[redacted]

[redacted]

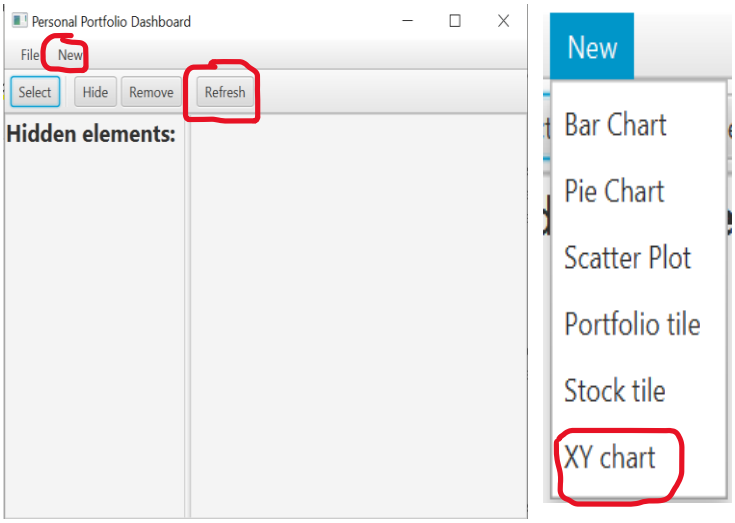
2. General description

[redacted] I chose to make a personal portfolio analysing program. You can save the stocks you own to a file and then analyze the performance your portfolio with interactive graphs. The graphs that I have implemented include time series, scatter plot, bar chart, pie chart and a tile where different statistics such as average and standard deviation can be visualised. You can also save dashboards for later use.

3. User interface

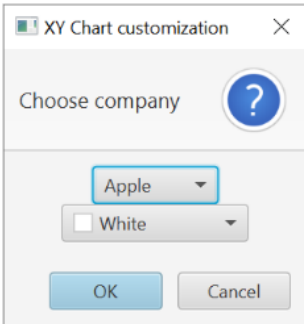
NB: before using the program claim a personal api key from here and add it to apiKey.txt in project root directory. The api free tier allows 25 calls per day. Pressing the refresh button uses 3 calls.

The program is run from the UI file. When the user opens the program they are presented with a user interface. If you try to add a chart, the chart will show minimal data. The user needs to refresh the data to get up to date data from the internet. This can be done by pressing the refresh button in the toolbar.



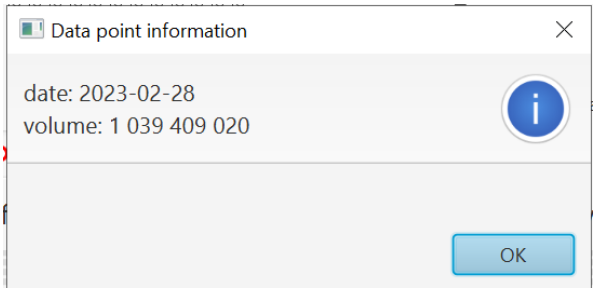
Now with sufficient data, the user can add a new elements to the dashboard. This is possible by clicking the “new” element in the menu bar. The menu item opens a drop box where you can select the element you want to use.

When the user chooses the element to show, the user is promted to give more information about the chart using an alert. The alerts differ slightly depending on whitch element you want to use. Typically the user is prompted to choose a company and a color. The picture is an alert for the xy chart.

