

# Stock Market Analyzer

Design Doc

## Table of Contents

Table Of Figures .....	2
<b>SCENARIO OUTLINE.....</b>	<b>5</b>
<b>LANGUAGES AND SOFTWARES .....</b>	<b>5</b>
<b>JAVA FX .....</b>	<b>6</b>
<b>SCENES AND CONNECTIONS .....</b>	<b>6</b>
<b>FXML FILES .....</b>	<b>7</b>
<b>WORLD FXML .....</b>	<b>7</b>
<b>DESIGNING WORLD .....</b>	<b>8</b>
<b>WORLD CONTROLLER .....</b>	<b>9</b>
<b>COMPANIES FXML .....</b>	<b>11</b>
<b>DESIGNING COMPANIES .....</b>	<b>12</b>
<b>COMPANIES CONTROLLER .....</b>	<b>14</b>
<b>TSLA FXML .....</b>	<b>20</b>
<b>APPLE FXML .....</b>	<b>21</b>
<b>GOOGLE FXML .....</b>	<b>22</b>
<b>FORD FXML .....</b>	<b>23</b>
<b>HP FXML .....</b>	<b>23</b>
<b>MICROSOFT FXML .....</b>	<b>24</b>
<b>TRADE FXML.....</b>	<b>25</b>
<b>DESIGNING TRADE .....</b>	<b>26</b>
<b>TRADE CONTROLLER.....</b>	<b>27</b>
<b>PORTFOLIO FXML.....</b>	<b>29</b>
<b>DESIGNING PORTFOLIO .....</b>	<b>30</b>
<b>PORTFOLIO CONTROLLER .....</b>	<b>30</b>
<b>CONTACT US FXML .....</b>	<b>34</b>
<b>ABOUT US FXML .....</b>	<b>35</b>
<b>FLOW CONTROL.....</b>	<b>35</b>
<b>COMPOSITE DATA TYPES.....</b>	<b>36</b>

<b>Method (Function) Overloading.....</b>	<b>37</b>
<b>SETTERS, GETTERS, DEFAULT AND PARAMETERIZED CONSTRUCTORS.....</b>	<b>37</b>
<b>METHOD OVERRIDING, CONSTRUCTORS IN SUBCLASSES.....</b>	<b>38</b>
<b>POLYMORPHISM .....</b>	<b>39</b>
.....	39
<b>this KEYWORD.....</b>	<b>39</b>
<b>STATIC CLASS MEMBERS .....</b>	<b>39</b>
<b>INTERFACES, ABSTRACT CLASSES, AND METHODS.....</b>	<b>40</b>
<b>ARRAYS AND FUNCTIONS .....</b>	<b>42</b>
<b>EXCEPTIONAL HANDLING AND DATA VALIDATION .....</b>	<b>42</b>
<b>OUTPUTS .....</b>	<b>44</b>
<b>ASSOCIATED PYTHON FILES.....</b>	<b>51</b>
<b>BUY.PY FILE .....</b>	<b>51</b>
<b>SELL.PY FILE .....</b>	<b>53</b>
<b>NEWS.PY FILE.....</b>	<b>55</b>
<b>RATES.PY: .....</b>	<b>56</b>
<b>FORECASTING.PY FILE .....</b>	<b>58</b>
<b>CONCLUSION .....</b>	<b>63</b>

## Table Of Figures

Figure 1: World Window.....	7
Figure 2: WORLD FXML.....	8
Figure 3: Buttons and Labels.....	9
Figure 4: Navigation to different Windows .....	9
Figure 5: loadPage Method .....	10
Figure 6: newsSetter Method .....	10
Figure 7: Abstract initialize Method .....	10
Figure 8: Adding Elements to the Combo Box.....	11

