Financial Sentiment Based on News: Test Plan

Test Scope

The test plan for our project will cover unit testing, integration testing, regression testing, and validation testing. Unit testing will target specific frontend JSX elements for the frontend, ensuring users are able to utilize the full functionality of our system without bugs. For the flask backend, we will test various processes, ensuring the flask server correctly communicates with external sources. Integration testing will focus on assembling JSX elements on the frontend top-down, as we look at the encompassing elements then hone in on smaller ones. For the backend, we want to ensure each process communicates correctly with each other and gives us a desired result. Then we also want to ensure the backend and frontend send and receive correct information too. For regression testing, we want to perform unit testing again to ensure when the system comes together all parts work correctly. Finally, for validation testing, we will refer back to the project requirements and make sure they are being fulfilled.

System testing will not be tested as we do not intend to deploy our program at the moment, but if we were to perform system testing, it would focus on the environment in which we deploy our front and backend.

Test Strategy

Test Type	Description	Notes
Unit Test	Test every unique component to ensure that specific component works correctly	Consider different test techniques for JSX elements and Flask server
Integration Test	For both backend and frontend, ensure each unit works correctly with each other	Also look at how backend and frontend communicate with each other
Regression Test	Rerun unit tests after integration tests to make sure each component is working correctly	
Validation Test	Once frontend and backend are working correctly, ensure requirements are fulfilled when running	

Test Objectives

Our overall test objectives include achieving correct functionality for individual elements on both front and backend, retrieving and returning the desired data for corresponding input. We also want to make sure components work with each other without error and efficiently as we want to ensure a smooth user experience. We'd also like to test for requirements that are not only functional but also behavioral, as we'd like to put an emphasis on customer experience and satisfaction.

Test List

Test ID	Test Name	Use Case	Tester
U_FRONT1	Display Correct Sentiment	Get Stock Sentiment	Nicholas
U_BACK1	Get Correct Company Name	Correct Company Misspellings	Andriy
U_BACK2	Check Valid Ticker	Retrieve Ticker Symbol	Andriy
U_BACK3	Get Related News Articles	Find News Articles	Nicholas
U_BACK4	Check Stock Data from Ticker	Retrieve Financial Data	Aziz
U_BACK5	Ensure Fast Access	Check Cache	Nicholas
U_BACK6	Asynchronously Get Article Info	Load News Articles, Browse News Articles	Andriy
U_BACK7	Get Article Sentiment	Assign Article Sentiment	Andriy
I_FRONT1	Ensure Responsive Design	Get Stock Sentiment	Nicholas
I_BACK1	Ensure API Functionality	Send JSON Data	Andriy
I_SYSTEM1	Display Data Based on API Call	Make API Call	Aziz
V_SYSTEM1	Test User Experience	All	Aziz

Test Schedule

Start Date	End Date	Tests To Perform	Test IDs
11/18	11/22	Frontend Unit Tests	U_FRONT1
11/18	11/22	Backend Unit Tests	U_BACK1, U_BACK2, U_BACK3, U_BACK4, U_BACK5, U_BACK6, U_BACK7
11/23	11/24	Frontend Integration Tests	I_FRONT1
11/23	11/24	Backend Integration Tests	I_BACK1
11/25	11/26	Overall Integration Tests	I_SYSTEM1
11/26	11/27	Frontend Regression Tests	U_FRONT1
11/26	11/27	Backend Regression Tests	U_BACK1, U_BACK2, U_BACK3, U_BACK4, U_BACK5, U_BACK6, U_BACK7
11/28	11/29	Validation Tests	V_SYSTEM1

Display Correct Sentiment

Use Case: Get Stock Sentiment
Test Case ID: U_FRONT1
Test Designed by: Nicholas
Test Priority (Low/Medium/High): High
Test designed date: 11/15
Module Name: SearchPage
Test Executed Date: 11/19
Test Executed by: Nicholas
Description: Input values for sentiment to display in the frontend in an illustrative way
Objective: Use dummy data to simulate real sentiment information and ensure correct values are displayed (for sentiment between -1 and 1)
Pre-conditions: User is on the search page having search for a query

Step	Test Steps	Test Data	Expected Result	Notes
1	Test for positive sentiment	{sentiment:0.5}	Green Arrow indicating positive sentiment	Desired Behavior
2	Test for negative sentiment	{sentiment:-0.5}	Red Arrow indicating negative sentiment	Desired Behavior
3	Test for 0 sentiment	{sentiment:0.0}	Gray bar indicating neutral sentiment	Desired Behavior
4	Test for no sentiment	{sentiment:}	No sentiment information displayed	
5	Test for out of bounds sentiment	{sentiment:-10}	No sentiment information displayed	
6	Test for non-number sentiment	{sentiment:“asdf”}	No sentiment information displayed	

