Software Requirements Specification

Mock Market

10/9/2021

Dylan Porter

Contents

1. Introduction

1.1. Purpose: Mission Statement

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 losses.

1.2. Scope

Mock Market will be used as a tool to mock stock market transactions without having to use actual funds within those transactions. This tool will provide beginning investors with the tools necessary to learn about stock market trading while also eliminating the risks associated with it. The system will support the following operations:

- Allowing investors to mock a purchase or sale of a stock on the open market.
- Allows investors to use fake funds for a Stock Transaction.
- Allows investors to visualize theoretical returns from their purchases through charts and on-screen affordances.

1.3. Definitions and Acronyms

1. Account – The account used by the user to mock the purchase and sale of stock.
2. Computer - Any compatible device that Mock Market can be used on.
3. Database – The database used to host all transactions for a given account.
4. Day Trade Policy – Refers to the rules outlined here by the SEC.
5. Deposit – When a user "adds" fake money into their account for mock trading.
6. Investor – The primary user of this application.
7. Order Description – Includes the granular details of a mock order (buy and sell) including purchase price.
8. Profit Chart – The chart that shows the "aggregate" of the profits or losses for a period of time including 1D, 1W, 1M, etc.
9. Profit Point – A point on the chart that indicates what the profit was for that point in time.
10. Settled Funds Policy – Refers to the rules outlined here by the SEC.
11. Stock Description – Refers to the details of the stock purchased such as current price, volume, market cap, etc.
12. Stock Market System – Refers to the external system (API) that is queried to simulate a stock market transaction.
13. Stock Purchase – Refers to the mocked purchase of a stock.
14. Stock Sell – Refers to the mocked sale of a stock.
15. Stock Transaction – Refers to either a stock market purchase or sale that is mocked.

16. Transaction Description – Refers to the details of a mocked transaction such as quantity and average cost.

17. Transaction History – Refers to previous and current mocked market transactions.

18. Withdrawal – Refers to removing fake funds from an account.

1.4. References

- *Investor alerts and bulletins*. SEC Emblem. (2017, August 22). Retrieved October 10, 2021, from https://www.sec.gov/oiea/investor-alerts-and-bulletins/ibsettlementcycle.

- *Margin rules for day trading - sec.gov | home*. (n.d.). Retrieved October 10, 2021, from https://www.sec.gov/files/daytrading.pdf.

1.5. Regulations & Standards

1.5.1. Provide an acknowledgement of terms and conditions specifying that the "money" used in the application is not legitimate currency.

1.5.2. Provide an acknowledgement of terms and conditions specifying that all transactions are not actual market transactions.

2. Overall Descriptions

2.1. Product Perspective

This application functions as an abstraction away from actual stock market brokerages. Instead of using real funds to make real transactions, this application is solely dedicated to the learning of real stock market trading through mocked transactions. Unlikely other applications that have similar functionality, Mock Market is specifically dedicated to simulating trades – so the entire UI and interactive affordance can be designed and created around that concept. In other words, the features of this application are not simply an add-on to another application.

Furthermore, the application's main purpose is to help soon to be market traders with the toolset for them to learn more about it. Fundamentally, all the general information and statistics about stock market securities are also available; so, a user should have a plethora of information to pull from to make informed decisions. Finally, there is no private information that is required to create an account – unlike other brokerages. Since there are no transactions with the actual stock market, tax information, and other personal information is not necessary to collect – making it easier to and more intuitive to open an account.

2.2. Product Functions

- Allowing investors to mock a purchase of a stock on the open market.
- Allowing investors to mock the sale of a stock on the open market.

- Allowing investors to deposit or withdrawal mocked funds into their account for trading.
- Allowing investors to see the returns from the mocked trades made within the system.
- Allowing investors to see a visual chart of their profit or loss from their trades.
- Allowing investors to view the past and current trade information.

2.3. Stakeholder Characteristics

Mock Market will have a few stakeholders in the system – but since the application does not require a great deal of infrastructure support currently, there are not many sources of funding needed. Mainly, the end-user of the application will be the sole stakeholder for this application.

- Who is paying for the system?
    - Not Applicable / Potentially Advertisements
- Who is going to use the system?
    - Beginning Investors
    - Those looking to test their portfolio over time without monetary risk
- Who is going to judge the fitness of the system for use?
    - End-user / investor in the application
- What agencies (government) and entities (non-government) regulate any aspect of the system?
    - None
- What laws govern the construction, deployment, and operation of the system?
    - None
- Who is involved in any aspect of the specification, design, construction, testing, maintenance, and retirement of the system?
    - Software Engineer
- Who will be negatively affected if the system is built?
    - Brokerages that facilitate market trades
- Who else cares if this system exists or doesn't exist?
    - None

2.4. Constraints

2.4.1. Data persistence while offline within application.
2.4.2. Interfacing with stock market API
2.4.3. Simulate S.E.C. day trading regulation.
2.4.4. Simulate S.E.C. settlement period.