Figure 9: WORLD Page with Drop Down.....	11
Figure 10: TESLA Page.....	12
Figure 11: FORD MOTORS Page.....	13
Figure 12: Setting the Image View.....	13
Figure 13: Setting the Text Area .....	14
Figure 14: Import statements in all of the Companies Controllers .....	15
Figure 15: Labels in Companies Controllers .....	15
Figure 16: newsSetter method .....	16
Figure 17: Setting value using newsSetter Method.....	16
Figure 18: Creating an object of CSVReader.....	17
Figure 19: Setting the value of labels using object of CSV Reader .....	17
Figure 20: Setting date using object of DATE Class .....	17
Figure 21: DATE Class.....	18
Figure 22: Data Members of CSVReader .....	18
Figure 23: Constructor of CSVReader .....	18
Figure 24: readFromLast method in CSVReader .....	19
Figure 25: Second snippet of the readFromLast method.....	20
Figure 26: Third snippet of the readFromLast method .....	20
Figure 27: TESLA FXML.....	21
Figure 28: APPLE FXML .....	22
Figure 29: GOOGLE FXML .....	22
Figure 30: FORD FXML .....	23
Figure 31: HP FXML.....	24
Figure 32: MICROSOFT FXML.....	25
Figure 33: TRADE FXML.....	26
Figure 34: Code snippet of TRADE FXML.....	26
Figure 35: Import statements used in TRADE controller.....	27
Figure 36: Declaration of Combo Boxes in TRADE.....	27
Figure 37: controlTrade method in TRADE Controller.....	28
Figure 38: else block of controlTrade method .....	28
Figure 39: initialize method in TRADE .....	28
Figure 40: Portfolio Page .....	29
Figure 41: Profit Calculator .....	30
Figure 42: Table View Code in Portfolio FXML .....	30
Figure 43: import statements in Portfolio Controller.....	31
Figure 44: Data Members in Portfolio .....	31
Figure 45: Table Columns and Text Field in Portfolio Controller .....	32
Figure 46: Text Field, button, and Table Column in Portfolio Controller .....	33
Figure 47: checkField method in Portfolio .....	33
Figure 48: getPortfolio method .....	34
Figure 49: Contact US Page.....	34
Figure 50: ABOUT US Page.....	35
Figure 51: IF ELSE example .....	36
Figure 52: Composite Data Types Example .....	36
Figure 53: String type Arrays .....	37
Figure 54: setters and getters example .....	38
Figure 55: Default and Parametrized Constructor .....	38
Figure 56: static class members.....	39

Figure 57: usage of static class members .....	40
Figure 58: abstract class Trade .....	40
Figure 59: data members of abstract class Trade .....	40
Figure 60: Sub Class of Trade.....	41
Figure 61: Sub Class of Trade.....	41
Figure 62: Arrays .....	42
Figure 63: try block .....	43
Figure 64: catch block.....	43
Figure 65: check Fields method .....	44
Figure 66: World Page .....	44
Figure 67: Google Page .....	45
Figure 68: Apple Page .....	46
Figure 69: Ford Page .....	47
Figure 70: HP Page .....	48
Figure 71: Microsoft Page.....	49
Figure 72: Tesla Page .....	49
Figure 73: Trade Page .....	50
Figure 74: dealIn button only works if all the combo boxes are selected.....	50
Figure 75: Alpaca Trading Website .....	51
Figure 76: Configuring Alpaca API for buying stocks .....	52
Figure 77: Reading contents of Buy.txt file .....	52
Figure 78: Buy.txt file .....	52
Figure 79: Placing buy order .....	53
Figure 80: Writing buy order in Orders.txt .....	53
Figure 81: Configuring Alpaca API for selling stocks.....	53
Figure 82: Reading contents of Sell.txt file .....	54
Figure 83: Sell.txt file .....	54
Figure 84: Placing sell order.....	54
Figure 85: Writing details of sell order in Orders.txt.....	54
Figure 86: News.txt file .....	55
Figure 87: Importing Google news API .....	55
Figure 88: Retrieving news from API and writing in news.txt .....	56
Figure 89: Pandas and stocks list .....	56
Figure 90: Slope function .....	56
Figure 91: Calculating rate of every stock and writing in rates.txt.....	57
Figure 92: Rates.txt .....	57
Figure 93: Rates reference .....	58
Figure 94: Importing required libraries .....	58
Figure 95: Plotting function .....	59
Figure 96: Retrieving data from Tingo API.....	59
Figure 97: Retrieved data format .....	60
Figure 98: Forecasting stock values and plotting .....	60
Figure 99: Facebook Prophet result.....	61
Figure 100: LSTM Architecture .....	62
Figure 101: LSTM results.....	62

## **SCENARIO OUTLINE**

This project has been made keeping in view the major problem faced by the investors in their mundane and hectic life. Often investors are forced to sell their stocks in loss and had to make up for their losses thereby risking their investment. This program shows the trends for six of the most major stocks in the market which the user can check before buying up the stocks and can save themselves from big losses. You can see the trends for MICROSOFT, TESLA, GOOGLE, HP, FORD, and APPLE. These are some of the big companies that everybody knows about and can understand the stocks buying/selling concept easily. They don't have to look up which stock to buy if they want a certain amount of profit in a specific period of time for, we also offer a profit calculator in our program. User can enter the profit and number of days and the stock will be referred to him. Hence, the main purpose of this program is to ease up the process of buying stock and give the best possible predictions to the investor.

