System & Unit Test Report

Product Name: BananaStocks Platform
Team Name: Stocks Team
Date: June 1st, 2021

System Test Scenarios: (25 points per sprint – total 75 points): You have identified user stories (that map to user requirements/functionality for your system) completed for each sprint (in that sprint's report). You will be using scenario-based testing. Scenario based testing focuses on how the user uses the system and allows for multiple user roles as well as multiple functions provided for each user role. For each sprint, list the user story or stories and the scenario or scenarios that show 'coverage' of those user stories. A scenario is a list of system level actions (including precise input and output) a user would follow to determine that each user story has been completed.

Example:
A. User story 1 from sprint 1: As a user I want to create an account so that I can use the toilet location system.
B. User story 2 from sprint 1: As a registered user I want to view a map showing locations of all toilets in 5 mile radius of current location so that I can visually choose more information about toilets of interest.

Scenario:
1. start Toilet app; select 'new user'; type • name = • password = • password confirmation = • press Enter Key • user should see verification message that account is now active
2. select 'view toilets in 5 mile radius';
3. User should see map of 5 miles radius of current location with all toilets within the toilet database marked**.**

Scenario #1 (Sprint 1):
As a customer, I should be able to type in the correct username and password to access my account through the login interface.

1. Start BananaStocks; go to login; type email and password.
2. Click log in or enter
3. Arrive at home page and see Welcome back, {username}!

Scenario #1 (Sprint 2):
As a customer, I should be able to interact with the landing page and see relevant stock data.

1. Start Bananastocks; go to login; type email and password.
2. Click log in or enter
3. Arrive at the home page and see Welcome back, {username}!
4. Go to the navbar and click the stock search tab.
5. Search up a stock and press enter or click on the stock result.
6. See the stock name you searched along with a price.

Scenario #1 (Sprint 3):
As a customer, I should be able to see stock data on a chart from our models.

1. Start Bananastocks; go to login; type email and password.
2. Click log in or enter
3. Arrive at the home page and see Welcome back, {username}!
4. Go to the navbar and click the project tab.
5. Pick a project that you have created (includes stocks, project title, and description)
6. User can see stock names and the charts associated with the stock from our NLP models

Scenario #1 (Sprint 4):
As a user, I should be able to easily view my projects and change time frames.

1. Start Bananastocks; go to login; type email and password.
2. Click log in or enter
3. Arrive at the home page and see Welcome back, {username}!
4. Go to the navbar and click the project tab.
5. Pick a project that you have created (includes stocks, project title, and description)
6. User can see stock names, charts, change time frames, download svg of chart, and zoom in and out

Unit Tests: Include a file/directory named ‘Testing’ in your Git Repository. There should be details (can be in a separate file in the directory) provided by each team member about the module and the functional testing they have done. Each team member picks a module or module and lists the equivalence classes and the test cases selected to cover all equivalence classes.

NLP Backend Testing Approach

Since the users never directly interface with the NLP backend, the approach we took to testing the backend was a whitebox testing approach. We wanted to ensure that all parts of the backend were fully working and could handle any possible bad inputs, either malicious or not.

The fundamental idea is that there are good inputs and bad inputs. Good inputs have expected results, and bad inputs should be handled appropriately.

Sprint 1: News Retrieval

In the first sprint, we completed the news retrieval from the FMP API. We decided to create it so that the backend retrieves news based on a stock ticker. Therefore the things we need to keep in mind during testing are:

1. What does our stock ticker input look like?
2. How does it handle “bad input”?
3. Does the module do its job correctly?

Sprints 2 & 3: Data processing pipeline

In sprints 2 & 3, we focused on creating the pipeline that processes all the data retrieved from the module created in Sprint 1. We took a multiprocessing approach, because we wanted the pipeline to be able to scale effectively with large amounts of incoming data. Because of this, we needed an additional stress test.

In addition, the pipeline is broken down into several stages to improve maintainability. This also helped to test each individual stage in the pipeline as if it was its own module. Therefore the things that keep in mind are:

1. Does the pipeline function correctly?
2. Does the pipeline work well under large loads?
3. Does the pipeline crash on bad input? (It should not)
4. Does each stage do its job?
5. Can each stage handle bad inputs?
6. What do the inputs/edge cases look like for each stage?

