CS1013-report-33

Programming Project

GraphApp Application

Team Members Group 33:

Keira Gatt (19334557)
Cian O'Gorman (19334475)
David Olowookere (19335061)
Samuel Alarco Cantos (19313268)

Date of last revision: 24/04/2020

Introduction

Objectives

Our objective, as described by the project specifications, was to design a stocks data visualization program that would enable the user to search and visualize a set of stocks' related data in a variety of ways. The program should therefore offer an array of functionality to enable the user to search the data with as many parameters as possible, with an emphasis on flexibility, speed and size of dataset. Furthermore, it should offer different visualizations (table views, line graphs, bar charts etc.) to allow the user to properly interpret and analyze the data, through a fluid and intuitive interface.

Outcomes and Features

Our software GraphApp successfully fulfills the above objectives in the following way –

- Through the use of a MySQL server and a Database API to be used by the rest of the program, it allows for the search and manipulation of extremely large datasets, while maintaining speed and scalability.

- Thanks to the unified Search Box design and date selectors, the user can filter the data using a wide variety of parameters, including: **date range, tickers, exchange, industry, and sector.** Most of this can be done through a single search box, reducing interface clutter and complexity.

- A **Hotbar** functionality further allows the user to store tickers of interest for inclusion and analysis in further searches, adding to the flexibility of the design.

- A main table visualization allows for quick inspection of queried data, displaying the essential information related to each stock data point.

- The data in a specific date range can further be displayed in a line graph view, to analyze the development of a certain data type across a span of time (e.g. closing price over selected days).

- Finally, a bar chart view displays aggregate data such as means for different data types, allowing for further comparison with other stocks over the desired period of time.

Design Outline and Extended Functionality

The program functionality is divided into 3 main areas, that also served to divide the work in the group as will be described later. These are: **Database backend**, **GUI and interface elements**, and **Controller elements** to link GUI and direct the database according to user input.

1. **Database**

   The Database class is responsible for importing and storing the data, and also serves as API for a local MySQL server. The database can operate in two modes: SQL and non-SQL mode. The non-SQL mode imports the daily_prices and stocks .csv files into native Processing Table class objects. It then uses a "stack concept" to iteratively refine a subset of the data using different search functions, until a final result is provided through the *getQueryResult()* function. The search functions provide an API-like functionality to the database that can be used by the controller classes to obtain the required data. This approach is practical with files of up to 100k

entries, but it was problematic with bigger datasets, as the program became too slow. This was a big problem for us, which we solved with SQL and a local MySQL server.

Database offers the same search functionality connecting to a MySQL server when in SQL mode. It effectively hides the SQL layer from the rest of the program, so there is no difference between the SQL and the non-SQL mode in what concerns the usage of the Database query functionality. The Database progressively builds up an SQL query statement, which is executed when the query results are requested. This approach increases speed with large datasets, as the whole query is only executed when all the parameters are specified, as opposed to the previous iterative approach that is only practical with small datasets. Thanks to the MySQL server, we have been able to use the 2GB dataset.

2. Controller Elements

At the heart of GraphApp are 2 classes that control what and how data is displayed, based on user input. The **SearchTools class** provides the user interface functionality to capture the search criteria, parses and validates input, prepares the search query tokens and issues calls to the **Database class** to retrieve data from the master database. Functionality is extended by allowing the user to enter multiple tokens and regex-based wildcards as part of the field input, to search by Industry, Sector and Exchange besides Ticker, to include Hot Bar entries in the search queries and to scope searches based on a date range. Other methods in this class keep track of what data is being displayed at any point in time, updates table and graph views immediately with new search results and resets view states back to main data whenever the user exits search mode through the search cancel button.

The other class in this category is **DataOps**, which is essentially a data provider for the **LineGraph** and **BarChart** classes. It is responsible to select the correct data source and to prepare the X and Y data series for each type of graph. All data points, whether derived values (such as those used for the Bar Charts e.g. Standard Deviation) or raw values (such as those used for the Line Graphs) are generated by methods in this class.

3. GUI and Visualizations

The **BarChart** and **LineGraph** classes implement the required functionality to view data as bar charts and line graphs and to scale the X and Y axis according to the data series provided by the **DataOps** class. The functionality is extended with use of navigation buttons that allows the user to cycle through numerous graph types and data sets. In addition, the bar chart also displays the actual values upon mouse hover in the display area whereas the line graph provides access to the Ticker's data series with the use of the quick access buttons.

The **TagTable** class is responsible for displaying the main table that can be seen when starting the program. It provides data to the user such as the open, close, and volume of a given ticker. It also allows the user to add Tickers to the hotbar through the use of the **"+ADD" Button.**

The **HotBar** class's main function is to provide the user with an on-screen location to store their preference of Tickers. It can be used to provide table data when viewing the **BarChart** or **LineGraph** and also allows the user to add unique Tickers to the a search to then be viewed in the LineGraph

Tasks and Responsibilities

Samuel Alarco Cantos

I was responsible for the main data backend of the application, including the design of a **Database** class to manage the import and initial processing of the data from the provided csv files. In order to use larger datasets, a **local MySQL server** was implemented (only for demonstration). For details refer to Design outline. Additionally, I was responsible for the **LineGraph class**, the **ScrollableContainer class**, and minor classes such as the **Drawable abstract class** and the **ScrollableTextBox class.**

Keira Gatt

My main responsibility was the development of the application's controller elements and one of the visualisation components i.e. the **SearchTools**, **DataOps** and **BarChart** classes, already discussed in the sections above. In addition, I also implemented the button navigation functionality for both bar charts and line graphs and the help and status display on the bottom line of the GUI.

David Olowookere

Most of my work was on the front-end of the program. I expanded on the **Widget** class provided by Gavin O'Doherty, creating a basic **Button** class, a toggleable **Checkbox** subclass, and a **Dropdown** subclass that handles the drawing and events of multiple buttons at once. I also created a **Date_Handler** class that dynamically draws the dates at the top of the program, and lets the user adjust the dates using Dropdowns.

Cian O' Gorman

My main responsibility was the development of the projects **appearance and GUI**, I did this by creating concept images during the first week for the group to follow when proceeding with the project. I then developed the concept images into the project using the **GUI** class. I was also responsible for the **TagTable** class which displays the data as a table, the **HotBar** class which allows the user to add their preference of Tickers to be viewed at the bottom of the screen, and the **GraphicButton** class which utilises PGraphics to draw the "+ADD" buttons on the Table.

Application Description and Operation

1. Environment

GraphApp runs in Processing 3 (3.5.4) using JRE Version 1.8.x (32-bit or 64-bit). The data repository can either be in the form of a CSV file or optionally, it can interface with a MySQL database server.