## **PROBLEM STATEMENT**

The stock trading community was faced with the problem of buying and selling profitable stocks. Most of the traders were unable to determine the trends of the stock and ultimately, they end up in investing in the depreciating stocks. Moreover, entry-level investors were faced with the problem of investing in right stocks or stocks which were more suitable depending upon their requirements.

## **LANGUAGES AND SOFTWARES**

We have mainly used two programming languages in our project. Project is based upon PYTHON and JAVA programming languages but mainly the concepts that we have learned in our semester are applied in JAVA PROGRAMMING LANGUAGE. Moreover, we have also used GUI in our project for better user experience and easy to use interface has been provided in our program. For GUI, we have used JAVAFX software which is a library of java

and we have also used SCENEBuilder. The details of the project have been explained below.

## **JAVA FX**

JAVA FX software has been used to design the Graphical User Interface for TESSERACT. Tesseract has been designed to forecast the trends for different stocks so the buyer or seller can take an idea of the market before making an investment. JAVA FX, although, is one of the best GUI libraries in JAVA if not the best out there. The details of the GUI have been given below.

## **SCENES AND CONNECTIONS**

Most of you can understand the meaning of the word “scene” after hearing or listening it. But in case you don’t understand it, it is a simple window on which different functions have been implemented and it basically looks like a stage where different actors are doing their acts and user can relate to the story more easily through their movements and gestures. Same is the case with scene. It helps the user interact with the program more easily and it also makes the program looks good. Different scenes have been designed for this project, the name of which are given below.

- WORLD
- TRADE
- PORTFOLIO
- ABOUT US
- CONTACT US
- TSLA
- HPQ
- GOOG
- MSFT
- AAPL
- F

These windows have their FXML files in which you design the scene according to the project requirements and each window also have their own controllers in which you can implement the functions which are to be performed by every window individually. All the windows are then connected through the main controller and can be changed from one window to the other window upon clicking buttons.

## FXML FILES

FXML is an XML file used for designing different windows for the project. There are 11 FXML files in total and all of them are explained below. Following are the details of each FXML files. The connections and the functions for different FXML files have been explained below.

## WORLD FXML

WORLD FXML is the most important window in the program as it is the first window that appears on the screen after when you run the program. WORLD window shows you graphs trends for three most trending stocks in the market. The buyer/seller can look at these graphs and decide what to do based on these trends given by the graphs. Snippet for world window is given below.

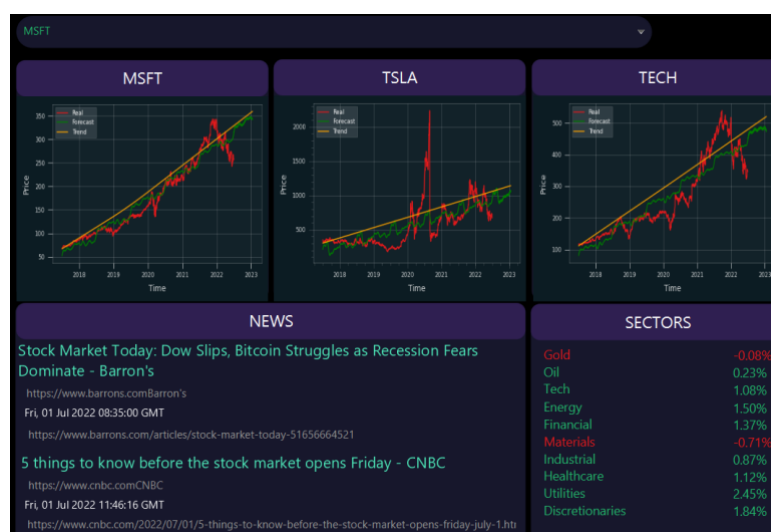


Figure 1: World Window



Figure 1 shows the World window along with its implementations and current news regarding the stocks market ups and downs.

World Window also provides you with the latest world news regarding the stock market. User can read the complete article by copying the given link and pasting it in the browser search bar. These articles can be helpful if you want to keep yourself updated with the current world scenarios and can also help you in trading stocks. Sectors are also provided in this window. These can include Gold, Tech, machines, Healthcare etc. as it gives you increasing or decreasing trends of these sectors and whether it is safe for you to invest in these sectors.