Get Correct Company Name

Use Case: Correct Company Misspellings
Test Case ID: U_BACK1
Test Designed by: Andriy
Test Priority (Low/Medium/High):
Test designed date: 11/15
Module Name: Flask API
Test Executed Date: 11/19
Test Executed by: Andriy
Description: The API should be able to detect misspelled company names and correct them
Objective: To correct company names if they are misspelled so that we can find the ticker
Pre-conditions: A company is searched for through the API

Step	Test Steps	Test Data	Expected Result	Notes
1	Search for a company missing a letter	palntir	Palantir	
2	Search for a company with a misplaced letter	ubre	Uber	
3	Search for a company with an extra letter	disneyy	Disney	

Check Valid Ticker

Use Case: Retrieve Ticker Symbol
Test Case ID: U_BACK2
Test Designed by: Andriy
Test Priority (Low/Medium/High): Medium
Test designed date: 11/15
Module Name: Flask API
Test Executed Date: 11/20
Test Executed by: Andriy
Description: To get stock data, we need to find the ticker for a given company. For example, Google->GOOGL
Objective: Ensure correct tickers are found based on a valid company search
Pre-conditions: A company name has been searched for

Step	Test Steps	Test Data	Expected Result	Notes
1	Search for a valid company	Google	GOOGL	
2	Search for an invalid company	DSLFJ(@23	Raise an error: Invalid Company	Should terminate program and return nothing
3	Search for a valid company but with many similarly named companies	Apple	AAPL	Should get the most popular company if ambiguous

Get Related News Articles

Use Case: Find News Articles
Test Case ID: U_BACK3
Test Designed by: Nicholas
Test Priority (Low/Medium/High): Medium
Test designed date: 11/15
Module Name: Backend Flask API
Test Executed Date: 11/20
Test Executed by: Nicholas
Description: Once getting a valid company name, we have to search for the company in the news to get relevant articles
Objective: To get a list of urls of related articles given a company name
Pre-conditions: A valid company name is passed

Step	Test Steps	Test Data	Expected Result	Notes
1	Give a valid company name	“Uber”	A set of article urls related to uber stock	
2	Give an invalid company name	“asdf92”	An empty set of no article urls	
3	Give a company name that can be interpreted in many ways	“Apple”	A set of article urls related to apple stock	There should be no articles about the apple fruit in the set of urls

Check Stock Data from Ticker

Use Case: Retrieve Financial Data
Test Case ID: U_BACK4
Test Designed by: Aziz
Test Priority (Low/Medium/High): High
Test designed date: 11/15
Module Name:
Test Executed Date: 11/20
Test Executed by: Aziz
Description: We want to display and take past data for a given stock based on its ticker through an external API
Objective: Check if when submitting a ticker, corresponding information about the stock is returned
Pre-conditions: A stock ticker is passed through

Step	Test Steps	Test Data	Expected Result	Notes
1	Submit a valid ticker	GOOGL	Historical and Current stock data for Google	
2	Submit an invalid ticker	123JKL	No stock data returned, raise an error	
3	Submit no ticker		No stock data returned, raise an error	

Ensure Fast Access

Use Case: Check Cache
Test Case ID: U_BACK5
Test Designed by: Nicholas
Test Priority (Low/Medium/High): Medium
Test designed date: 11/15
Module Name: Data cache
Test Executed Date: 11/21
Test Executed by: Nicholas
Description: Getting the same financial data each time a search query is made is time consuming and a waste. Use a cache to make sure we can get preloaded data fast.
Objective: Use a cache to speed up the time it takes to load searched for data
Pre-conditions: company information has been cache according to its ticker

Step	Test Steps	Test Data	Expected Result	Notes
1	Look for a cached ticker	cache[‘GOOGL’]	Corresponding stock information for google from the cache	
2	Look for an uncached ticker	cache[‘AAPL’]	Corresponding stock information for apple from the external APIs	Information will be searched for and then stored in cache, so next time access is fast
3	Look for an invalid ticker	cache[‘1234S’]	No stock information, raise an error	Cache empty result so next time this query is made we get invalid results fast

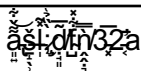
Asynchronously Get Article Info

Use Case: Load News Articles, Browse News Articles
Test Case ID: U_BACK6
Test Designed by: Andriy
Test Priority (Low/Medium/High): Medium
Test designed date: 11/15
Module Name: Asynchronous loader
Test Executed Date: 11/21
Test Executed by: Andriy
Description: We want to load articles and extract information from them but doing so sequentially takes a lot of time, so we want to use asynchronous loading
Objective: Load articles asynchronously, getting title, text, and publisher information for each article so that the overall load time is lower
Pre-conditions: a set of article urls are given to the system