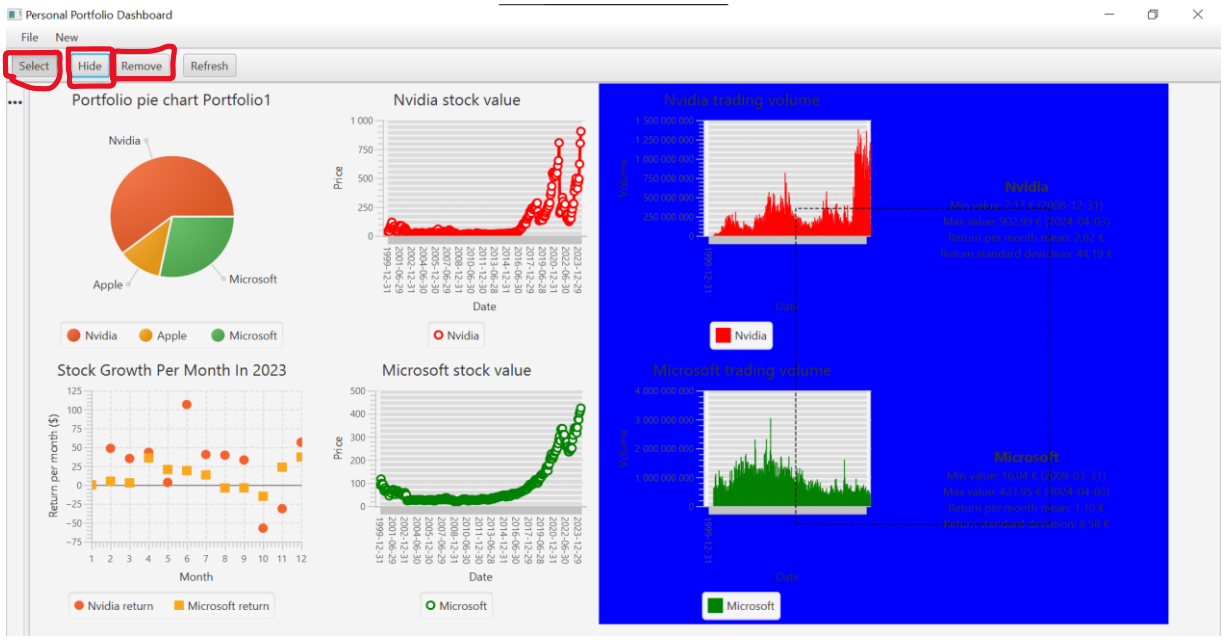


After adding a few charts the window might look like something similar to the picture on the right. You can hide the left split where the hidden components are located by dragging the separator all the way to the left. This way you can see all of the charts.

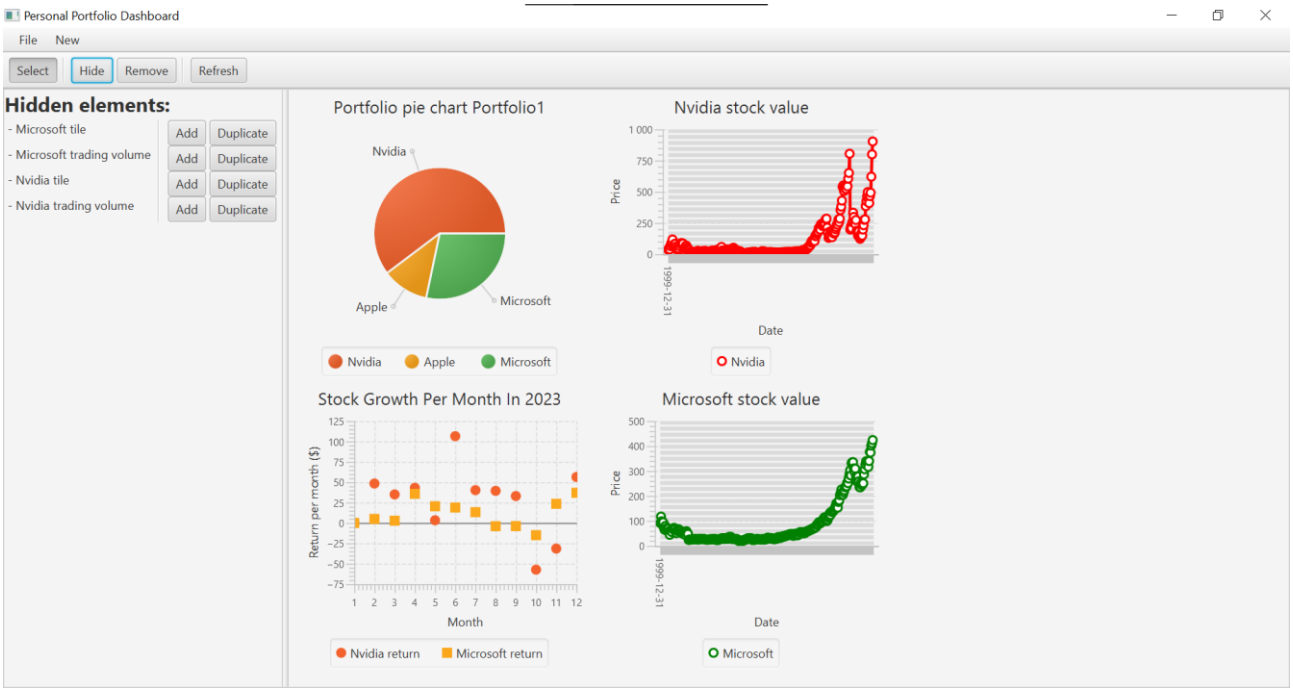
You can get additional information about datapoints of the bar chart and the xy chart when clicking on their data points. When using this feature, you have to make sure that the selection tool is inactive.



