

F.T.A.H.

Hayden Pour, Julian Beaulieu, Mohamed Rayyan, Padraic Reilly, Nicholas Cardinal

Artifact

Platform:	1
Platform:	1
Programming Languages:	1
Feature List:	1
Important Features:	1
Implement if we have time, because we need to use data outside of the database:	2
Sprints:	
Sprint 2	2
Sprint 3	3
Sprint 4	6
Sprint 5	10

Historical daily prices and volumes of all U.S. stocks

Data we have:

1. Trading Date
2. Opening Price
3. Daily Price (High)
4. Daily Price (Low)
5. Closing Price
6. Volume Sold

Platform:

Web Application / Desktop Application

Programming Languages:

- JavaScript
- HTML
- CSS
- Python

Feature List (Question of Interest):

Important features:

1. *How does a stock's price change over a given period?*
2. *How does the volume of stock sold change over a given period?*

3. *What is the moving average for a stock over a given period?*
4. *What is the highest/lowest closing price over a given period?*
5. *Which stock has the largest margin over a given period?*
6. *What days had the largest increases or decreases in price? (Useful for correlating to real world events)*
7. *How did a specific stock's daily change compare to the market average change? (High or low performing stocks)*

Implement if we have time, because we need to use data outside of the database:

1. *How do real-world events affect stock prices?*
2. *What is the predicted opening and closing price of a stock?*
3. *What stocks are the best to trade for the day?*

Sprint-2:

Action Items:

- **Client/UI**
 - **JS Promises**
 - **Create import routine for data retrieved from server**
 - **Sort retrieved labels alphabetically**
 - **Sort retrieved data by date**
 - **Display graph from label click**
 - **Display other various information on label click**
- **Server**
 - **Update csv file name to stock ticker name - stock name**
 - **Return labels in chunks to client backend**
 - **Parallel processing when importing from csv**

Tests:

- **Client/UI**
 - **JS Promises**
Correct Output: The user's data will load completely from server before displaying information to the UI
 - **Create import routine for data retrieved from server**
Correct Output: The website displays stock information when user clicks the stock they want to view
 - **Sort retrieved labels alphabetically**
Correct Output: The website will display the stock list in alphabetical order
 - **Sort retrieved data by date**
Correct Output: The data received by the server is properly sorted by date
 - **Display graph from label click**
Correct Output: When a label is clicked, the website will display all pertinent information on the right side of the UI

- **Server**
 - **Update csv file name to stock ticker name - stock name**
Correct Output: All files in our dataset are renamed correctly
 - **Return labels in chunks to client backend**
Correct Output: When called, the website requests data from the server and it is returned in a format that can be understood by the website
 - **Parallel processing when importing from csv**
Correct Output: Multiple files can be read from at any given time

Sprint-3:

Features:

Feature 1:

User Story: As an administrative user, I would like to delete, and insert and reload, new stock data into the CSV files and know when it is done.

Tasks:

