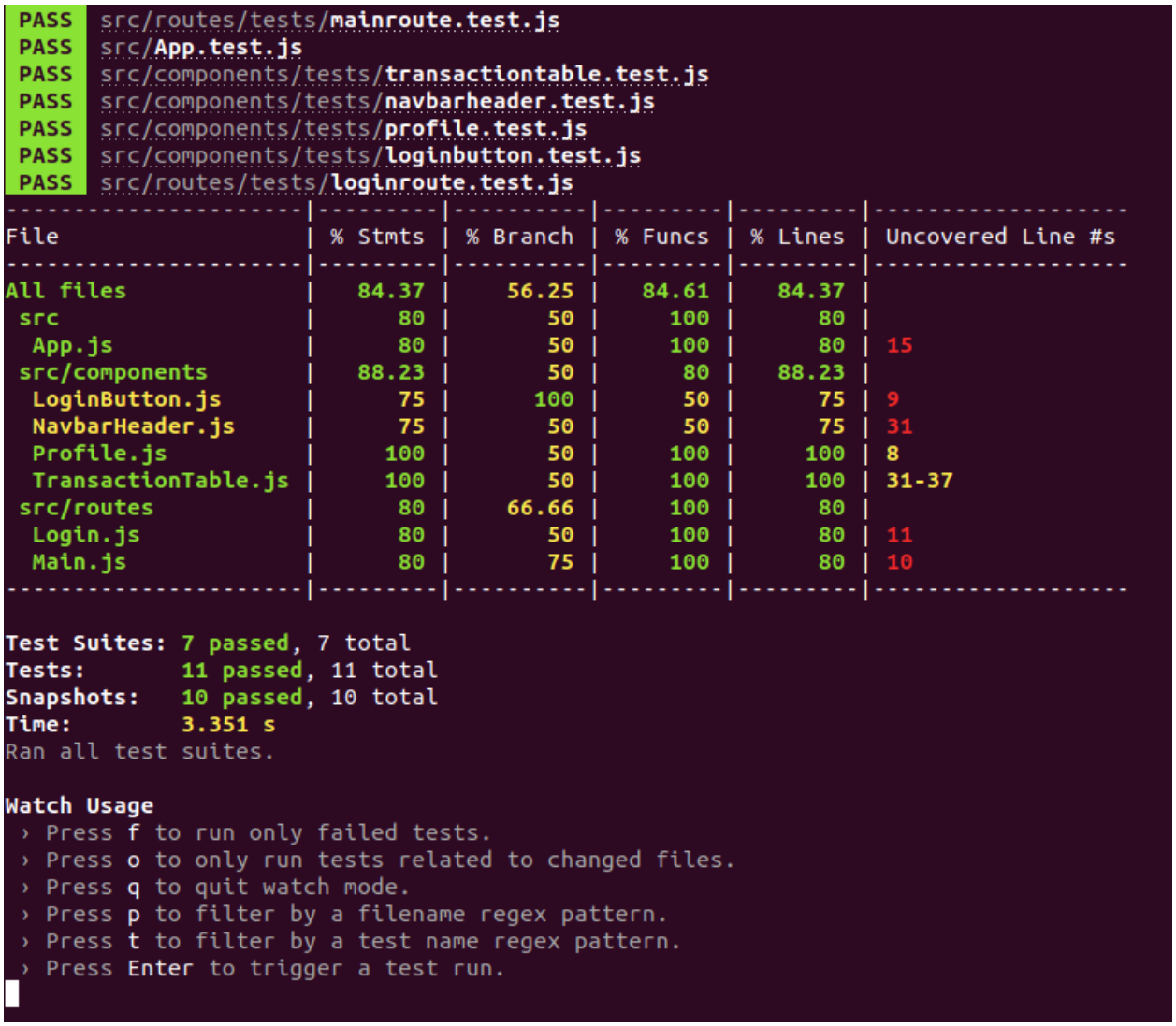
5.1. Sprint 1

At the conclusion of Sprint 1, I had written 7 test suites with a total of 11 tests (all of which passed). The tests covered 84% of statements, 56% of branches, 85% of

functions, and 85% of lines. Here are the corresponding screenshots from running the test cases:

Sprint 1 Test Results:

Sprint 1 Test Coverage



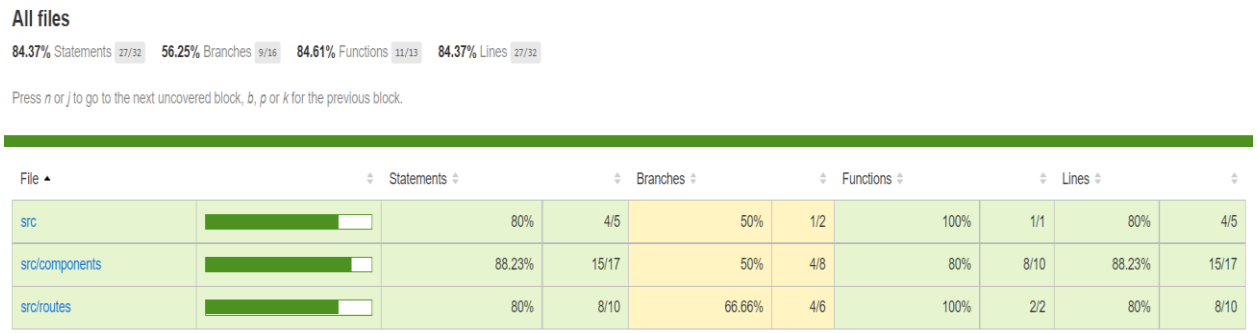
5.2. Sprint 2

At the conclusion of Sprint 3, I had written 7 test suites with a total of 11 tests (all of which passed) on the front-end. For the back-end, I wrote 1 test suite with 31 unit tests (all of which passed). The tests for the front-end covered 84% of statements, 56% of branches, 85% of functions, and 85% of lines. For the back-end, the tests covered 80% of statements, 77% of branches, 65% of functions, and 80% of lines. Here are the corresponding screenshots from the test cases:

Sprint 2 Front-End Test Results

```
PASS  src/App.test.js
PASS  src/routes/tests/mainroute.test.js
PASS  src/components/tests/transactiontable.test.js
PASS  src/components/tests/navbarheader.test.js
PASS  src/routes/tests/loginroute.test.js
PASS  src/components/tests/loginbutton.test.js
 › 1 snapshot written.
 › 1 snapshot obsolete.
   • test should match snapshot 1
PASS  src/components/tests/profile.test.js
 › 1 snapshot written.
 › 1 snapshot obsolete.
   • test should match snapshot 1

Snapshot Summary
 › 2 snapshots written from 2 test suites.
 › 2 snapshots obsolete from 2 test suites. To remove them all, press `u`.
   ↳ src/components/tests/loginbutton.test.js
       • test should match snapshot 1
   ↳ src/components/tests/profile.test.js
       • test should match snapshot 1

Test Suites: 7 passed, 7 total
Tests:       11 passed, 11 total
Snapshots:   2 obsolete, 2 written, 8 passed, 10 total
Time:        2.682 s, estimated 12 s
Ran all test suites.

Watch Usage: Press w to show more.
```