2. Initialisation

GraphApp loads the initial data set either from the CSV file or optionally by issuing calls to MySQL. It initially displays a table view of all tickers at a default date. Tickers to be entered into the HotBar must be selected from this default ticker list.

3. Help, Status & Search Indicators

The bottom line of the GUI (ie status line) provides context based help tips that update with mouse hover and which are aimed to guide the user while navigating the GUI. Shown also on the status line is the type of data currently in view. Lastly, the Search Box at the top of the GUI indicates which type of query will be invoked when executing a search.

4. Date Range

GraphApp provides start and end dates parameters that define a date range to be used for searching.

5. Table View

This type of view, displayed during initialisation or when activated from the sidebar button, shows either default initial data or search query results (generally grouped by ticker and sorted by date). Tickers that

the user wants to keep for future reference may be added to the HotBar by using the +ADD buttons on the right hand side of the data pane.

6. HotBar

This section of the GUI serves as a keep area where Table View entries can be held for future reference. HotBar entries can be flagged to be included in subsequent searches. Alternatively, any entry in the HotBar can be delisted with the Remove button.

7. Query Type

When a search is invoked, GraphApp will act on the criteria specified in the Search Box. The dropdown box next to the Search Box provides the user with the option to select how the search criteria should be treated. The options are as follows –

Ticker	Search by Ticker name (e.g. AHL, GTT)
Exchange	Search by Stocks Exchange (e.g. NASDAQ)
Industry	Search by Industry (e.g. HOMEBUILDING, MAJOR CHEMICALS)
Sector	Search by Sector (e.g. FINANCE, TECHNOLOGY)

8. Search Function

Search entries are referred to as tokens and any number of tokens is accepted, up to a maximum of 18 characters. Each token represents a search item as defined by the Query Type and can specified on the basis of the following rules –

Canonical	Full name of item to be searched (e.g. AHL for Ticker Query Type / e.g. FINANCE for Sector Query Type)
*** Wildcard**	The search will match any number of literal characters up to the next letter specified, if any. (e.g. A* == AHL, AFSI, AAPL, etc. / e.g. G*T == GTT)
? Wildcard	This is a positional placeholder and will match only one literal character (e.g. G?T == GTT / e.g. A?? == AHL)
Mixed	A search can also be invoked with a mix of Canonical and Wildcard tokens (e.g. AHL, G*, CA?? == AHL, GHDX, GTT, GTN, CAAS)

A successful search will update the current view (table or graphs, as applicable) and the data view status on the status line. Other additional features of the Search Function are as follows –

Edit	The Backspace key can be used to delete the last entered character whereas the DEL key when used will clear the entire Search Box
Execute	Execute query with the Search Button or by pressing the ENTER key

Cancel	Cancel search with Cancel Search Button

9 Line Graph View

Plots Search Results as a set of continuous data series against a timeline on the X-axis.

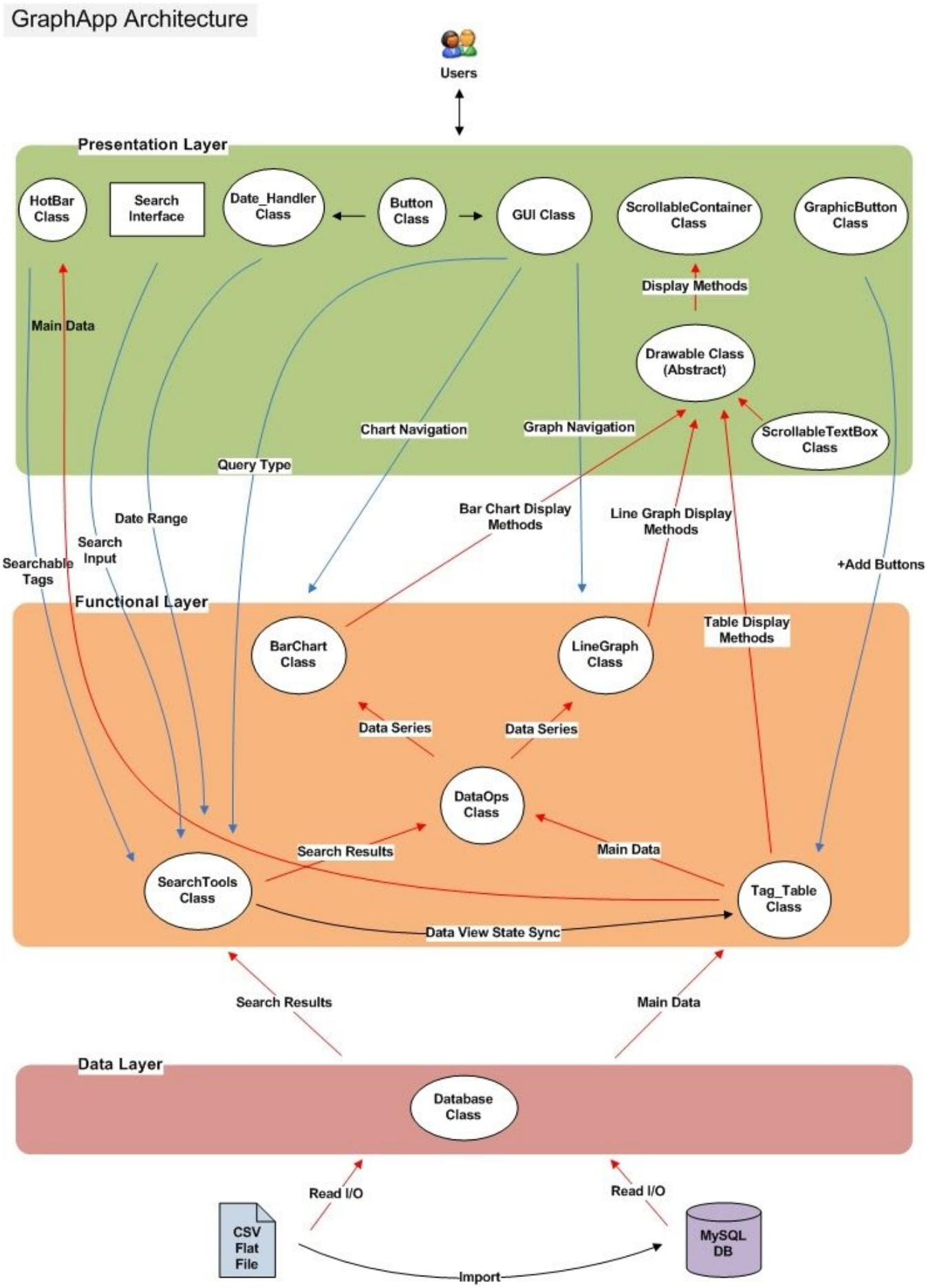
Lateral Arrows	Used to go through the entire data series (open, close, adj_close, low, high, volume) for a Ticker.
Vertical Arrows	Used to cycle through the list of tickers available in the Search Results