## DESIGNING WORLD

For the WORLD FXML window to look like this different colour schemes were looked upon by the team members and carefully decided which colour scheme will look better. We also used different tools provided by the JAVAFX library such as COMBO BOXES so that we can select other windows from them such as trends for different stocks can be accessed through these combo box selections. Moreover, different Labels as well Text Areas (Tools) have also been implemented in this window to make the experience for the user easy to understand and more efficient. The major part of the code for FXML file is given below.

```
<Button fx:id="worldButton" layoutX="-6.0" layoutY="26.0" mnemonicParsing="false" onMouseClicked="#WORLD" prefHeight="32.0" style="background-color: #f0f0f0; border: 1px solid #ccc; border-radius: 5px; text-align: center; width: 100px;">
    <font>
        <Font name="SansSerif" size="17.0" />
    </font>
    <cursor>
        <Cursor fx:constant="HAND" />
    </cursor>
</Button>
<ImageView fitHeight="32.0" fitWidth="35.0" layoutX="14.0" layoutY="35.0" pickOnBounds="true" preserveRatio="true">
    <image>
        <Image url="@world.png" />
    </image>
</ImageView>
```

Figure 2: WORLD FXML

Figure 2 shows the piece of code for WORLD FXML file in which the colour scheme and different trends graph are being set by the programmer. If you want to analyse the complete code than you can see it in the WORLD.fxml file.

## WORLD CONTROLLER

WORLD controller basically shows how functionality is implemented in these windows. What classes and methods are being used in these windows. World controller snippet is shown below.

```
public class WorldController implements Initializable {  
  
    ObservableList<String> stockList = FXCollections.observableArrayList("ISLA", "AAPL", "MSFT", "F", "HPQ", "GOOG");  
  
    @FXML  
    private Button aboutUsButton;  
  
    @FXML  
    private AnchorPane anchorPane;  
  
    @FXML  
    private BorderPane borderPaneMain;  
  
    @FXML  
    private Button contactUsButton;  
}
```

*Figure 3: Buttons and Labels*

Figure 3 shows how each button, Labels are first made variables of their respective classes using the prefix @FXML with each private variable. This connects the labels and the buttons as well as combo boxes and other tools to the controller and helps to implement the functionality.

```
@FXML  
void ABOUTUS(MouseEvent event) {  
  
    loadPage("ABOUTUS");  
}  
  
@FXML  
void CONTACTUS(MouseEvent event) {  
  
    loadPage("CONTACTUS");  
}
```

*Figure 4: Navigation to different Windows*

Figure 4 shows how we can navigate through different windows by clicking on relevant buttons. Different methods are being used in which a parameter of MouseEvent is passed since such methods are called when mouse button is clicked. To interchange between Windows, loadPage method is called inside each mouse event method.

```
private void loadPage(String page) {
    Parent root = null;

    try {
        root = FXMLLoader.load(Objects.requireNonNull(getClass().getResource("page + ".fxml"))));
    } catch (IOException e) {
        e.printStackTrace();
    }

    borderPaneMain.setCenter(root);
}
```

Figure 5: loadPage Method

Figure 5 shows the loadPage method. As the name suggests, this method is used to load different pages when the mouse button is pressed and helps to navigate between different windows.

```
private String newsSetter(int n) {
    String line = null;
    try (BufferedReader br = new BufferedReader(new FileReader("C:\\Users\\HPP\\P
        for (int i = 0; i < n; i++)
            br.readLine();
        line = br.readLine();
    } catch (IOException e) {
        System.out.println(e);
    }
    return line;
}
```

Figure 6: newsSetter Method

Figure 6 shows how daily news about stock is set on the WORLD page dynamically and will change daily. For this purpose, we have used news setter method which is a buffered line reader used to read a specific line from a file.

```
@Override
public void initialize(URL url, ResourceBundle resourceBundle) {
```

Figure 7: Abstract initialize Method

Figure 7 shows the abstract method initialize of the interface initializable which is always executed when the WORLD window opens.

```
stockComboBox.setItems(stockList);
```

Figure 8: Adding Elements to the Combo Box

This statement is used to set the options in the combo box.

## COMPANIES FXML

So, we listed at maximum of six companies whose stock value we were going to predict.