Sprint 2 Front-End Coverage



Sprint 2 Back-End Test Results

```
PASS ./index.test.js (18.751 s)
  POST Requests App API
    POST /account success
      ✓ should respond with a 200 status code (40 ms)
    POST /account failure
      ✓ should respond with a 400 status code (6 ms)
    POST /deposit success
      ✓ should respond with a 200 status code (177 ms)
    POST /deposit failure
      ✓ should respond with a 400 status code (did not include username) (3 ms)
      ✓ should respond with a 400 status code (did not include amount) (4 ms)
      ✓ should respond with a 400 status code (0 account amount) (4 ms)
      ✓ should respond with a 400 status code (negative account amount) (3 ms)
    POST /withdrawal success
      ✓ should respond with a 200 status code (163 ms)
    POST /withdrawal failure
      ✓ should respond with a 400 status code (did not include username) (17 ms)
      ✓ should respond with a 400 status code (did not include amount) (9 ms)
      ✓ should respond with a 400 status code (0 account amount) (126 ms)
      ✓ should respond with a 400 status code (negative account amount) (156 ms)
    POST /purchase success
      ✓ should respond with a 200 status code (564 ms)
    POST /purchase failure
      ✓ should respond with a 400 status code (did not include username) (12 ms)
      ✓ should respond with a 400 status code (did not include amount) (9 ms)
      ✓ should respond with a 400 status code (did not include ticker) (6 ms)
      ✓ should respond with a 400 status code (invalid ticker) (4519 ms)
      ✓ should respond with a 400 status code (negative account amount) (443 ms)
    POST /sale success
      ✓ should respond with a 200 status code (534 ms)
    POST /sale failure
      ✓ should respond with a 400 status code (selling more than owned) (350 ms)
      ✓ should respond with a 400 status code (did not include username) (8 ms)
      ✓ should respond with a 400 status code (did not include amount) (9 ms)
      ✓ should respond with a 400 status code (did not include ticker) (6 ms)
      ✓ should respond with a 400 status code (invalid ticker) (4907 ms)
      ✓ should respond with a 400 status code (negative account amount) (371 ms)
  GET Requests App API
    GET /query success
      ✓ should respond with a 200 status code (31 ms)
    GET /query failure
      ✓ should respond with a 400 status code (16 ms)
    GET /account/details success
      ✓ should respond with a 200 status code (146 ms)
    GET /account/details failure
      ✓ should respond with a 400 status code (8 ms)
  GET Requests Market
    GET /query success
      ✓ should respond with a 200 status code (180 ms)
    GET /ticker/info success
      ✓ should respond with a 200 status code and correct ticker (251 ms)

Test Suites: 1 passed, 1 total
Tests:       31 passed, 31 total
Snapshots:   0 total
```

Sprint 2 Back-End Coverage

All files

79.95% Statements 291/364 77.45% Branches 79/102 64.71% Functions 33/51 79.89% Lines 290/363

Press *n* or *j* to go to the next uncovered block, *b*, *p* or *k* for the previous block.

File		Statements		Branches		Functions		Lines	
mock-market-server		95.45%	21/22	100%	2/2	50%	1/2	95.45%	21/22
mock-market-server/db		78.34%	246/314	77.55%	76/98	62.16%	23/37	78.27%	245/313
mock-market-server/market		85.71%	24/28	50%	1/2	75%	9/12	85.71%	24/28

5.3. Sprint 3

At the conclusion of Sprint 3, I had written 10 test suites with a total of 14 unit tests for the front-end. For the back-end, no additional unit tests were added (since no functionality was added). The tests for the front-end covered 40% of statements, 40% of branches, 20% of functions, and 40% of lines. For the back-end, the tests covered 80% of statements, 77% of branches, 65% of functions, and 80% of lines. Here are the corresponding screenshots from the test cases:

Sprint 3 Front-End Test Results

```
PASS  src/routes/tests/mainroute.test.js
PASS  src/routes/tests/loginroute.test.js
PASS  src/routes/tests/inforoute.test.js
PASS  src/routes/tests/transferroute.test.js
PASS  src/routes/tests/searchroute.test.js
PASS  src/components/tests/loginbutton.test.js
PASS  src/components/tests/profile.test.js
PASS  src/App.test.js
PASS  src/components/tests/transactiontable.test.js
PASS  src/components/tests/navbarheader.test.js

Test Suites: 10 passed, 10 total
Tests:       14 passed, 14 total
Snapshots:   13 passed, 13 total
Time:        3.028 s
Ran all test suites.

Watch Usage: Press w to show more.
```