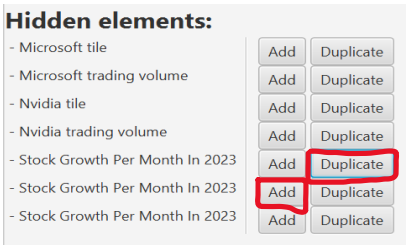
Let's say the user would like to hide the volume bar charts and tiles and replace these with other components. The user can use the rectangle selection tool whitch can be accessed by clicking the select toggle button in the toolbar. Selectin works by dragging the selection rectangle over the components you want to hide. When the wanted components are selected the user can press the "hide" button in the top left corner to hide the components. The user can also delete selected components by pressing the "Remove" button.



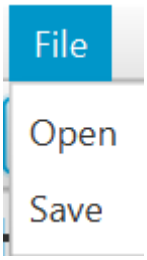
After hiding the components the components move into the left split. The size of the left split can be modified by dragging the bar splitting the two views.

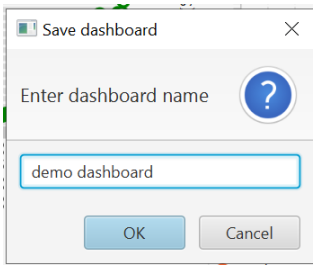


In this situation the user can do many things. The user can add more elements to the view now that space has cleared up. The user can also duplicate the hidden components if they wish. The user can also add the hidden components back to the pane by pressing the “add” button. The user also rearrange the layout of the charts by hiding and adding the charts in the desired order.

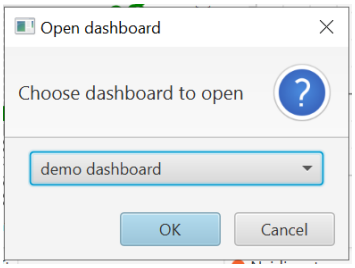


By opening the file drop bar (top left) the user can save their dashboard to a save file. When save is clicked the user is prompted to to give a dashboard name.