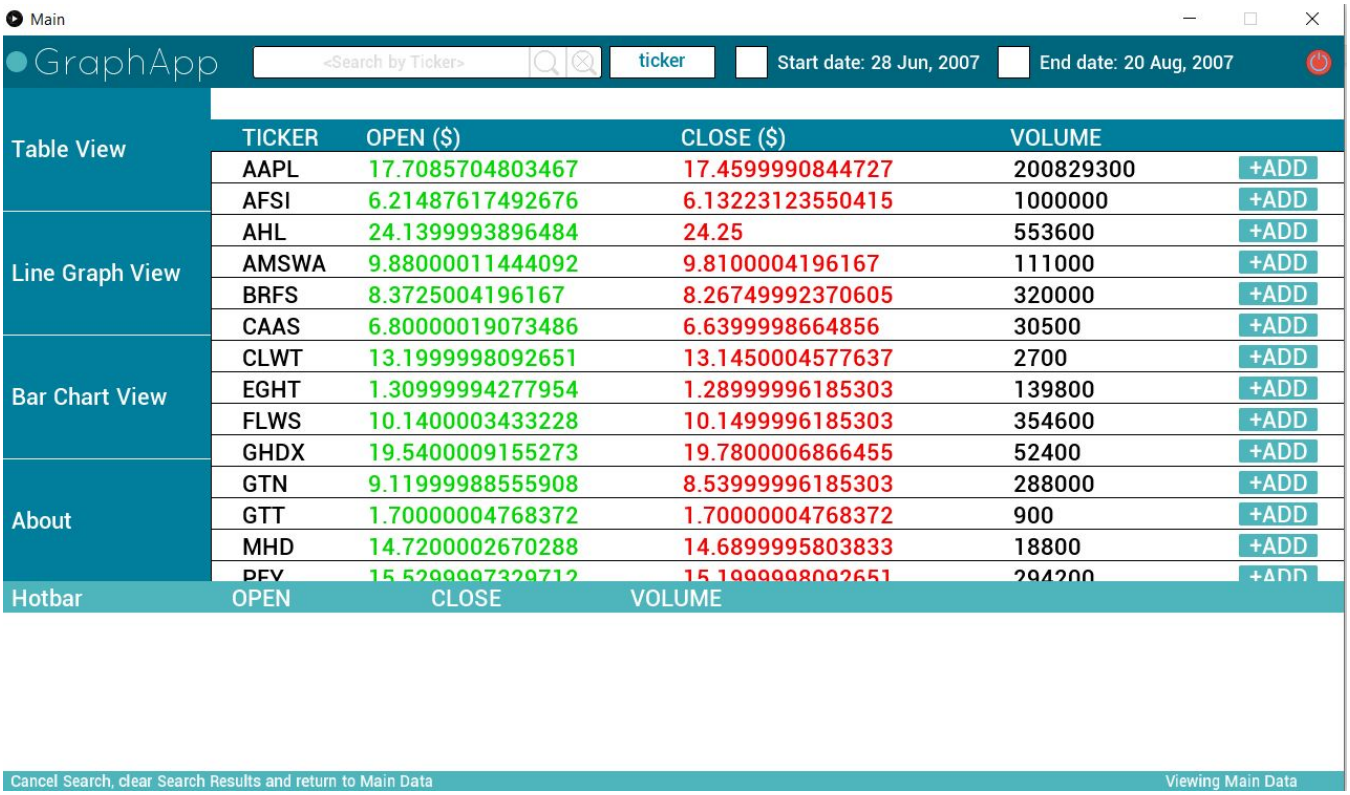
10 Bar Chart View

Displays data series in barchart form. Unlike the Line Graph View, barcharts represent calculations done on the original data, where each derived data set can be accessed via the nav buttons.

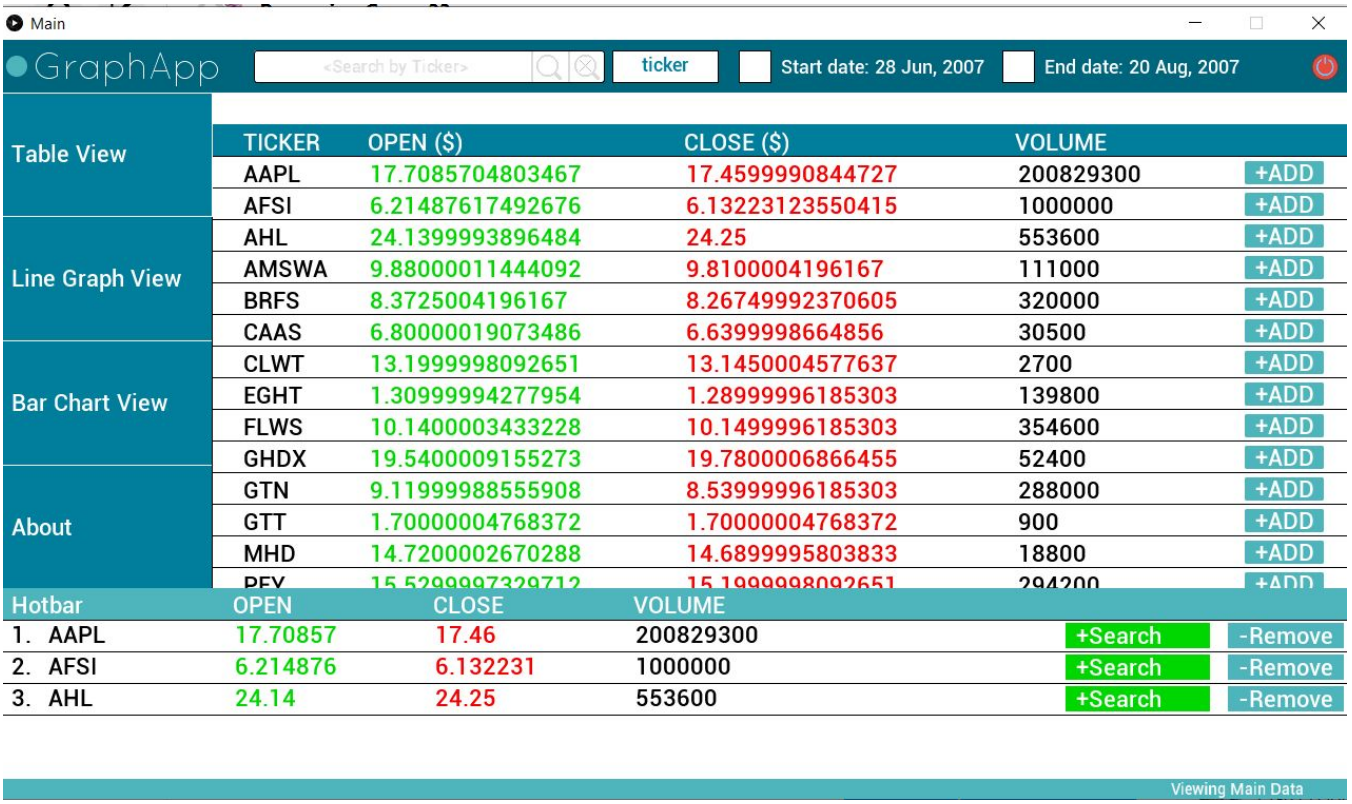
Period Average	Calculates average across indicated date range for each ticker. The Lateral Arrows are used to navigate through each data series in the data set (open, close, adj_close, low, high, volume)
Max Variance	Shows data series for each Ticker, calculated as the difference between the minimum and the maximum values across the indicated date range.
Max Variance % of Minimum	Calculates the Maximum Variance as a percentage of the minimum value across the indicated date range for each ticker.
Standard Deviation	Calculates the Standard Deviation across the indicated date range for each ticker.
Price Change (Open vs Close)	Shows data series as the difference between the daily Open and Close prices across the indicated date range. The Lateral Arrows are used to cycle through the list of tickers available in the Main Data or Search Results
Price Change (Low vs High)	Shows data series as the difference between the daily Low and High prices across the indicated date range.
Price Gain %	Shows data series as the maximum difference between the Open vs Close or Low vs High prices across the indicated data range, shown as a percentage of the lowest Open or lowest Low. The Lateral Arrows are used to navigate between the two data sets, i.e. Open vs Close and Low vs High