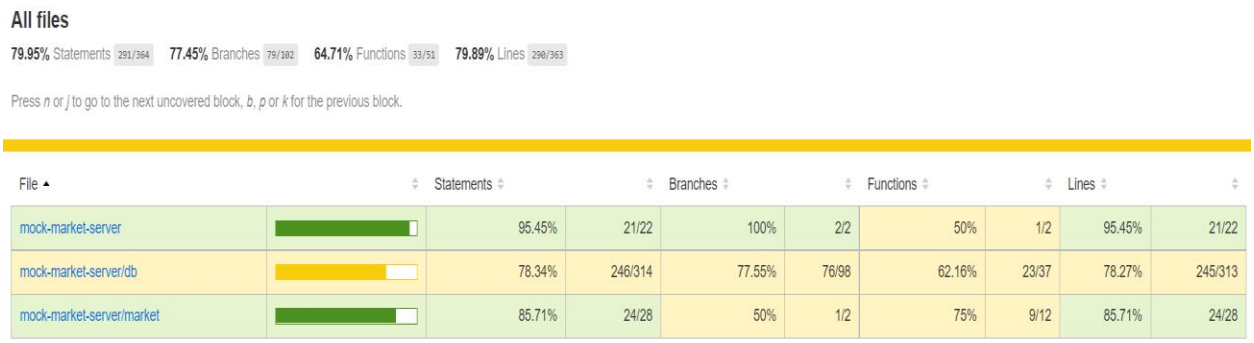
Sprint 3 Front-End Code Coverage

```
 PASS  src/routes/tests/loginroute.test.js
 PASS  src/routes/tests/mainroute.test.js
 PASS  src/components/tests/transactiontable.test.js
 PASS  src/routes/tests/searchroute.test.js
 PASS  src/routes/tests/inforoute.test.js
 PASS  src/App.test.js
 PASS  src/components/tests/navbarheader.test.js
 PASS  src/routes/tests/transferroute.test.js
 PASS  src/components/tests/loginbutton.test.js
 PASS  src/components/tests/profile.test.js
----------------------|---------|----------|---------|---------|-------------------------------------------------
File                  | % Stmts | % Branch | % Funcs | % Lines | Uncovered Line #s
----------------------|---------|----------|---------|---------|-------------------------------------------------
All files             |   39.09 |    38.46 |    19.1 |   39.81 |
 src                  |   45.45 |       50 |   16.66 |   45.45 |
  App.js              |   45.45 |       50 |   16.66 |   45.45 | 21-66
 src/components       |   88.23 |       50 |      80 |   88.23 |
  LoginButton.js      |      75 |      100 |      50 |      75 | 9
  NavbarHeader.js     |      75 |       50 |      50 |      75 | 32
  Profile.js          |     100 |       50 |     100 |     100 | 8
  TransactionTable.js |     100 |       50 |     100 |     100 | 31-37
 src/routes           |   34.37 |    35.71 |   10.95 |   34.97 |
  Info.js             |   28.16 |       25 |    7.14 |   28.35 | 15,24-72,78-113,122-157,170,179-180,215-217,301-349
  Login.js            |      80 |       50 |     100 |      80 | 11
  Main.js             |      50 |       75 |   18.18 |   53.84 | 15,22-41,48,81-83,139-145
  Search.js           |   55.55 |    66.66 |   22.22 |      56 | 14,22-48,57,104-108
  Transfer.js         |   21.31 |    21.42 |    4.16 |   21.66 | 14,21-34,40-75,84-119,128-133,150-152,167-210
----------------------|---------|----------|---------|---------|-------------------------------------------------

Test Suites: 10 passed, 10 total
Tests:       14 passed, 14 total
Snapshots:   13 passed, 13 total
Time:        5.024 s
Ran all test suites.

Watch Usage
 › Press f to run only failed tests.
 › Press o to only run tests related to changed files.
 › Press q to quit watch mode.
 › Press p to filter by a filename regex pattern.
 › Press t to filter by a test name regex pattern.
 › Press Enter to trigger a test run.
```

Sprint 3 Back-End Test Results

```
PASS ./index.test.js (18.751 s)
  POST Requests App API
    POST /account success
      ✓ should respond with a 200 status code (40 ms)
    POST /account failure
      ✓ should respond with a 400 status code (6 ms)
    POST /deposit success
      ✓ should respond with a 200 status code (177 ms)
    POST /deposit failure
      ✓ should respond with a 400 status code (did not include username) (3 ms)
      ✓ should respond with a 400 status code (did not include amount) (4 ms)
      ✓ should respond with a 400 status code (0 account amount) (4 ms)
      ✓ should respond with a 400 status code (negative account amount) (3 ms)
    POST /withdrawal success
      ✓ should respond with a 200 status code (163 ms)
    POST /withdrawal failure
      ✓ should respond with a 400 status code (did not include username) (17 ms)
      ✓ should respond with a 400 status code (did not include amount) (9 ms)
      ✓ should respond with a 400 status code (0 account amount) (126 ms)
      ✓ should respond with a 400 status code (negative account amount) (156 ms)
    POST /purchase success
      ✓ should respond with a 200 status code (564 ms)
    POST /purchase failure
      ✓ should respond with a 400 status code (did not include username) (12 ms)
      ✓ should respond with a 400 status code (did not include amount) (9 ms)
      ✓ should respond with a 400 status code (did not include ticker) (6 ms)
      ✓ should respond with a 400 status code (invalid ticker) (4519 ms)
      ✓ should respond with a 400 status code (negative account amount) (443 ms)
    POST /sale success
      ✓ should respond with a 200 status code (534 ms)
    POST /sale failure
      ✓ should respond with a 400 status code (selling more than owned) (350 ms)
      ✓ should respond with a 400 status code (did not include username) (8 ms)
      ✓ should respond with a 400 status code (did not include amount) (9 ms)
      ✓ should respond with a 400 status code (did not include ticker) (6 ms)
      ✓ should respond with a 400 status code (invalid ticker) (4907 ms)
      ✓ should respond with a 400 status code (negative account amount) (371 ms)
  GET Requests App API
    GET /query success
      ✓ should respond with a 200 status code (31 ms)
    GET /query failure
      ✓ should respond with a 400 status code (16 ms)
    GET /account/details success
      ✓ should respond with a 200 status code (146 ms)
    GET /account/details failure
      ✓ should respond with a 400 status code (8 ms)
  GET Requests Market
    GET /query success
      ✓ should respond with a 200 status code (180 ms)
    GET /ticker/info success
      ✓ should respond with a 200 status code and correct ticker (251 ms)

Test Suites: 1 passed, 1 total
Tests:       31 passed, 31 total
Snapshots:   0 total
```

Sprint 3 Back-End Coverage



All files

79.95% Statements 291/364 77.45% Branches 79/102 64.71% Functions 33/51 79.89% Lines 290/363

Press *n* or *j* to go to the next uncovered block, *b*, *p* or *k* for the previous block.

File	Statements		Branches		Functions		Lines	
mock-market-server	95.45%	21/22	100%	2/2	50%	1/2	95.45%	21/22
mock-market-server/db	78.34%	246/314	77.55%	76/98	62.16%	23/37	78.27%	245/313
mock-market-server/market	85.71%	24/28	50%	1/2	75%	9/12	85.71%	24/28