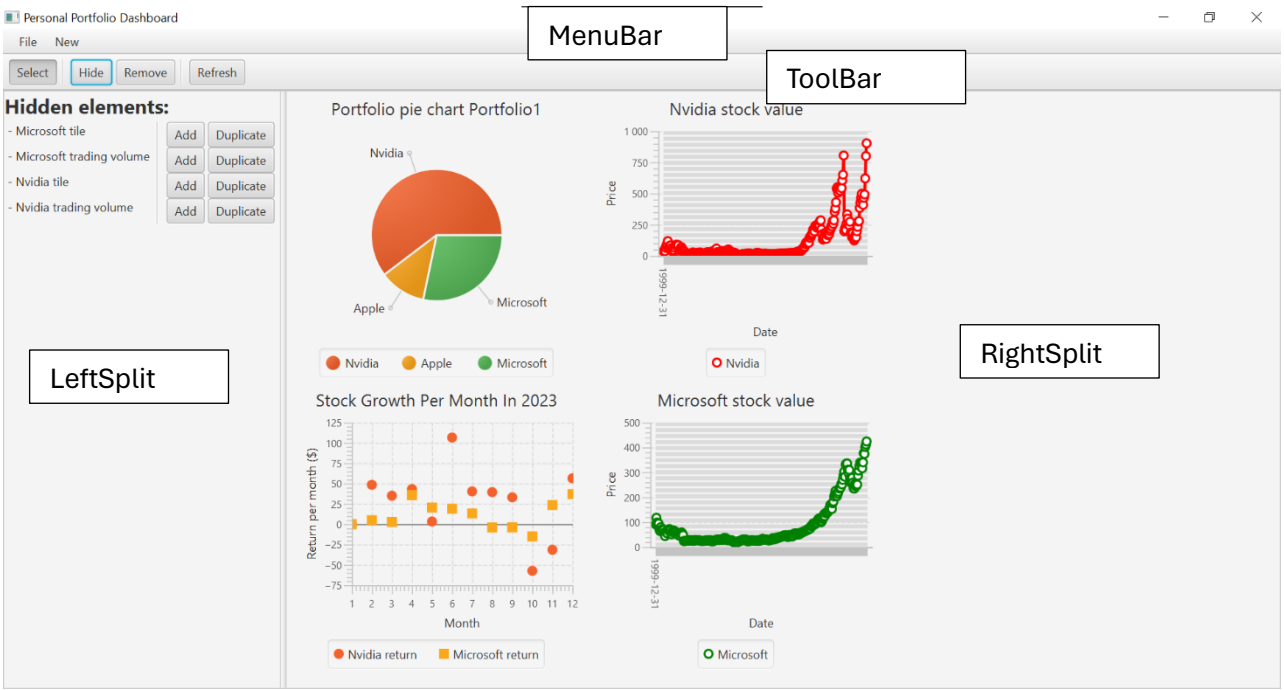


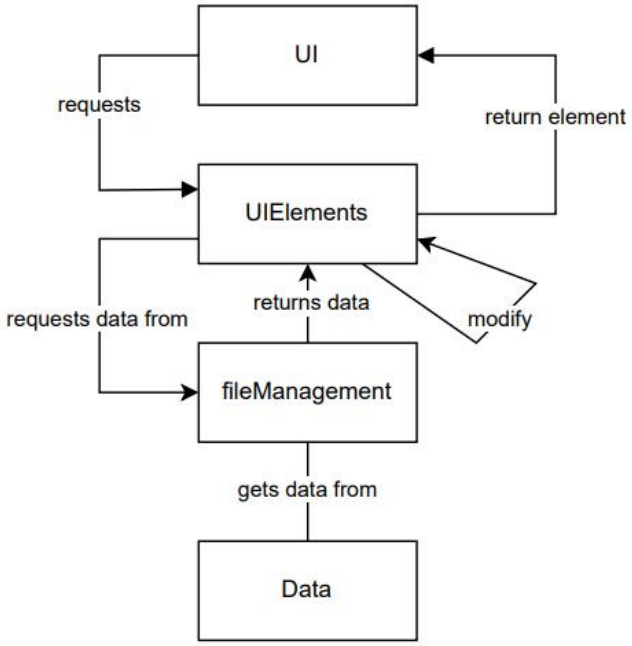
When the user wants to open the saved dashboard again the user can press the “open” button. The button opens an alert where the user can choose whitch of the saved dashboards they want to open. When the user chooses the dashboard they want, it will remove any previous content from the current window and render the saved dashboard.