2.5. Assumptions and Dependencies

2.5.1. Reliance on external stock market API.

3. Specific Requirements

3.1. Domain Requirements

3.1.1. The system shall restrict the user from violating the defined S.E.C. day trading regulations.

3.1.2. The system shall restrict the user from violating the defined S.E.C. settlement period.

3.2. System Requirements

3.2.1. Hardware

3.2.1.1. The system requires a computer that can render HTML webpages.

3.2.1.2. The system requires an internet connection to communicate with various remote servers.

3.2.2. Software

3.2.2.1. The system requires a browser that can render HTML webpages.

3.2.2.2. The system requires session storing capabilities within the browser.

3.2.2.3. The system requires javascript to be enabled.

3.2.2.4. Ability to register for a new account (requires a valid email).

3.3. Application Requirements

3.3.1. Registering for a new account

3.3.1.1. The system shall allow a user to create a new account with a valid email address not currently in use within the system.

3.3.1.2. The system shall redirect the user to the main page of the application when registration is complete.

3.3.1.3. The system may provide feedback to the user once account registration is complete.

3.3.1.4. The system must not allow duplicate emails to be used for registration for different accounts.

3.3.1.5. The system may provide the user with a terms of service agreement upon account registration.

3.3.1.6. The system may prompt the user on the amount of mocked funds the account should start out with before trading.

3.3.2. Logging in with an existing account

3.3.2.1. The system shall allow a user to log in with a valid account that is currently in use within the system.

3.3.2.2. The system shall redirect the user to the main page of the application once the login process is complete.

3.3.2.3. The system may provide feedback to the user once the login process is complete.

3.3.2.4. The system shall display the display name of the logged in user on the application interface.

3.3.2.5. The system shall allow the user to access all pages of the application related to their account once logged in.

3.3.3. Deleting an account

3.3.3.1. The system shall support the deletion of an account via the user interface.

3.3.3.2. The system may provide feedback to the user once the deletion process is complete.

3.3.3.3. The system shall delete all personal trading information once the account is deleted.

3.3.3.4. The system shall allow for the account to be registered again once deleted.

3.3.4. Purchasing a stock

3.3.4.1. The system shall provide an interface for searching for stocks by name.

3.3.4.1.1. The system shall return results based on user input

3.3.4.2. The system shall provide an interface for the user to purchase a stock including amount.

3.3.4.3. The system shall provide information on the stock such as market cap, average volume, current price, etc.

3.3.4.4. The system shall check to make sure there are sufficient funds to make the purchase.

3.3.4.5. The system shall provide the user with feedback on whether the stock purchase was successful.

3.3.4.6. The system shall provide the details of the stock purchase such as average cost and quantity of the stock if the purchase was successful.

3.3.4.7. The system shall append the stock purchase to the purchase history.

3.3.4.8. The system shall update the state of the application to reflect the purchase including availability of funds and profit and loss.

3.3.5. Selling a stock

3.3.5.1. The system shall provide a list of already purchased stocks.

3.3.5.2. The system shall provide an interface for the user to sell a stock including amount.

3.3.5.3. The system shall provide the user with feedback on whether the stock sale was successful.

3.3.5.4. The system shall provide the details of the stock sale such as average cost sold and quantity sold if the sale was successful.

3.3.6. Withdrawing funds from an account

3.3.6.1. The system shall provide an interface for the user to withdrawal any amount of money that is available in their account.

3.3.6.2. The system shall notify the user if their withdrawal request is invalid due to insufficient funds.

3.3.6.3. The system may provide a reset capability for the user to start over and change their account value.

3.3.6.4. The system shall update the application state with the updated account value amount.

3.3.7. Depositing funds into an account

3.3.7.1. The system shall provide an interface for the user to deposit any amount of money into their account up to $1,000,000.

3.3.7.2. The system shall notify the user if their deposit request is invalid due to trying to deposit an amount greater than $1,000,000.

3.3.7.3. The system shall update the application state with the updated account value amount after deposit.

3.3.8. Profit chart

3.3.8.1. The system may provide a chart that shows the accounts profit and loss over time including '1D, 1W, 1M, 1Y, 5Y, All' options.