5.4. Summary of Unit Tests

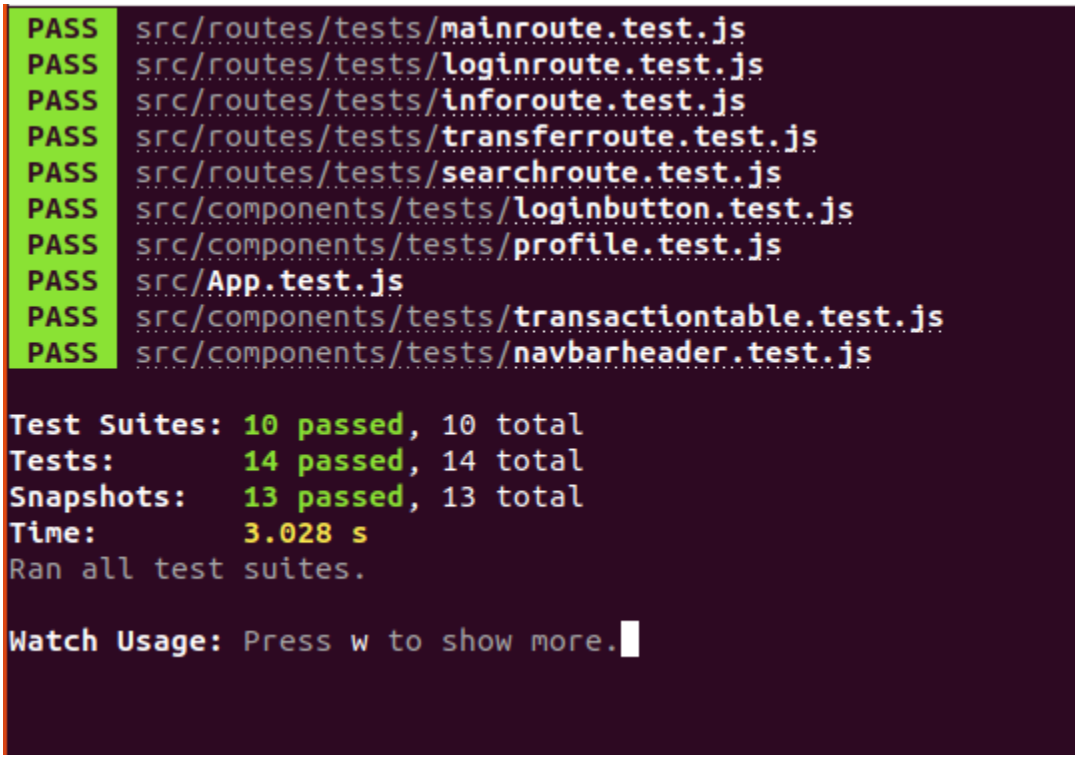
There was progress made on unit tests throughout all sprints. From the first sprint, a great deal of snapshot testing was implemented for the front-end. This gave us an inflated measure of code coverage given future improvements to the front-end. During sprint 2, there was testing progress made for the back-end. This was significant because the back-end was where a majority of the coding logic was specified. As a result, a good majority of the functionality was tested in sprint 2. For Sprint 3, I added snapshots – but I also added logic to the front-end that was unable to be tested due to unforeseen limitations with the Jest testing platform. Either way, all major components are at least tested through snapshot testing.

6. Overview of Unit Tests

Below are the results of the execution and coverage generation for the front-end and back-end unit tests. They detail the success of each unit tests and the corresponding coverage accomplished by the unit testing throughout the application.

6.1. Unit Test Execution Results

Front-End Test Results



Back-end Test Results

```
PASS ./index.test.js (13.692 s)
  POST Requests App API
    POST /account success
      ✓ should respond with a 200 status code (26 ms)
    POST /account failure
      ✓ should respond with a 400 status code (3 ms)
    POST /deposit success
      ✓ should respond with a 200 status code (170 ms)
    POST /deposit failure
      ✓ should respond with a 400 status code (did not include username) (14 ms)
      ✓ should respond with a 400 status code (did not include amount) (6 ms)
      ✓ should respond with a 400 status code (0 account amount) (4 ms)
      ✓ should respond with a 400 status code (negative account amount) (4 ms)
    POST /withdrawal success
      ✓ should respond with a 200 status code (172 ms)
    POST /withdrawal failure
      ✓ should respond with a 400 status code (did not include username) (15 ms)
      ✓ should respond with a 400 status code (did not include amount) (10 ms)
      ✓ should respond with a 400 status code (0 account amount) (147 ms)
      ✓ should respond with a 400 status code (negative account amount) (156 ms)
    POST /purchase success
      ✓ should respond with a 200 status code (376 ms)
    POST /purchase failure
      ✓ should respond with a 400 status code (did not include username) (12 ms)
      ✓ should respond with a 400 status code (did not include amount) (8 ms)
      ✓ should respond with a 400 status code (did not include ticker) (6 ms)
      ✓ should respond with a 400 status code (invalid ticker) (4462 ms)
      ✓ should respond with a 400 status code (negative account amount) (411 ms)
    POST /sale success
      ✓ should respond with a 200 status code (535 ms)
    POST /sale failure
      ✓ should respond with a 400 status code (selling more than owned) (548 ms)
      ✓ should respond with a 400 status code (did not include username) (8 ms)
      ✓ should respond with a 400 status code (did not include amount) (7 ms)
      ✓ should respond with a 400 status code (did not include ticker) (7 ms)
      ✓ should respond with a 400 status code (invalid ticker) (4525 ms)
      ✓ should respond with a 400 status code (negative account amount) (489 ms)
  GET Requests App API
    GET /query success
      ✓ should respond with a 200 status code (29 ms)
    GET /query failure
      ✓ should respond with a 400 status code (7 ms)
    GET /account/details success
      ✓ should respond with a 200 status code (281 ms)
    GET /account/details failure
      ✓ should respond with a 400 status code (6 ms)
  GET Requests Market
    GET /query success
      ✓ should respond with a 200 status code (111 ms)
    GET /ticker/info success
      ✓ should respond with a 200 status code and correct ticker (296 ms)
```

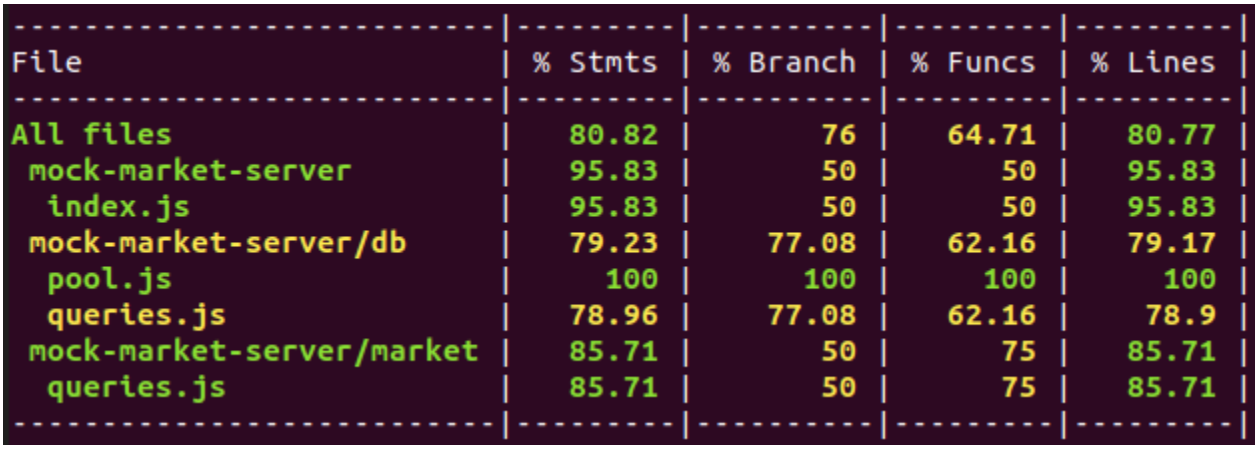
```
Test Suites: 1 passed, 1 total
Tests:       31 passed, 31 total
Snapshots:   0 total
Time:        13.753 s, estimated 16 s
Ran all test suites.
Jest did not exit one second after the test run has completed.
```