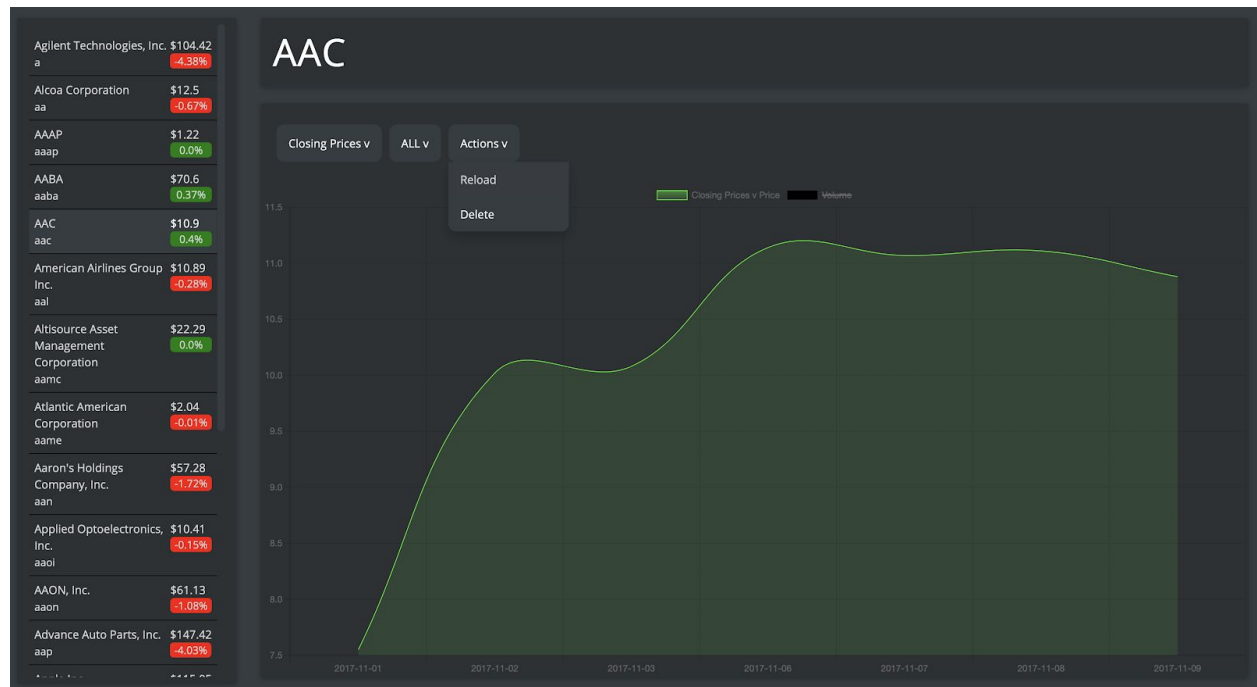
- Step 1: Get access to new stock data from stock data source
[Completed by]
- Step 2: Write functions that inserts lines into CSV and memory
[Completed by]
- Step 3: GUI is updated to reflect the newly inserted data
[Completed by]
- Step 4: Write function that deletes a selected stock from memory (front and back)
[Completed by]
- Step 5: Back up deleted stock by keeping reference in CSV
[Completed by]
- Step 6: GUI is updated to reflect the newly deleted data from portfolio
[Completed by]
- Step 7: Write function that reloads a selected stock from server
[Completed by]
- Step 8: Reload chart with newly imported data
[Completed by]

Tests:

- Step 1 Test: Get access to new stock data from stock data source
Correct Output: Backend server can receive data from some stock data source
 [Completed by]

- Step 2 Test: (INSERT) Write functions that inserts records to CSV file
Correct Output: The inserted records are successfully written to the CSV file
[Completed by]
- Step 3 Test: (INSERT) Stock chart UI is updated with new data
Correct Output: The chart now shows the latest data after the data was pulled from server
[Completed by]
- Step 4 Test: (DELETE) Write functions that delete records (server and local)
Correct Output: Once the delete function is ran, the server's dictionary should have one less record
[Completed by]
- Step 5 Test: (DELETE) Stock List UI will remove deleted stock from list
Correct Output: The chart now shows one less stock in the stock list and the chart data is cleared
[Completed by]
- Step 6 Test: (RELOAD) Write function that reloads a selected stock from server
Correct Output: Local datastore now hold a newly pulled set of data from the server
[Completed by]
- Step 7 Test: (RELOAD) Reload chart with newly imported data
Correct Output: Once the data is reloaded from the server the chart is reloaded for the user to see.
[Completed by]

UI Example



Sprint-4:

Features:

Feature 1:

User Story: As a user, I would like to view my stock data in a candle graph

Tasks:

- Step 1: Display Candle Chart on web page with dummy data
[Completed by]
- Step 2: Create a function to gather correct data to display on chart
[Completed by]
- Step 3: Create function to reload chart with newly filtered data
[Completed by]

Tests:

- Step 1 Test: Display Candle Chart on web page with dummy data
Correct Output: The website will display a candle chart
[Completed by]
- Step 2 Test: Create a function to gather correct data to display on chart
Correct Output: The website will display a candle chart with data selected the user
[Completed by]
- Step 3 Test: Create function to reload chart with newly filtered data
Correct Output: The websites UI will refresh when the user requests a change in stock
[Completed by]

Feature 2:

User Story: As a user, I would like to know a year over year percent change by stock

Tasks:

- Step 1: Add YOY to date selector drop down
[Completed by]
- Step 2: Calculate year over year percentage on backend by ticker
[Completed by]
- Step 3: Pass data back to front end and store in object
[Completed by]
- Step 4: Display year over year percentage on chart
[Completed by]

Tests:

- Step 1 Test: Add YOY to date selector drop down
Correct Output: Year Over Year list item is visible in UI drop down
[Completed by]
- Step 2 Test: Calculate year over year percentage on backend by ticker

Correct Output: Python script to confirm accuracy of YOY percentages
[Completed by]

- Step 3 Test: Pass data back to front end and store in object

Correct Output: Data is retrieved from server and output to console in the correct format
[Completed by]