Screenshots and Diagrams

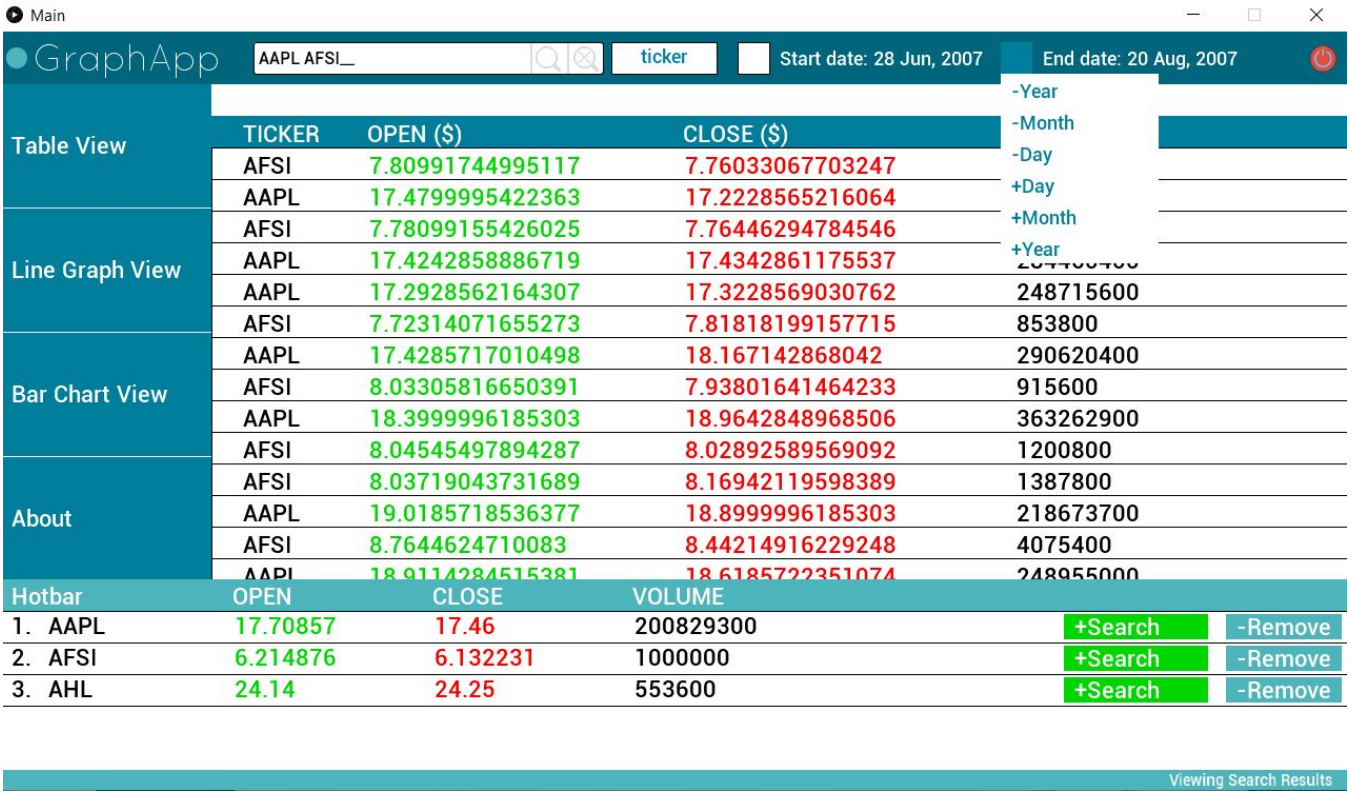




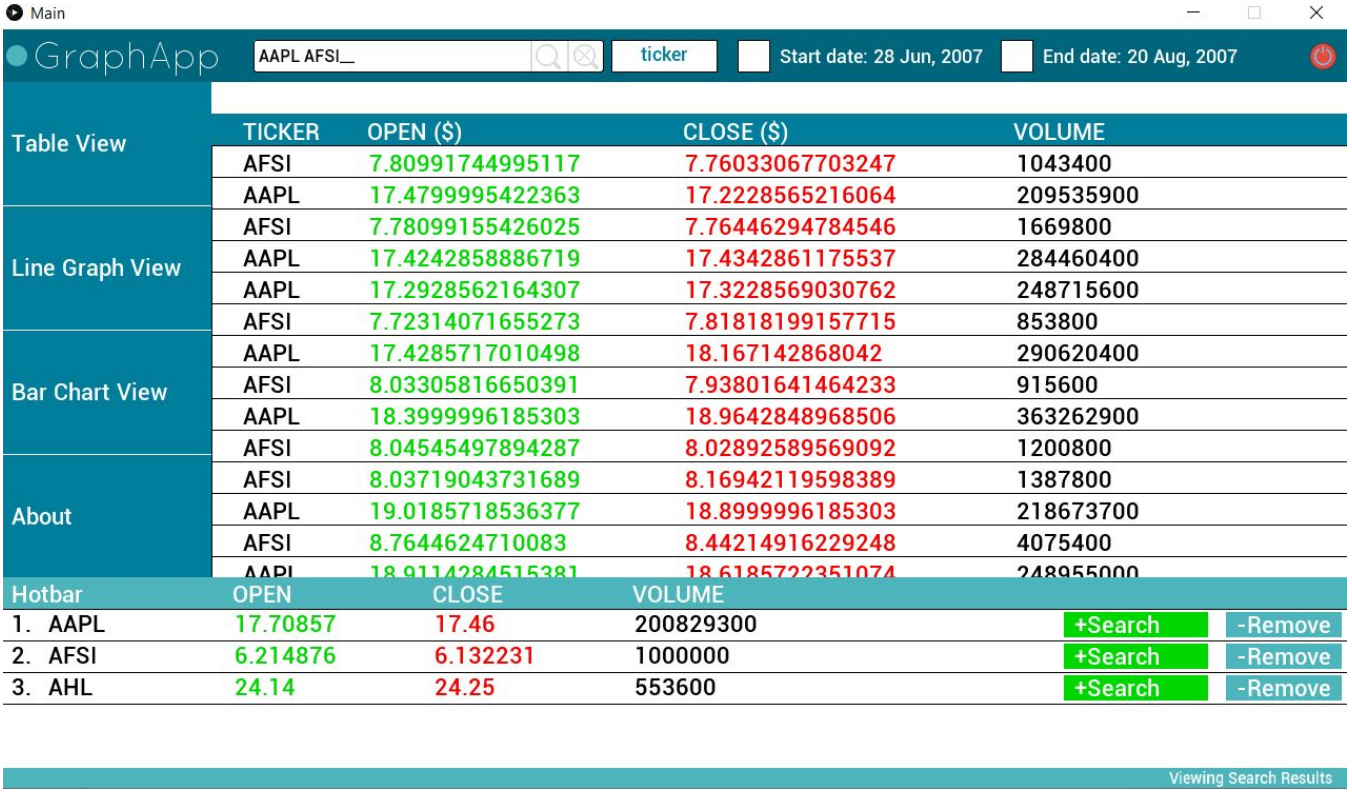
View on initialization



Addition of elements to the HotBar