6.2. Unit Test Coverage Report

Front-end Coverage

```
 PASS  src/routes/tests/loginroute.test.js
 PASS  src/routes/tests/mainroute.test.js
 PASS  src/components/tests/transactiontable.test.js
 PASS  src/routes/tests/searchroute.test.js
 PASS  src/routes/tests/inforoute.test.js
 PASS  src/App.test.js
 PASS  src/components/tests/navbarheader.test.js
 PASS  src/routes/tests/transferroute.test.js
 PASS  src/components/tests/loginbutton.test.js
 PASS  src/components/tests/profile.test.js
----------------------|---------|----------|---------|---------|-------------------------------------------------
File                  | % Stmts | % Branch | % Funcs | % Lines | Uncovered Line #s
----------------------|---------|----------|---------|---------|-------------------------------------------------
All files             |   39.09 |    38.46 |    19.1 |   39.81 |
 src                  |   45.45 |       50 |   16.66 |   45.45 |
  App.js              |   45.45 |       50 |   16.66 |   45.45 | 21-66
 src/components       |   88.23 |       50 |      80 |   88.23 |
  LoginButton.js      |      75 |      100 |      50 |      75 | 9
  NavbarHeader.js     |      75 |       50 |      50 |      75 | 32
  Profile.js          |     100 |       50 |     100 |     100 | 8
  TransactionTable.js |     100 |       50 |     100 |     100 | 31-37
 src/routes           |   34.37 |    35.71 |   10.95 |   34.97 |
  Info.js             |   28.16 |       25 |    7.14 |   28.35 | 15,24-72,78-113,122-157,170,179-180,215-217,301-349
  Login.js            |      80 |       50 |     100 |      80 | 11
  Main.js             |      50 |       75 |   18.18 |   53.84 | 15,22-41,48,81-83,139-145
  Search.js           |   55.55 |    66.66 |   22.22 |      56 | 14,22-48,57,104-108
  Transfer.js         |   21.31 |    21.42 |    4.16 |   21.66 | 14,21-34,40-75,84-119,128-133,150-152,167-210
----------------------|---------|----------|---------|---------|-------------------------------------------------

Test Suites: 10 passed, 10 total
Tests:       14 passed, 14 total
Snapshots:   13 passed, 13 total
Time:        5.024 s
Ran all test suites.

Watch Usage
 › Press f to run only failed tests.
 › Press o to only run tests related to changed files.
 › Press q to quit watch mode.
 › Press p to filter by a filename regex pattern.
 › Press t to filter by a test name regex pattern.
 › Press Enter to trigger a test run.
```

Back-end Coverage



File	% Stmts	% Branch	% Funcs	% Lines
All files	80.82	76	64.71	80.77
mock-market-server	95.83	50	50	95.83
index.js	95.83	50	50	95.83
mock-market-server/db	79.23	77.08	62.16	79.17
pool.js	100	100	100	100
queries.js	78.96	77.08	62.16	78.9
mock-market-server/market	85.71	50	75	85.71
queries.js	85.71	50	75	85.71

6.3. Summary

The code coverage for the back-end tests a good amount of the code and logic that is presented in the application. Because of this, I have good confidence that a lot of the functionality presented within the app is tested adequately. For the front-end, there is code coverage in the form of snapshot testing. Of course, we do not have as much code coverage on the front-end, but I do think that the snapshot testing presents an adequate means to testing the front-end side of the application.

6.4. Junit Unit Tests

6.4.1. Description of Columns

Column Name	Description
Test Class Name **Requirement ID's**	The name of the class or component being tested in the Mock Market project. Below each class are the relevant requirements being tested.
Test Methods	This is the name of the test method being ran. In Junit, it is the 'describes' for each test case.
Status	The indication of pass or fail on last test run.
Date	The last date that the test case was run.

6.4.2. Description of Columns

Test Class Name Requirement ID's	Test Methods	Status	Date
App	Renders with crashing	Passed	12/10/2021

Routes/tests/loginroute Requirement ID's: 3.3.1, 3.3.2, 3.3.4, 3.4.1	Should match snapshot	Passed	12/10/2021
Routes/tests/mainroute Requirement ID's: 3.3.4, 3.3.5, 3.3.6, 3.3.7	Should match snapshot	Passed	12/10/2021
Routes/tests/searchroute Requirement ID's: 3.3.4, 3.3.5	Should match snapshot	Passed	12/10/2021
Routes/tests/inforoute Requirement ID's: 3.3.4, 3.3.5	Should match snapshot	Passed	12/10/2021
Routes/tests/transferroute Requirement ID's: 3.3.7,3.3.8	Should match snapshot	Passed	12/10/2021
Components/tests/loginbutton Requirement ID's: 3.3.1, 3.3.2, 3.3.4, 3.4.1	Should match snapshot	Passed	12/10/2021
Components/tests/navbarheader Requirement ID's: 3.3.2	Should match snapshot	Passed	12/10/2021
Components/tests/profile Requirement ID's: 3.3.2, 3.4.1	Should match snapshot	Passed	12/10/2021
Components/tests/transactiontable	Should match snapshot with empty headers and rows	Passed	12/10/2021
Requirement ID's: 3.3.4, 3.3.5	Should match snapshot with empty headers and rows	Passed	12/10/2021
	Should match snapshot with empty headers and populated rows	Passed	12/10/2021
	Should match snapshot with populated headers and empty rows	Passed	12/10/2021
	Should match snapshot with populated headers and partial rows	Passed	12/10/2021
	Should match snapshot with partial headers and populated rows	Passed	12/10/2021

6.4.3. Back-End Unit Tests

Test Class Name **Requirement ID's**	**Test Methods**	**Status**	**Date**

Index Requirement ID's: 3.3.1, 3.3.2, 3.3.3, 3.3.4, 3.3.5, 3.3.6, 3.3.7, 3.4.1, 3.4.2	POST /account success should respond with a 200 status code	Passed	12/10/2021
	POST /account failure should respond with a 400 status code	Passed	12/10/2021
	POST /deposit success should respond with a 200 status code	Passed	12/10/2021
	POST /deposit failure should respond with a 400 status code (did not include username)	Passed	12/10/2021
	POST /deposit failure should respond with a 400 status code (did not include amount)	Passed	12/10/2021
	POST /deposit failure should respond with a 400 status code (0 account amount)	Passed	12/10/2021
	POST /deposit failure should respond with a 400 status code (negative account amount)	Passed	12/10/2021
	POST /withdrawal success should respond with a 200 status code	Passed	12/10/2021
	POST /withdrawal failure should respond with a 400 status code (did not include username)	Passed	12/10/2021
	POST /withdrawal failure should respond with a 400 status code (did not include amount)	Passed	12/10/2021
	POST /withdrawal failure should respond	Passed	12/10/2021