3.3.8.2. The system may provide a visual analysis of the graph so that the user can see their profit or loss for a particular point in time.

3.3.9. Leadership chart

3.3.9.1. The system may provide a chart that shows the top earners across the Mock Market application.

3.3.9.2. The system may provide a top 10 chart that shows relative percentage gain off of investments and total profit made.

3.3.9.3. The system may provide your relative ranking within the application.

3.4. Non-Functional Requirements

3.4.1. Maintaining a secure application

3.4.1.1. System shall allow for a secure login process with a verified domain.

3.4.1.2. System shall be reliable and secure according to Google Chrome browser.

3.4.1.3. System shall maintain secure login through the application pages.

3.4.2. Logging system errors

3.4.2.1. System shall log when there are any server-related errors within the application.

3.4.2.2. System shall log date and time of error for further debugging.

3.4.3. Offline error handling

3.4.4. System may provide offline transaction processing capabilities that allow a user to perform regular actions without an internet connection.

3.4.5. System may queue all transactions made offline so that once online, those transactions can be processed on a first-come-first-serve basis.

4. Accessibility

4.1. Access Restrictions

4.1.1. None

4.2. Disabilities Accommodations

4.2.1. None

5. Security

5.1. Authentication

5.1.1. The system shall provide an authentication mechanism to establish a session for the user with a given account.

5.1.2. The session established for the user shall be secure and verified by Google Chrome with a trusted DNS.

5.1.3. The system shall maintain a secure session for the duration of the application's use.

6. Interfaces

6.1. User Interface

6.1.1. The system shall support click input and typing for interaction via the web page interface.

6.1.2. The system shall register all events through an internet browser such as Google Chrome.

6.2. Server Interface

6.2.1. The server interface shall support RESTful web services and return JSON responses based on the query from the friend end application.

6.3. Stock Market Interface

6.3.1. The application shall query an appropriate interface to retrieve stock market information for time of purchase or sale.

7. Test Plan

7.1. Test Cases

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	N/A
Related Use Cases	MM-1 (User Registers for new Account)
Test Category	Functional

Test Case ID	MM-5.1
Name	User registers for 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	N/A
Related Use Cases	MM-5 (User logs in with existing credentials)
Test Category	Functional

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	N/A
Related Use Cases	MM-9 (User logs insecurely to the application)
Test Category	Non-Functional

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	N/A
Related Use Cases	MM-11 (Implement Postgresql database)
Test Category	Non-Functional

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	N/A
Related Use Cases	MM-11 (Implement Postgresql database) MM-27 (Implement backend for postgres query processing)
Test Category	Non-Functional

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	N/A
Related Use Cases	MM-11
Test Category	Functional

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	N/A
Related Use Cases	MM-21 (Migrate Application to Cloud Provider)
Test Category	Non-Functional

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	N/A
Related Use Cases	MM-27
Test Category	Functional

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	N/A
Related Use Cases	MM-29

Test Category	Functional
Test Case ID	MM-29.2
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	N/A
Related Use Cases	MM-29
Test Category	Functional

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	N/A
Related Use Cases	MM-29
Test Category	Functional

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	N/A
Related Use Cases	MM-29
Test Category	Functional

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	N/A
Related Use Cases	MM-34
Test Category	Functional

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	N/A
Related Use Cases	MM-34
Test Category	Functional

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	N/A
Related Use Cases	MM-34
Test Category	Functional

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	N/A
Related Use Cases	MM-34
Test Category	Functional

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	N/A
Related Use Cases	MM-35
Test Category	Functional

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	N/A
Related Use Cases	MM-35
Test Category	Functional

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	N/A
Related Use Cases	MM-35
Test Category	Functional

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	N/A
Related Use Cases	MM-35
Test Category	Functional

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	N/A
Related Use Cases	MM-35
Test Category	Functional

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	N/A
Related Use Cases	MM-35
Test Category	Functional

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	N/A
Related Use Cases	MM-35
Test Category	Functional

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	N/A
Related Use Cases	MM-35
Test Category	Functional

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	N/A
Related Use Cases	MM-35
Test Category	Functional

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	N/A
Related Use Cases	MM-35
Test Category	Functional

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	N/A
Related Use Cases	MM-35
Test Category	Functional

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	N/A
Related Use Cases	MM-35
Test Category	Functional

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	N/A
Related Use Cases	MM-35
Test Category	Functional

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	N/A
Related Use Cases	MM-45
Test Category	Functional