- Step 4 Test: Display year over year percentage on chart

Correct Output: Data that is retrieved from server in previous step is visible and accurate on the chart tool
[Completed by]

Feature 3:

User Story: As a user, I would like to be able to switch between stocks and ETFs

Tasks:

Step 1: Create UI object to switch between ETF and Stock
[Completed by]

Step 2: Create a function that loads ETF or Stock depending on UI
[Completed by]

Step 3: Create function to reload stock list with newly filtered data
[Completed by]

Tests:

- Step 1 Test: Create UI object to switch between ETF and Stock

Correct Output: The website would display radio buttons that say “ETF” and “Stocks”
[Completed by]

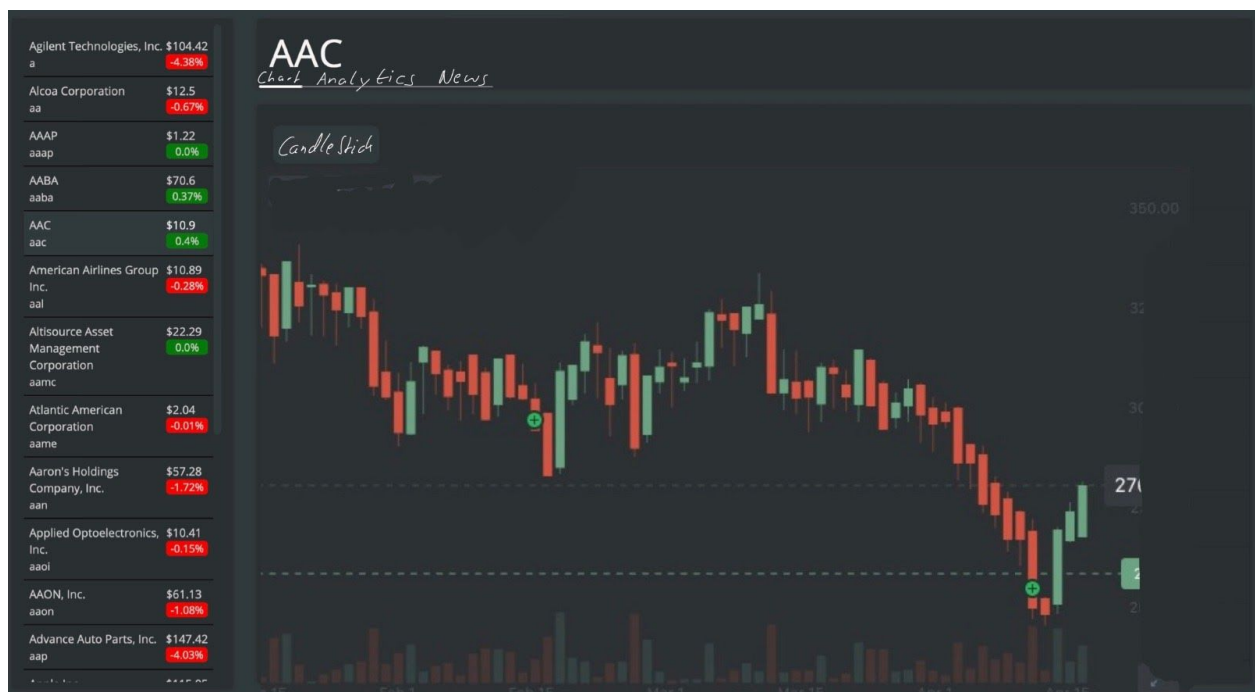
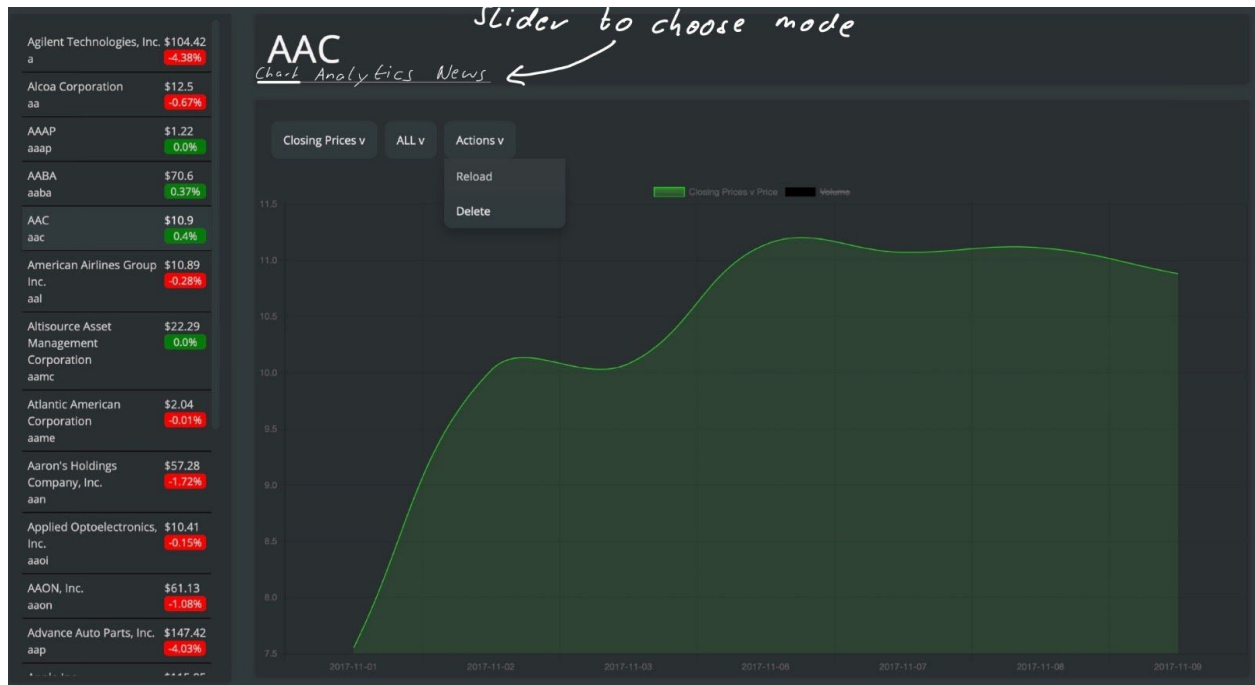
- Step 2 Test: Create a function that loads ETF or Stock depending on UI