	with a 400 status code (0 account amount)		
	POST /withdrawal failure should respond with a 400 status code (negative account amount)	Passed	12/10/2021
	POST /purchase success should respond with a 200 status code	Passed	12/10/2021
	POST /purchase failure should respond with a 400 status code (did not include username)	Passed	12/10/2021
	POST /purchase failure should respond with a 400 status code (did not include amount)	Passed	12/10/2021
	POST /purchase failure should respond with a 400 status code (did not include ticker)	Passed	12/10/2021
	POST /purchase failure should respond with a 400 status code (invalid ticker)	Passed	12/10/2021
	POST /purchase failure should respond with a 400 status code (negative account amount)	Passed	12/10/2021
	POST /sale success should respond with a 200 status code	Passed	12/10/2021
	POST /sale failure should respond with a 400 status code (selling more than owned)	Passed	12/10/2021
	POST /sale failure should respond with a 400 status code (did not include username)	Passed	12/10/2021

	POST /sale failure should respond with a 400 status code (did not include amount)	Passed	12/10/2021
	POST /sale failure should respond with a 400 status code (did not include ticker)	Passed	12/10/2021
	POST /sale failure should respond with a 400 status code (invalid ticker)	Passed	12/10/2021
	POST /sale failure should respond with a 400 status code (negative account amount)	Passed	12/10/2021
	GET /query success should respond with a 200 status code	Passed	12/10/2021
	GET /query failure should respond with a 400 status code	Passed	12/10/2021
	GET /account/details success should respond with a 200 status code	Passed	12/10/2021
	GET /account/details success should respond with a 200 status code	Passed	12/10/2021
	GET /account/details failure should respond with a 400 status code	Passed	12/10/2021
	GET requests Market GET /query success should respond with a 200 status code	Passed	12/10/2021
	GET requests Market GET /ticker/info success should respond with a 200 status code and correct ticker	Passed	12/10/2021

7. Functional Tests

7.1. Description of Columns

Column Name	Description
Test Case ID	The id of the functional test case being defined.
Description	A description for the purpose of the functional test case.
Execution Steps	The steps required to execute in order to test the functional test case.
Related Requirements	Any requirements related to the test case.
Test Category	Can be either a functional or an acceptance test.
Date	The date the test was last executed.
Status	Can be pass or fail

7.2. Functional Tests

1. Test Case ID	MM-1.1
Name	User registers for new account
Description	Check to see if a user can register for a new account.
Execution Steps	1. Click 'Sign in' for the application. 2. Once redirected to the auth0.com website, click 'Sign up' or 'Continue with Google' 3. Once registered, the site redirects you back to the Mock Market application page.
Related Requirements	3.3.1
Test Category	Functional
Date	12/10/2021
Status	Pass

Test Case ID	MM-5.1
Name	User logins to new account
Description	Check to see if a user can login from an existing account.
Execution Steps	1. Click 'Sign in' for the application. 2. Once redirected to the auth0.com website, enter the 'Email address' and 'Password' associated with the account. You can also 'Continue with Google' 3. Once logged in, the site redirects you back to the Mock Market application page.
Related Requirements	3.3.2

Test Category	Functional
Date	12/10/2021
Status	Pass

Test Case ID	MM-9.1
Name	User logs in securely to the application.
Description	Check to see if the application webpage is secure and certified with a certificate.
Execution Steps	1. Click 'Sign in' for the application. 2. Login or register for an account and get redirected back to the main application page. 3. Click on the upper-left hand lock of the browser and verify that it says "Connection is secure" 4. View the certificate information and verify that it is issued to *.herokuapp.com **by** Amazon.
Related Requirements	3.3.2, 3.4.1
Test Category	Non-Functional
Date	12/10/2021
Status	Pass

Test Case ID	MM-11.1
Name	Implement Postgresql database
Description	Very that the postgresql plugin is installed.
Execution Steps	1. Log into Heroku admin console. 2. Click the 'fierce mountain' application. 3. Verify that the installed add-ons has 'Heroku Postgres' 4. Go into settings → config vars and verify that there is a DATABASE_URL variable.
Related Requirements	3.3.1, 3.3.2, 3.3.3, 3.3.4, 3.3.5, 3.3.6, 3.3.7, 3.4.1
Test Category	Non-Functional
Date	12/10/2021
Status	Pass

Test Case ID	MM-27.1
Name	Implement backend for postgres query processing.
Description	Very that the backend is working and integrated with postgres.
Execution Steps	1. Go the application server site on the designated port (generally 3000). 2. Verify that the webpage loads and doesn't return an HTTP error such as 500.

	3. Verify you get the response “{"info":"Node.js, Express, and Postgres API"}” within the return body for the http request.
Related Requirements	3.3.1, 3.3.2, 3.3.3, 3.3.4, 3.3.5, 3.3.6, 3.3.7, 3.4.1
Test Category	Non-Functional
Date	12/10/2021
Status	Pass

Test Case ID	MM-11.2
Name	Implement Postgresql database
Description	Check to see if the base database schema is outlined and imported into the database.
Execution Steps	1. Verify that the data base schema in the database (after the initial db import script is ran) is correct and executed without any errors on the backend. 2. Verify that there is a sale, purchase, account, withdrawal, and deposit database. 3. Verify that the sale, purchase, withdrawal, and deposit tables have a foreign key ‘account_username’ that references the Account table’s username field.
Related Requirements	3.3.1, 3.3.2, 3.3.3, 3.3.4, 3.3.5, 3.3.6, 3.3.7, 3.4.1
Test Category	Functional
Date	12/10/2021
Status	Pass

Test Case ID	MM-21.1
Name	Verify application migration to cloud provider.
Description	Verify that the Mock Market application can be accessed on a public URL hosted on Heroku.
Execution Steps	1. Go to https://fierce-mountain-28314.herokuapp.com/ 2. Verify the Mock Market Sign in page is presented. 3. Go to https://fathomless-fjord-29866.herokuapp.com/ 4. Verify that {“info”:”Node.js, Express, and Postgres API”} response body is returned. 5. Verify that both sites are secure as outlined in MM-9.1
Related Requirements	3.3.1, 3.3.2, 3.3.3, 3.3.4, 3.3.5, 3.3.6, 3.3.7, 3.4.1
Test Category	Non-Functional
Date	12/10/2021
Status	Pass

Test Case ID	MM-27.1
Name	Implement Backend for postgres query processing.
Description	Verify that the backend can interact with the SQL database.
Execution Steps	1. Verify that the backend is active by going to https://fathomless-fjord-29866.herokuapp.com/ and receiving a non-error HTTP response. This means that the application is properly connected to the SQL database.
Related Requirements	3.3.1, 3.3.2, 3.3.3, 3.3.4, 3.3.5, 3.3.6, 3.3.7, 3.4.1
Test Category	Functional
Date	12/10/2021
Status	Pass