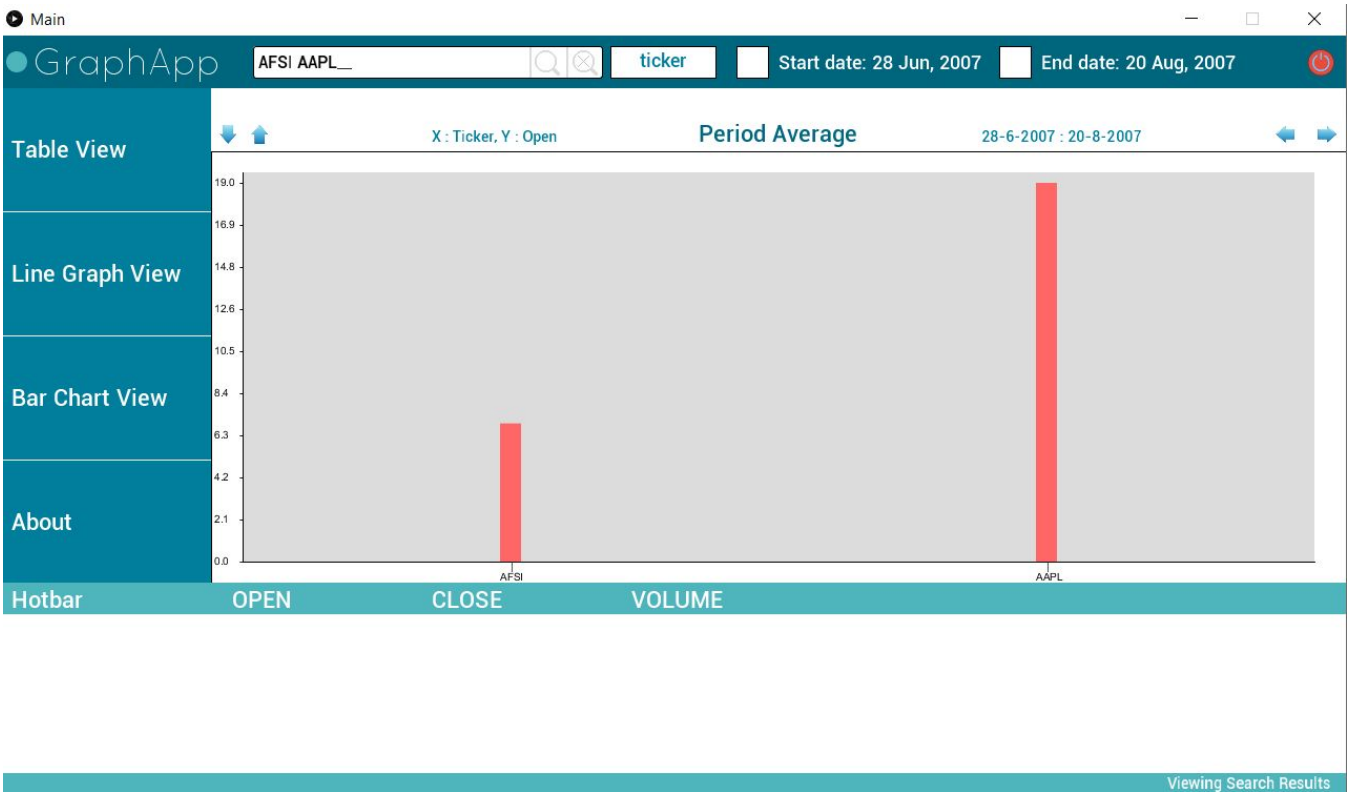


Date selector dropdown menu opening

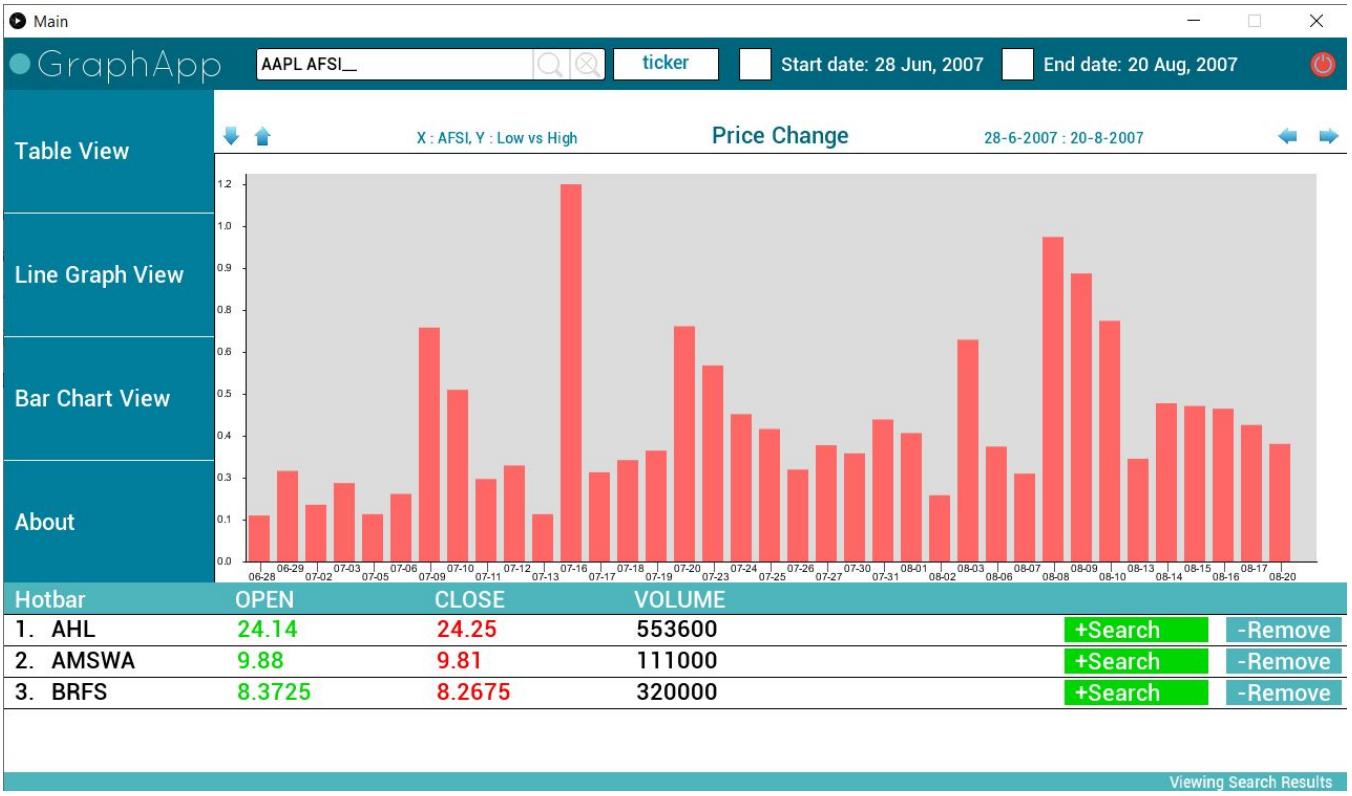


Running query with ticker = “AAPL” AND “AFSI” on dates 28-JUN-2007 => 20-AUG-2007

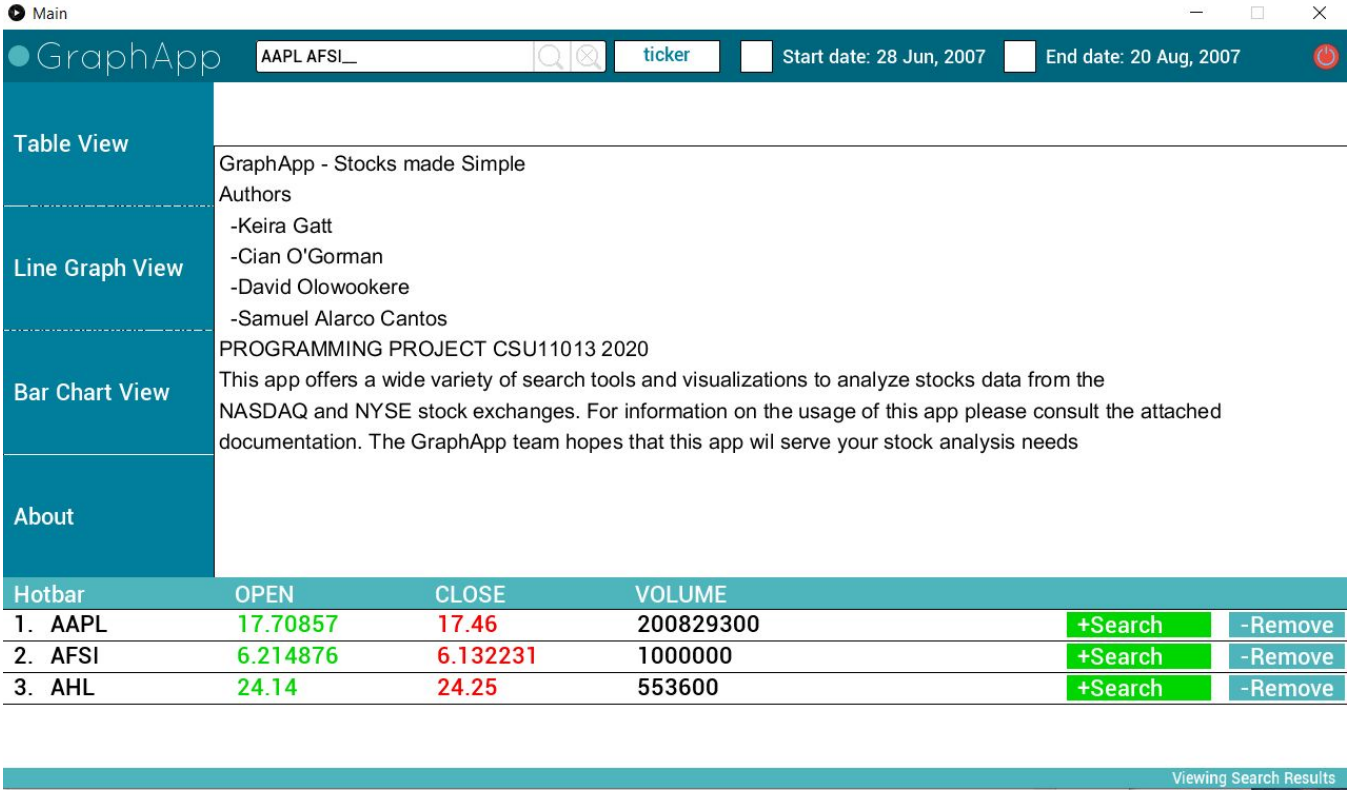
Line Graph view (AFSI open price across specified date range)