These companies included,

- TESLA
- MICROSOFT
- GOOGLE
- FORD
- APPLE
- HEWLETT PACKARD

So, the first thing is that how do you get access to these windows. The answer is simple; when you open the WORLD page you can see a combo box named “Select a ticker” from where you can select any of the company whose stocks you are eager in. This window looks like this:

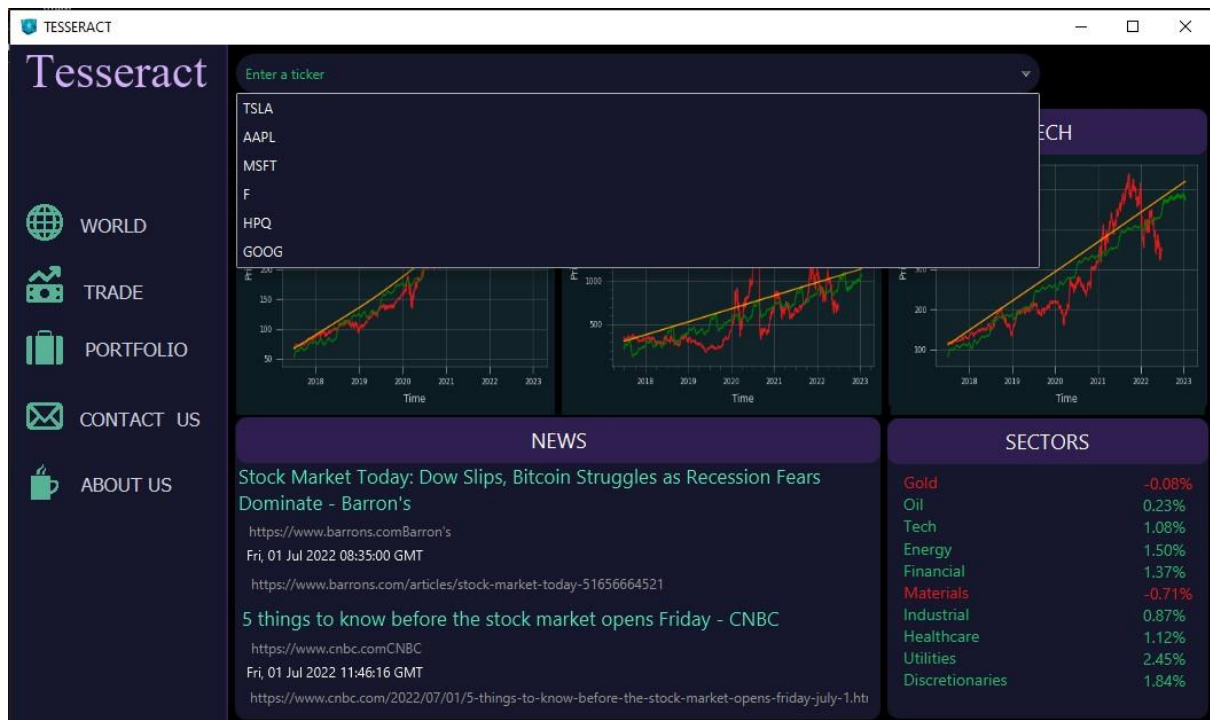


Figure 9: WORLD Page with Drop Down

Then let's say you clicked on TSLA, then a following window will pop up on your screen:



Figure 10: TESLA Page

## DESIGNING COMPANIES

This was also the most difficult task in our project that took a lot of our efforts. So, at the end the idea that we all members agreed upon was to have a **MENU** bar at the left side, then on the upper portion of the window we will show user the graph of that stock and on its right side the information of the window like its current price, opening price, high price, low price, P/E, Diversion, and the market value of that company.

Now excluding the MENU bar, the remaining window looks like:



Figure 11: FORD MOTORS Page

In this figure, I have shown you the page of FORD MOTOR COMPANY. So, what is happening here. Let's have a deeper conversation. So, on the top left of the window you can see the chart which is just a label and below is the actual graph where you can see the real, forecast and the trend of that stock (in this case of FORD). So, this basically works in the way that when we get our work done after training our model, we get a PNG file as a result, so we just access that file and display it here.

```
<ImageView fx:id="pic" fitHeight="214.0" fitWidth="348.0" layoutX="7.0" layoutY="50.0"
    <image>
        <Image url="@F.png" />
    </image>
</ImageView>
```

Figure 12: Setting the Image View

In the above figure, we have done just that. In the FXML file of ford we have made an image view and get F.png file and displayed it on the window.