Test Case ID	MM-29.1
Name	Implement / Design application page UI
Description	Verify that the log in page UI is correct.
Execution Steps	1. Once at the login page, verify that there is a there is a div panel centered on the webpage. 2. Verify that there is a Sign In button. 3. Verify that there is a header displayed with the term 'Mock Market' and the associated logo.
Related Requirements	3.3.1, 3.3.2
Test Category	Functional
Test Case ID	MM-29.2
Date	12/10/2021
Status	Pass
Name	Implement / Design application page UI
Description	Verify that once logged in, the main page displays all relevant information.
Execution Steps	1. Click 'Sign in' for the application. 2. Once redirected to the auth0.com website, click 'Sign up' or 'Continue with Google' 3. Once registered, the site redirects you back to the Mock Market application page. 4. Verify that the main page displays personal account information such as Profit / Loss, portfolio history (such as transaction history). 5. Verify that the net account value (Stock Value, Funds Remaining, Total Profit, Total Deposit, and % Change from Deposit) are displayed.

Related Requirements	3.3.1, 3.3.2
Test Category	Functional
Date	12/10/2021
Status	Pass

Test Case ID	MM-29.3
Name	Implement / Design application page UI
Description	Verify that the search page is correct.
Execution Steps	1. Log into the application. 2. Press the “search” button in the header. 3. Search a stock and verify that it returns the results of the stocks that match the string entered in a loose fashion (fuzzy matching). 4. Verify that clicking on one of the stocks takes you to the stock information page.
Related Requirements	3.3.1, 3.3.2, 3.3.4, 3.3.5
Test Category	Functional
Date	12/10/2021
Status	Pass

Test Case ID	MM-29.4
Name	Implement / Design application page UI
Description	Verify that the stock information page is correct.
Execution Steps	1. Search for a valid stock using the search panel in the application panel and select it. 2. Verify that the corresponding page displays data about the stock such as open price, market capitalization, previous close, etc. 3. Verify that the page displays an option to buy or sell the stock (if it is owned already).
Related Requirements	3.3.4, 3.3.5
Test Category	Functional
Date	12/10/2021
Status	Pass

Test Case ID	MM-34.1
Name	Front-end integration with backend
Description	Verify that purchasing a stock is persisted on the front-end.
Execution Steps	1. Click ‘Sign in’ for the application. 2. Purchase a stock using the UI. 3. Verify that the stock purchases was persisted and given affordance on the UI by going to the

	dashboard / main page and viewing transaction history.
Related Requirements	3.3.4, 3.3.5
Test Category	Functional
Date	12/10/2021
Status	Pass

Test Case ID	MM-34.2
Name	Front-end integration with backend
Description	Verify that selling a stock is persisted on the front-end.
Execution Steps	1. Click 'Sign in' for the application. 2. Sell a stock using the UI. 3. Verify that the stock sale was persisted and given affordance on the UI by going to the dashboard / main page and viewing transaction history.
Related Requirements	3.3.4, 3.3.5
Test Category	Functional
Date	12/10/2021
Status	Pass

Test Case ID	MM-34.3
Name	Front-end integration with backend
Description	Verify that withdrawing funds is persisted on the front-end.
Execution Steps	1. Click 'Sign in' for the application. 2. Withdrawal funds from the application. 3. Verify that the total account value decreases by the amount that you withdrew.
Related Requirements	3.3.6, 3.3.7
Test Category	Functional
Date	12/10/2021
Status	Pass

Test Case ID	MM-34.3
Name	Front-end integration with backend
Description	Verify that depositing funds is persisted on the front-end.
Execution Steps	1. Click 'Sign in' for the application. 2. Deposit funds into the application. 3. Verify that the total account value increases by the amount that you deposited.
Related Requirements	3.3.6, 3.3.7
Test Category	Functional

Date	12/10/2021
Status	Pass

Test Case ID	MM-35.1
Name	Backend functional implementation.
Description	Check to see if a stock purchase is persisted on the backend.
Execution Steps	1. Click ‘Sign in’ for the application. 2. Purchase a stock using the application UI. 3. Log out and sign in again. 4. Verify that the purchase is noted in the transaction history.
Related Requirements	3.3.4, 3.3.5
Test Category	Functional
Date	12/10/2021
Status	Pass

Test Case ID	MM-35.2
Name	Backend functional implementation.
Description	Check to see if a stock sale is persisted on the backend.
Execution Steps	1. Click ‘Sign in’ for the application. 2. Sell a stock using the application UI. 3. Log out and sign in again. 4. Verify that the sale is noted in the transaction history.
Related Requirements	3.3.4, 3.3.5
Test Category	Functional
Date	12/10/2021
Status	Pass

Test Case ID	MM-35.3
Name	Backend functional implementation.
Description	Check to see if a withdrawal is persisted on the backend.
Execution Steps	1. Click ‘Sign in’ for the application. 2. Withdrawal from the account using the application UI. 3. Log out and sign in again. 4. Verify that the withdrawal is noted in the transaction history and that the total account value is updated appropriately.
Related Requirements	3.3.6, 3.3.7
Test Category	Functional
Date	12/10/2021

Status	Pass

Test Case ID	MM-35.4
Name	Backend functional implementation.
Description	Check to see if a deposit is persisted on the backend.
Execution Steps	1. Click ‘Sign in’ for the application. 2. Deposit into the account using the application UI. 3. Log out and sign in again. 4. Verify that the deposit is noted in the transaction history and that the total account value is updated appropriately.
Related Requirements	3.3.6, 3.3.7
Test Category	Functional
Date	12/10/2021
Status	Pass

Test Case ID	MM-35.5
Name	Backend functional implementation.
Description	Check to see if the server returns expected results for a deposit.
Execution Steps	1. Using Postman, perform a post request with the username and amount parameters to /deposit. Set deposit to a positive number 2. Execute the request and verify that the server returns a 200 response. 3. Verify that the server returns the username, amount, and updated account value in the response.
Related Requirements	3.3.6, 3.3.7
Test Category	Functional
Date	12/10/2021
Status	Pass

Test Case ID	MM-35.6
Name	Backend functional implementation.
Description	Check to see if the server returns expected results for a deposit with negative value.
Execution Steps	1. Using Postman, perform a post request with the username and amount parameters to /deposit. Set deposit to a positive zero or negative number. 2. Execute the request and verify that the server returns a 400 error response.