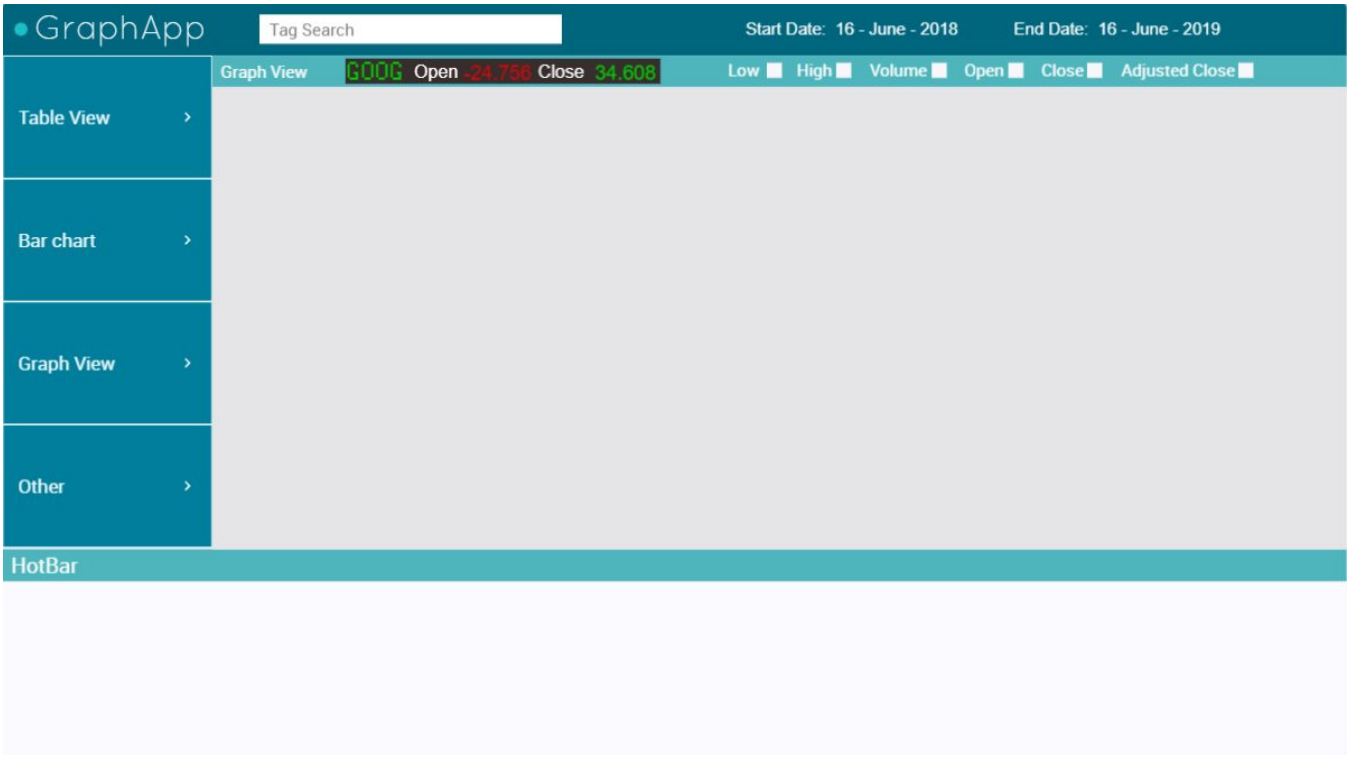
Barchart view of query results: AAPL and AFSI Open Price Period Avg comparison



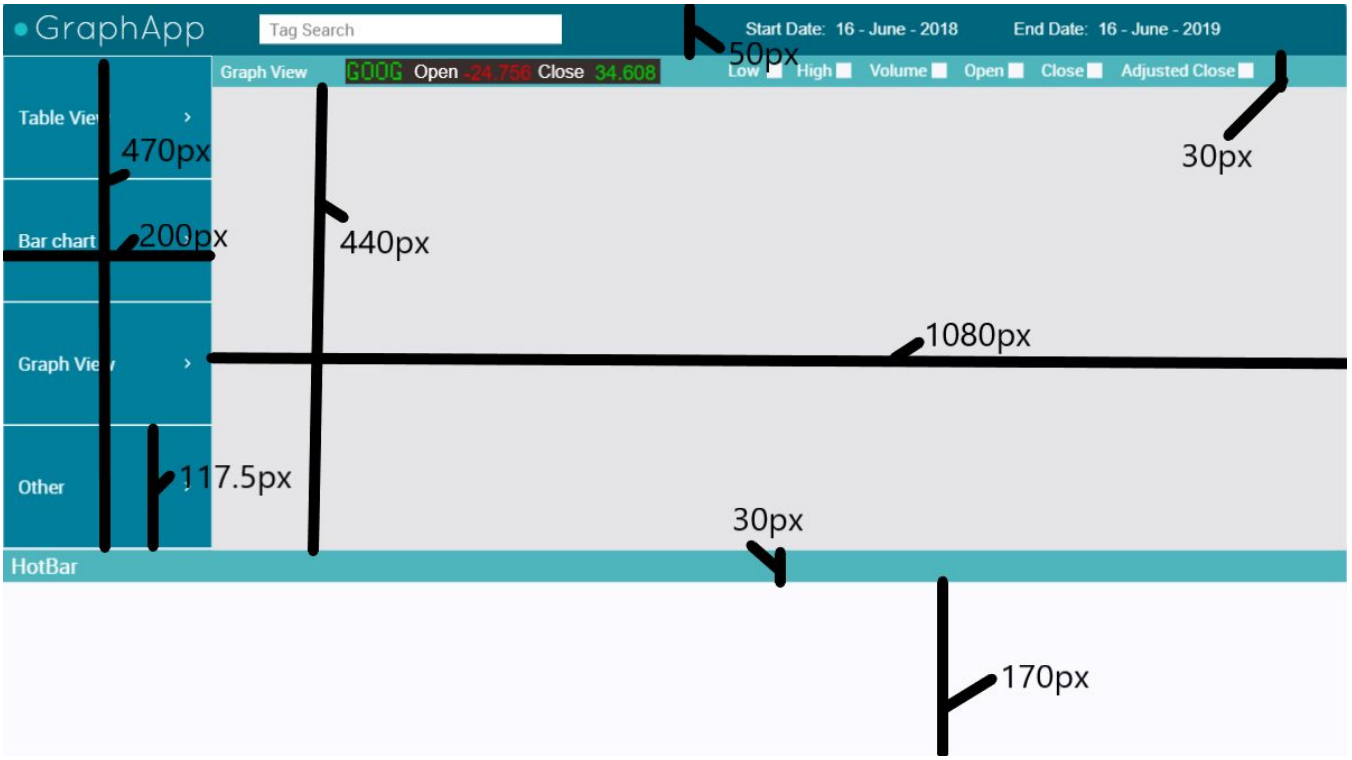
Barchart view of query results: AFSI Absolute Price change evolution over date period (Low vs High)