Sprint 4: Database and REST API

For the final sprint, we focused on cleaning up our database query functions so that the REST API could be correctly implemented and give users whatever they query for. For this sprint, the testing approach can be more scenario based since the customer has near direct access to the queries (through front end).

Test scenarios:

1. Can I look at news articles about a certain ticker?
2. Can I look at news articles by a certain source?
3. Can I look at sentiment about a stock?
4. Can I do all of the above within a restricted time frame?

Financial and UI Testing Approach

With the UI heavily dependent on backend sourced data, it was important for our team to ensure that elements on the UI display property when given dynamic data for rendering. Using ReactJS, a single-page-application framework, we were able to test end to end scenarios. Initially through a manual process, we later moved to an automated approach using tools such as Selenium and MochaJS.

Sprint 1: Basic Display
Our initial goal for the front and back end was to be able to see a loaded page and to make manual queries to fetch for data.

1. Can I load a page using my browser?
2. Does the reverse proxy allow me to not use port numbers?
3. Can I use Postman to make manual requests to the backend?
4. Can I fetch information from the backend and print it to the browser's console?

Sprint 2: User Related Functions
This sprint allowed us to focus on allowing users to sign up and log in to the platform. We wanted to be sure that sessions could be stored and that forms were updating properly.

1. Can I navigate to different pages (ie. sign up page)?
2. Can I fill out a form and have a local state be updated?
3. Can I submit the form to the backend using a dynamic API handler to prevent hard coded URL's?
4. Can I update the state of the application once a request has gone through?

Sprint 3: Restricted Access to Financial Projects
At this point in the project, we began to restrict pages to users not currently logged in. Projects were now able to be created, and we wanted to be sure that users could have their projects load and saved properly.

1. Can I only access the main dashboard after logging in?
2. Can I create a project and have both the front and back end update their state?
3. Can I search for stocks to add to my project and dynamically display results?
4. Can I prevent users from seeing each others' projects?

Sprint 4: Charting and OAuth
The culmination of our project is the financial metrics dashboards, which allow users to analyze their stock choices. Furthermore, we now allow users to log in through third parties. Finally, we focused on automated testing of the UI for end to end workflows.

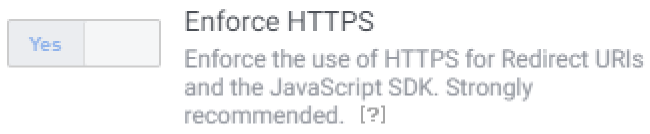
1. Can I see price metrics for stocks on a dynamically rendered chart?
2. Can I run end to end tests for UI workflows?
3. Can I log in with Google and still maintain a proper experience on the web service?

Working Prototype Known Problems Report

Product Name: BananaStocks Platform
Team Name: Stocks Team
Date: June 1st, 2021

List of functions not working correctly

Function 1: Facebook OAuth
Input/action that causes failure: Unable to set HTTPS in our domains for Facebook Live Mode, also Development Mode doesn't work for http://localhost.
Location of fault (if known):


Yes

Enforce HTTPS

Enforce the use of HTTPS for Redirect URIs and the JavaScript SDK. Strongly recommended. [?]

Facebook requires HTTPS to be enabled and I am unable to disable this option. As of now, using the source account registered to the application would allow us to log in, but fail and redirect without a secure connection.
Possible action for removal of fault: Set up a deployment with HTTPS enabled

Function 2: OAuth login does not set session
Input/action that causes failure: No user session is created when logging in with OAuth providers, preventing access to profile on the UI,
Location of fault (if known):
Login.
Possible action for removal of fault: Use a backend session manager to handle logins and have the UI use said session through a cookie.

Function 3: Login Attempt
Input/action that causes failure: Unknown. When testing in localhost (in Chrome), the login can repeatedly refresh. The issue occurred a couple of times for some team members and resolved itself later.
Location of fault (if known): Landing page
Possible action for removal of fault: Need to investigate more

Function 4: Stock Search
Input/action that causes failure: Unable to search for new stock prices on "Search" tab without refreshing page.
Location of fault (if known):
Stock search dashboard panel
Possible action for removal of fault: Reset state of panel when searching for new stock and reset price query interval.