Correct Output: The server will return the selected type of data to the client
[Completed by]

- Step 3 Test: Create function to reload stock list with newly filtered data

Correct Output: The website will refresh the stock list when the user changes the selected type
[Completed by]

UI Example



Agilent Technologies, Inc.	\$104.42	
a	-4.38%	
Alcoa Corporation	\$12.5	
aa	-0.67%	
AAAP	\$1.22	
aaap	0.0%	
AABA	\$70.6	
aaba	0.37%	
AAC	\$10.9	
aac	0.4%	
American Airlines Group Inc.	\$10.89	
aal	-0.28%	
Altisource Asset Management Corporation	\$22.29	
aamc	0.0%	
Atlantic American Corporation	\$2.04	
aame	-0.01%	
Aaron's Holdings Company, Inc.	\$57.28	
aan	-1.72%	
Applied Optoelectronics, Inc.	\$10.41	
aaol	-0.15%	
AAON, Inc.	\$61.13	
aaon	-1.08%	
Advance Auto Parts, Inc.	\$147.42	
aap	-4.03%	

AAC

Chart Analytics News



Sprint-5:

Features:

Feature 1:

User Story: As a user, I would like to see the cross over and have the graph indicate if it is a golden or a death cross

Tasks:

- Step 1: Display chart and 2 moving averages on web page with dummy data
- Step 2: Create a function to gather correct data to display on chart
- Step 3: Create function to reload chart with newly filtered data

Tests:

- Step 1 Test: Display 2 moving averages in chart on web page with dummy data
Correct Output: The website will display the moving average chart.
- Step 2 Test: select a stock, and a analytics filter
Correct Output: The webpage must display the moving average graph of the filtered data.
- Step 3 Test: reload the chart with the filters
Correct Output: The webpage must display an updated moving average graph if new data was appended to the stock or else the same graph.

Feature 2:

User Story: As a user, I would like to see the velocity.

Tasks:

- Step 1: Display chart and velocity on web page with dummy data
- Step 2: Create a function to gather correct data to display on chart
- Step 3: Create function to reload chart with newly filtered data

Tests:

- Step 1 Test: Display Chart and velocity on web page with dummy data
Correct Output: The website will display the chart and velocity
- Step 2 Test: select a stock, and a analytics filter
Correct Output: The webpage must display the velocity graph of the filtered data.
- Step 3 Test: reload the chart with the filters
Correct Output: The webpage must display an updated velocity graph if new data was appended to the stock or else the same graph.

Feature 3:

User Story: As a user, I would like GitHub CI to work and test all the unit tests

Tasks:

Step 1: Figure out what CI is.

Step 2: Create more unit tests for the client and the server

Step 3: Setup CI to run unit tests.

Tests:

NA

UI Example