	3. Verify that the server indicates that the amount must be a positive value.
Related Requirements	3.3.6, 3.3.7
Test Category	Functional
Date	12/10/2021
Status	Pass

Test Case ID	MM-35.7
Name	Backend functional implementation.
Description	Check to see if the server returns expected results for a deposit with empty username.
Execution Steps	1. Using Postman, perform a post request with the username and amount parameters to /deposit. Set deposit to a positive zero and set username to an empty string. 2. Execute the request and verify that the server returns a 400 error response. 3. Verify that the server indicates that the username is not valid
Related Requirements	3.3.6, 3.3.7
Test Category	Functional
Date	12/10/2021
Status	Pass

Test Case ID	MM-35.8
Name	Backend functional implementation.
Description	Check to see if the server returns expected results for a withdrawal.
Execution Steps	1. Using Postman, perform a post request with the username and amount parameters to /withdrawal. Set withdrawal to a lesser amount than the account total. 2. Execute the request and verify that the server returns a 200 response. 3. Verify that the server returns the username, amount, and updated account value in the response.
Related Requirements	3.3.6, 3.3.7
Test Category	Functional
Date	12/10/2021
Status	Pass

Test Case ID	MM-35.9
Name	Backend functional implementation.

Description	Check to see if the server returns expected results for a withdrawal given a value that is greater than the account total.
Execution Steps	1. Using Postman, perform a post request with the username and amount parameters to /withdrawal. Set withdrawal to a higher amount than the account total. 2. Execute the request and verify that the server returns a 400 response. 3. Verify that the server returns a message indicating that you can't withdrawal more than what is in the account.
Related Requirements	3.3.6, 3.3.7
Related Use Cases	MM-35
Test Category	Functional
Date	12/10/2021
Status	Pass

Test Case ID	MM-35.10
Name	Backend functional implementation.
Description	Check to see if the server returns expected results for a withdrawal given an empty username.
Execution Steps	1. Using Postman, perform a post request with the username and amount parameters to /withdrawal. Set withdrawal to a lower amount than the account total. Set username to an empty string. 2. Execute the request and verify that the server returns a 400 response. 3. Verify that the server returns a message indicating that the user is invalid.
Related Requirements	3.3.6, 3.3.7
Test Category	Functional
Date	12/10/2021
Status	Pass

Test Case ID	MM-35.11
Name	Backend functional implementation.
Description	Check to see if the server returns expected results for a username creation given a valid username.
Execution Steps	1. Using Postman, perform a post request with the username to /account. Set username to a valid string such as user_1

	2. Execute the request and verify that the server returns a 200 response. 3. Verify that the server returns a response containing the user that was added.
Related Requirements	3.3.1, 3.3.2
Test Category	Functional
Date	12/10/2021
Status	Pass

Test Case ID	MM-35.12
Name	Backend functional implementation.
Description	Check to see if the server returns expected results for a username that already exists.
Execution Steps	1. Using Postman, perform a get request with the username to /account. Set username to a valid string such as user_1. The user should already exist. 2. Execute the request and verify that the server returns a 200 response. 3. Verify that the server returns a response containing the user that was contained in the database.
Related Requirements	3.3.1, 3.3.2
Test Category	Functional
Date	12/10/2021
Status	Pass

Test Case ID	MM-35.13
Name	Backend functional implementation.
Description	Check to see if the server returns expected results for a username that does not exist..
Execution Steps	1. Using Postman, perform a get request with the username to /account. Set username to a valid string such as user_1. The user should not already exist. 2. Execute the request and verify that the server returns a 400 response. 3. Verify that the server returns a response specifying that the user does not exist in the system.
Related Requirements	3.3.1, 3.3.2
Test Category	Functional
Date	12/10/2021
Status	Pass

Test Case ID	MM-45
Name	Log any system errors
Description	Check to see if the server logs all output when server is running.
Execution Steps	1. Log into the Heroku dashboard 2. Exec into the fathomless-fjord-29866 machine. 3. Verify that logs are being created when the application is running. They should contain a timestamp.
Related Requirements	3.4.2
Test Category	Functional

8. Acceptance Tests

8.1. Description of Columns

Column Name	Description
Requirement ID	The id of the requirement being tested.
Purpose	The purpose of this acceptance test.
Expected Result	The outcome that is expected to happen as a result of the test.
Status	Can be pass or fail.
Date	The date the test was last executed.

8.2. Acceptance Tests

Requirement ID	Purpose	Expected Result	Status	Date
3.3.1	Check to see if a user can register with a new account.	A user can register a new account and will be redirected to the main page.	Pass	12/10/2021
3.3.1	Check to see if the user can navigate through the application after initially registering	The user can access all pages of the application once registered.	Pass	12/10/2021
3.3.2	Check to see if the user can login to the application with an existing account	A user is logged in and sent to the main page.	Pass	12/10/2021
3.3.2	Check to see if the user can navigate through the application after logging in with an existing account.	The user can access all pages of the application once logged in.	Pass	12/10/2021
3.3.2	Check to see if the logged in user can view their display name in the interface.	The display name is in the application header once logged in.	Pass	12/10/2021

3.3.4	Check to see if the user can search for stocks.	The user can enter a ticker and results are returned based on available stocks.	Pass	12/10/2021
3.3.4	Check to see if the user can view stock information on the stock before purchasing.	The interface displays stock information such as market cap, average volume, etc to the user.	Pass	12/10/2021
3.3.4	Check to see if a user cannot purchase a stock if they have insufficient funds.	The user is unable to purchase a stock if their account value is below the cost of the purchase.	Pass	12/10/2021
3.3.4	Check to see if the application is updated once a purchase is made.	The application UI displays the purchase and the updated account value. The purchase is now shown in the purchase history.	Pass	12/10/2021
3.3.5	Check to see if the user can search for stocks that are already purchased.	The interface provides a list of purchased stocks to the user.	Pass	12/10/2021
3.3.5	Check to see if the user can view stock information on the stock before selling.	The interface displays stock information about the sale such as average cost and quantity purchased.	Pass	12/10/2021
3.3.5	Check to see if a user cannot purchase a stock if they are selling more than what is available.	The interface rejects the sale request if the user is selling more stock than they have.	Pass	12/10/2021
3.3.6	Check to see if the user is able to withdrawal funds from their account.	The interface has a page that allows the user to deposit or withdrawal funds.	Pass	12/10/2021
3.3.6	Check to see if a withdrawal on an amount greater than an account value is not allowed.	The system rejects the withdrawal if the user is trying to withdrawal more funds than are available.	Pass	12/10/2021
3.3.7	Check to see if the user is able to deposit funds into their account	The interface has a page that allows the user to	Pass	12/10/2021

		deposit or withdrawal funds.		
3.3.7	Check to see if a user is unable to deposit an amount greater than $1,000,000 into their account.	The system rejects the user from depositing an amount greater than $1,000,000 into their account at a time.	Pass	12/10/2021
3.4.1	Check to see that the application is secure when in use.	The user is able to log in securely via OAuth. The icon in browsers indicates a trust address.	Pass	12/10/2021