4. Program structure

All of the functionality of the program is focused to the UIElements. For example if I want to delete a chart from the view, the method for deleting it is in the RightSplit object. This means that I use exclusively objects. Another example is that I use an accessing method getPieChart() to create new instances of a pie chart. The next pictures show the file structure, an abstact view of the class structure and the elements in the actual UI.





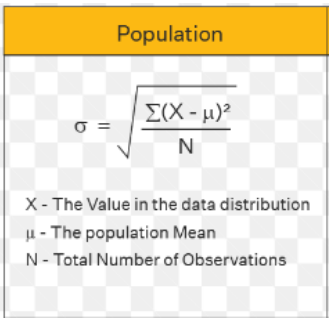

```
dashboard
├── UI.scala
├── UIElements
│   ├── DataAnalysisTools
│   │   ├── PortfolioPieChart.scala
│   │   ├── ReturnScatterPlot.scala
│   │   ├── Tile.scala
│   │   ├── TimeSeriesChart.scala
│   │   └── VolumeBarChart.scala
│   └── FunctionalityElements
│       ├── Alerts
│       │   ├── BarChartAlert.scala
│       │   ├── DashboardOpeningAlert.scala
│       │   ├── DashboardSavingAlert.scala
│       │   ├── PieChartAlert.scala
│       │   ├── PortfolioTileAlert.scala
│       │   ├── ScatterPlotAlert.scala
│       │   ├── StockTileAlert.scala
│       │   └── XYChartAlert.scala
│       ├── LeftSplit.scala
│       ├── MenuElement.scala
│       ├── RightSplit.scala
│       └── ToolBar.scala
├── data
│   ├── portfolios
│   │   └── portfolio1.json
│   ├── saveFiles
│   │   └── saveFile.json
│   └── timeSeries
│       ├── TIME_SERIES_MONTHLY_AAPL.json
│       ├── TIME_SERIES_MONTHLY_MSFT.json
│       └── TIME_SERIES_MONTHLY_NVDA.json
└── fileManagement
    ├── Api
    │   ├── ReadApiData.scala
    │   └── SaveApiData.scala
    └── SaveAndOpenDashboards
        ├── OpenDashboard.scala
        └── SaveDashboard.scala
```

I feel like the style that I chose for my program worked well. I thougth about object oriented programming with classes by extending the scalafx classes and writing my own additional methods for the classes but desided to stick with my original style. The object oriented style would work equally well with my program.

My implementation allows all of the actions of a certain compoent to happen in the same file. In practical this means that I have focused the functions that add, delete etc. to the UIElement that is responcible of the graph at that particular monent. For example the RightPane does has the method for removing graphs and the LeftSplit has a method for duplicating.

5. Algorithms

Algorithms were not in a significant role in my project. Calculatin the standard deviation was the hardest logic that I had to implement. The formula for that is on the right. I also implemented a sum, average and mean but the formulas for these are trivial so I'll not include them.



A generalized algorithm for creating charts is the following:

a. get data from file
b. modify data to correct form for chart

c. create chart with data

The algorithm for the selection tool is also worth mentioning. I calculated the rectangles starging point and dimensions using the point where the mouse was first pressed and the point where the mouse currently is. The rule for the starting point is if either of the current mouse coordinate x or y is smaller than the start, use the current x or y. The height and with are easy to calculate with a difference operation and an taking the absolute value.