Step	Test Steps	Test Data	Expected Result	Notes
1	Give a valid set of urls	[url1, url2, …]	Get title, text, and publisher information for each article asynchronously	
2	Give an invalid set of urls	[valid url1, invalid url2, …]	Get title, text, and publisher information only for each valid article, notify the user of invalid urls	Omit invalid articles in the response
3	Give an empty set of urls	[]	Nothing returned	

Get Article Sentiment

Use Case: Assign Article Sentiment
Test Case ID: U_BACK7
Test Designed by: Andriy
Test Priority (Low/Medium/High): High
Test designed date: 11/15
Module Name: Sentiment Analyzer
Test Executed Date: 11/22
Test Executed by: Andriy
Description: For each article, we need the sentiment of each article so that we can return that value to the user
Objective: Use sentiment analysis to get the sentiment of an article given its title and text
Pre-conditions: an article has been parsed and data sent to system

Step	Test Steps	Test Data	Expected Result	Notes
1	Send valid article text and title data	Valid title, valid text	Sentiment ranging from -1 to 1 is returned based on the analysis	
2	Send broken text		No sentiment returned, an error is raised	
3	Send no text	“”	No sentiment is returned, and an error is raise	

Ensure Responsive Design

Use Case: Get Stock Sentiment
Test Case ID: I_FRONT1
Test Designed by: Nicholas
Test Priority (Low/Medium/High): High
Test designed date: 11/15
Module Name: SearchPage, LandingPage
Test Executed Date: 11/23
Test Executed by: Nicholas
Description: The site must work on mobile, tablet, and desktop devices and display data in a readable manner
Objective: Ensure all components fit on the page and in their respective containers without overflow
Pre-conditions: User loads any page

Step	Test Steps	Test Data	Expected Result	Notes
1	Load pages on mobile	Use mobile to load “/” and “/search”	Page should fit on the screen and no article should overflow vertically or horizontally	Use different values for stock predictions
2	Load pages on tablet	Use tablet to load “/” and “/search”	Page should fit on the screen and no article should overflow vertically or horizontally	Use different values for stock predictions
3	Load pages on desktop	Use desktop to load “/” and “/search”	Page should fit on the screen and no article should overflow vertically or horizontally	Use different values for stock predictions

Ensure API Functionality

Use Case: Send JSON Data
Test Case ID: I_BACK1
Test Designed by: Andriy
Test Priority (Low/Medium/High): High
Test designed date: 11/15
Module Name: All Backend Modules
Test Executed Date: 11/23
Test Executed by: Andriy
Description: The backend functions in a way that the backend gives a JSON output about certain stocks based on input. This test checks to see the returned value is valid
Objective: Ensure the api gives a valid output upon an input. It should return information about the stock if it can find it as well as past information
Pre-conditions: All backend functionality is set up

Step	Test Steps	Test Data	Expected Result	Notes
1	Search for a valid company	Query: “google”	Stock data for Google (current and past)	
2	Search for a misspelled company	Query: “palntir”	Stock data for Palantir (current and past)	
3	Search for no company	Query:	{}	Empty JSON result
4	Search for an invalid company	Query: “123asdf”	{}	Empty JSON result as no data is found

Display Data Based on API Call

Use Case: Make API Call
Test Case ID: I_SYSTEM1
Test Designed by: Aziz
Test Priority (Low/Medium/High): High
Test designed date: 11/15
Module Name: Backend API and Frontend
Test Executed Date: 11/25
Test Executed by: Aziz
Description: Frontend should send information and correctly display what is returned
Objective: Ensure frontend and backend are working correctly as frontend makes an api call and parses the response properly
Pre-conditions: API and React programs running

Step	Test Steps	Test Data	Expected Result	Notes
1	Search for a valid company on frontend	“apple”	Sentiment, stock, and article information should be displayed on frontend for apple	
2	Search for a misspelled company on frontend	“palntir”	Sentiment, stock, and article information should be displayed on frontend for palantir	
3	Search for an invalid company on frontend	“123asdf!3e”	No information should be displayed	No info found->no info displayed

Test User Experience

Use Case: All
Test Case ID: V_SYSTEM1
Test Designed by: Aziz
Test Priority (Low/Medium/High): High
Test designed date: 11/15
Module Name: All Modules
Test Executed Date: 11/29
Test Executed by: Aziz
Description: Users must be able to have a fluid experience getting the correct information
Objective: Ensure a user can get accurate stock information efficiently
Pre-conditions: Backend and Frontend are connected and ready for production

Step	Test Steps	Test Data	Expected Result	Notes
1	Input valid company	“Google”	Correct financial data for google should appear quickly	
2	Input misspelled company	“Palntir”	Correct financial data for palantir should appear quickly as the name gets corrected	Backend should correct the query
3	Input invalid company	“asdfasdf”	No data should be displayed, with an error message saying no results found	