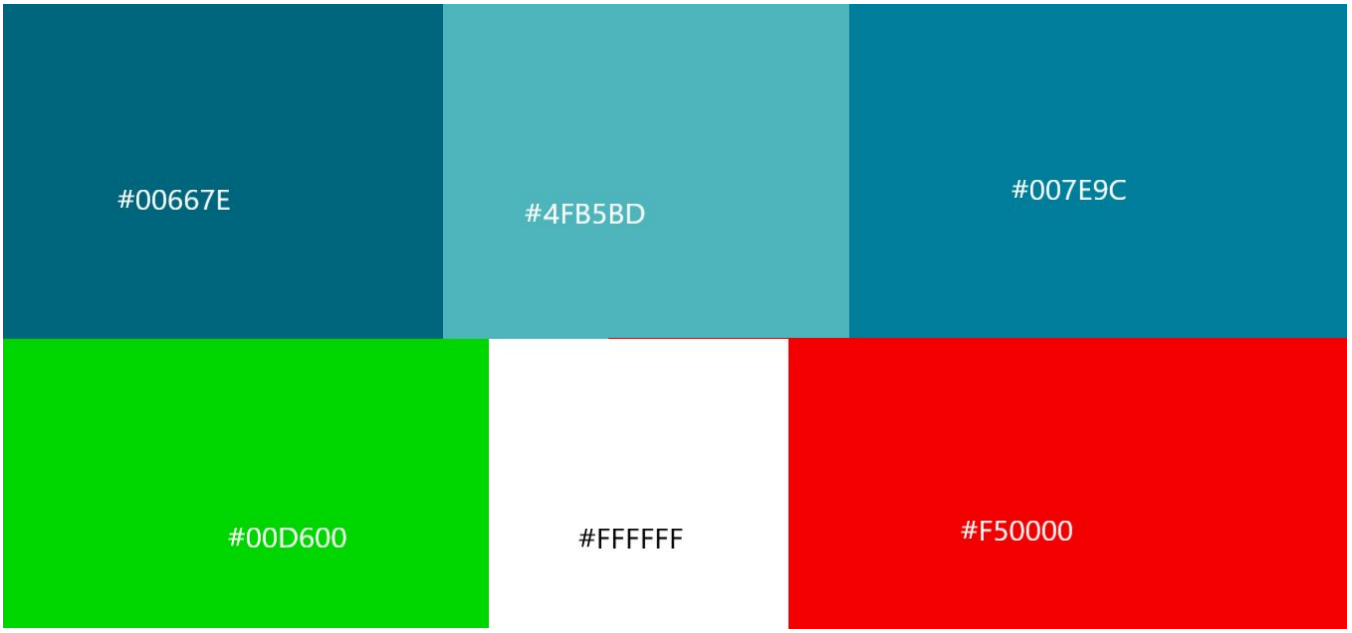
About section



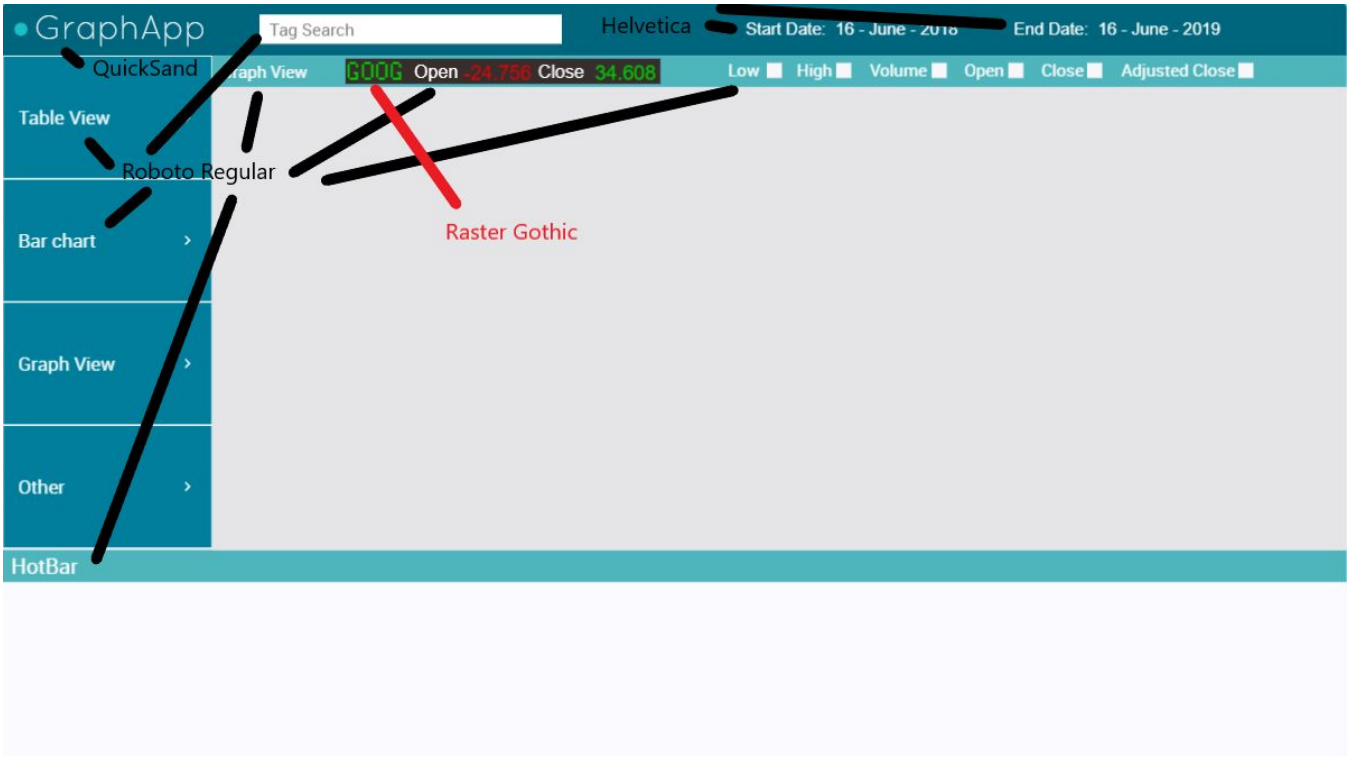
Original Concept art created for the project by Cian O'Gorman



Original pixel map created for the project, intended to show measurements and location of all elements to all team members.



Original colour palette used, created to ensure consistency between all developers when developing the project.



Original font plan used, created to ensure consistency between all developers when developing the project.