6. Data structures

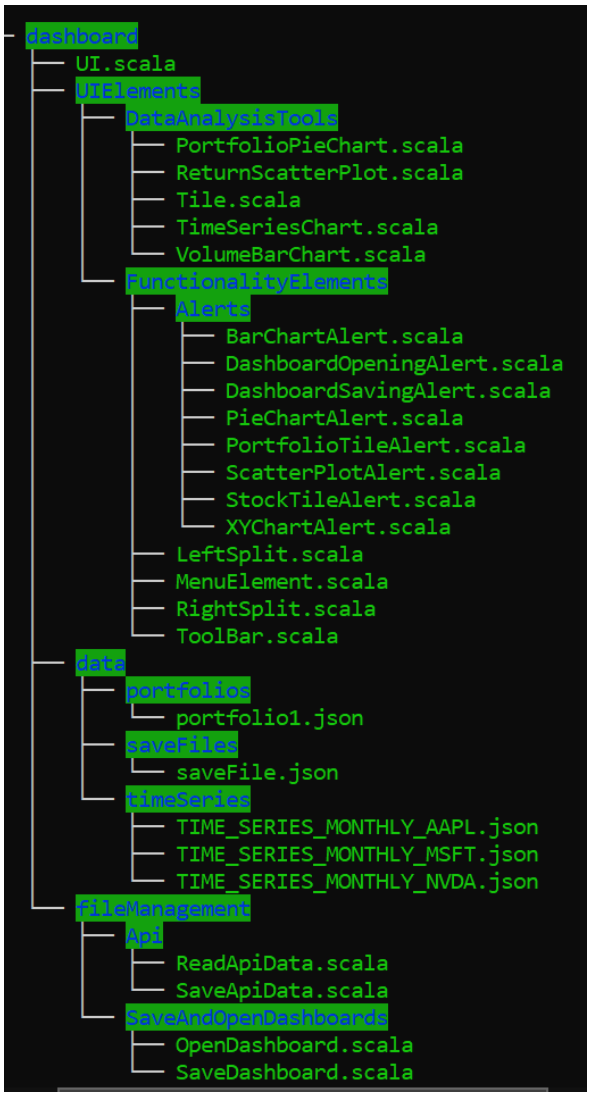
For handling JSON data I used maps. I used maps when handling the data from the financial api and for saving and opening dashboards. I needed a mutable data type to handle data that has key-value pairs so maps where an obvious choise.

I also used different types of arrays for listing values. I used arrays, buffers, observable buffers and tuples to name a few. I used buffers when data had to be easily modifiable. I used arrays when the collection did not need to be modified a lot and appending values was enough. Observable buffers where required when setting up the data for the charts.

7. Files and Internet access

All of my files are in JSON format. I used one file to save dashboars to, one for each company to save the time series from the API and one to save a portfolio. The file saveApiData.scala handels the api call. The saveApiData file only writes the data from the call to the file. I have all of the code for reading the api data in the readApiData.scala file.

If you wanted to modify the portfolio, you have to modify the portfolio file itself. Note that the portfolio only works with Apple, Nvidia and Microsoft. This is because th api call only gets data from these three companies.



8. Testing

I tested my program mainly by starting the program and visually confirming that the program worked correctly. A couple of times I had to write unit tests to identify bugs in my code. One time I was having a hard time figuring out a bug based on the compiler errors so I built unit tests to test the functionality bottom up and I figured out the bug by doing this.

For making shure that the data was given to the grapghs in correct form I simply printed the data to the console by using println() and visually confirmed that I had the data I wanted. In my original testing plan I planned to do more unit tests but when programming I felt like they would only hinder my progress.

9. Known bugs and missing features

When I set the background color of a time series or a bar chart to some color, the values of the x-axis disappear. I believe that this is a bug in the scalafx library because I am setting the background color with a method provided by the library itself.

The clicking functionality that shows more information about the data points only works when the rectangle selection tool is not activated. To be honest I am not sure why this happens. I have investigated the problem a lot but just bet more and more confused the more I dig into it. The feature starts working correctly again when you deselect the selection tool.

The files management functions are sensitive to the form the data is in the files. If the data is in an incorrect form the program will output "key not found" errors. The file saveFile.json requires at least an empty map. The files that store the time series require at least a stump of the full time series to function. The portfolio file can only have three stocks: Microsoft, Nvidia and Apple. The whole program works only with these three stocks.

I should write more tests. When the program starts growing, it will become harder and harder to visually verify that the program works correctly. It could also be beneficial write tests to make sure that the matematical calculations such as standard deviation and mean do not have any bugs with any values.

I have implemented all of the features required for the demaning level. I don't have any features missing.

10. 3 best sides and 3 weaknesses

3 strengths

1. Clear program structure. It was a top priority for me to keep my code organized. Adding new features was easier when all of the code was easily comprehensible and organized.
2. Minimalistic and intuitive user interface. The user interface has all of the nesessary features and all of the features can be found in reasonable places.
3. Problems are solved in separate parts of the code. This makes the code more easily scalable and modifyable and reduces the chance of future bugs.

3 weaknesses

1. The program only works with Microsoft, Nvidia and Apple. If in real production, the program should be able to function with all of the stocks in the market. This could be implemented by writing the company symbol (e.g. AAPL for Apple) to the portfolio file. The company symbol is used as the key for the api to return correct stock information.
2. When opening a saved dashboard the user is not presented with an option to save the current dashboard. When you open a new dashboard all of the graphs are lost that are currently in the window
3. The code doesn't use fancy programming techniques like traits and threads. Only elementary techniques are used.

11. Deviations from the plan, realized process and schedule

I sticked to my plan pretty well. I was ahead of my schedule all of the time and got the program done well before the deadline. The only difference between the plan and the implementation was that I had to implement a basic ui for the testing of the graphs relatively early in the project. In my original plan I wrote that I would implement the user interface last and do everything else before that. In hindsight this is completely understandable.

In my schedule I thought that I would use much less time implementing the user interface. I thought that the hard part would be parsing data into a usable form. The hard part in reality was figuring out how to use scalafx. I had to learn to read documentation and in general I spent a lot of time just figuring out the syntax of scalafx and correct methods to use. Of course the logical parts were also time consuming but I never got stuck when doing these tasks because I had a solid skillset in basic scala programming.

12. Final evaluation

It would be interesting to implement the project in a way where the functionality of the graphs would be focused in the graph classes. It would be intresting to see what the differences would be in code readibility and scalability. The way I implemented the project makes it easier to modify the UI components rather than addning new graphs. For example if I wanted to add a new graph to the program I would have to also modify all of its functionality (hiding, saving, selecting etc.) in different folders. With my program structure modifying the UI components is easier.

I believe that my program structure was heavily influenced by the order I implemented features. My intuition says that having the functionality of each graph in each class would make the program more scalable. The new program structure would have a trait such as Graph and then have methods such as hide(), show(), duplicate() and select(). I didn't have a clear class structure in mid when starting the program because I was extremely focused on the technical details of how to implement differernt features. If I could do the project again, I would move all of the functionality of the charts to the chart classes from the UIElement objects.

13. References

What books, websites or other material you have been using (excluding the online course textbook)?

- scalafx github page:
  - https://github.com/scalafx/scalafx/tree/master/scalafx-demos/src/main/scala/scalafx/scene/chart
- the official scalafx documentation
  - https://javadoc.io/doc/org.scalafx/scalafx_3/latest/index.html
- Alpha Vantage documentation for the api
  - https://www.alphavantage.co/documentation/#

- Oracle blog posts for javafx references
    - https://www.oracle.com/java/technologies/javase/javafx-overview.html
- Mark Lewis YouTube playlist of scalafx basics
    - https://www.youtube.com/playlist?list=PLLMXbkbDbVt9MIJ9DV4ps-_trOzWtphYO
- stack overflow different pages related to scalafx and javafx
    - https://stackoverflow.com/
- other random blog posts from all topics
- ChatGPT a couple of times when I havn't found sufficient material elsewhere