Now, let's move on the next part which is on the right side of the chart. It's the place where you can get all the information about that stock. So, we declared them as labels and changed the value of them after reading their values from the CSV file. The explanation of this is provided in the "Controller Section".

Next, below the chart and the information section is a text area where you will find a little about each of these companies. In the FXML file, it looks something like this:

```
<TextArea editable="false" layoutX="6.0" layoutY="272.0" prefHeight="72.0" prefWidth="842.0" scrollTop="0" style="-fx-control-inner-background: rgb(23,23,44); -fx-border-color: black;" text="Ford Motor Company is a global company based in Dearborn, Michigan, that is committed to helping build a better world, where every person is free to move and pursue their dreams. The company's Ford+ plan for growth and value creation combines existing strengths, new capabilities and always-on relationships with customers to enrich experiences for and deepen the loyalty of those customers. Ford develops and delivers innovative, must-have Ford trucks, sport utility vehicles, commercial vans and cars and Lincoln luxury vehicles, as well as connected services. Additionally, Ford is establishing leadership positions in mobility solutions, including self-driving technology, and provides financial services through Ford Motor Credit Company. Ford employs about 182,000 people worldwide." wrapText="true">
<font>
  <Font size="14.0" />
</font></TextArea>
```

Figure 13: Setting the Text Area

Then, below this text area there is a whole section dedicated to the news of that particular stock. The news is extracted from the web using an API (Application Programming Interface). And this work is done in the python, that is, extracting data from the web using API. But once the data is retrieved and saved in a text file, it is then displayed here using labels, text fields and text areas.

## COMPANIES CONTROLLER

So, instead of explaining all the controller files. I would just take one controller file and explain it all here, else it would make it so much redundant. So, to implement this page or all the pages the basic import packages that we used were as follows:

```

import javafx.fxml.FXML;
import javafx.fxml.Initializable;
import javafx.scene.control.Label;
import javafx.scene.control.TextField;
import javafx.scene.text.Text;

import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.net.URL;
import java.util.ResourceBundle;

```

Figure 14: Import statements in all of the Companies Controllers

Next, we declared some of the text fields, labels, and text areas that we used in the whole of the page. And these were as follows:

```

2 usages
@FXML
private TextField f44;

2 usages
@FXML
private Label high;

2 usages
@FXML
private Label low;

2 usages
@FXML
private Label open;

2 usages
@FXML
private Label current_price;

2 usages
@FXML
private Label date;

```

Figure 15: Labels in Companies Controllers



In this figure, we not only used these but also more of the text fields which are not showed here to make sure that the report you are reading is not redundant. For reading the news after it has been stored in the text file, we made a method **newsSetter** and its snippet is shown below:

```
private String newsSetter(int n) {
    String line = null;
    try (BufferedReader br = new BufferedReader(new FileReader( fileName: "C:\\Users\\HPP\\ProjectCrypto\\src" +
        "\\main\\resources\\com\\project\\projectcrypto\\NEWS.txt"))) {
        for (int i = 0; i < n; i++)
            br.readLine();
        line = br.readLine();
    } catch (IOException e) {
        System.out.println(e);
    }
    return line;
}
```

Figure 16: newsSetter method

This method reads a line of the text file and then returns it. And then in the initialize method we set the value of the label to that string which this method returns. The initialize method is shown below:

```
@Override
public void initialize(URL url, ResourceBundle resourceBundle) {

    f37.setText(newsSetter( n: 34));
    f38.setText(newsSetter( n: 35));
    f39.setText(newsSetter( n: 36));
    f40.setText(newsSetter( n: 37));

    f41.setText(newsSetter( n: 39));
    f42.setText(newsSetter( n: 40));
    f43.setText(newsSetter( n: 41));
    f44.setText(newsSetter( n: 42));
}
```

Figure 17: Setting value using newsSetter Method

Then, to display the value of the stock like the current price, opening price, and closing price et cetera, we made a class and named it **CSVReader** that would get us the value of the label we wanted. The object of the CSVReader is created in the following way:

```
CSVReader csv = new CSVReader( filename: "F.csv");
```

*Figure 18: Creating an object of CSVReader*

In this figure, we have passed the value of the CSV we want our program to read. And since we are in the controller of the form, we passed the constructor "F.csv" and it will change depending upon the controller in which we are.

Next, we set up the value of the label as follows:

```
high.setText("$" + csv.high);  
low.setText("$" + csv.low);  
open.setText("$" + csv.open);  
current_price.setText("$" + csv.current_price);
```

*Figure 19: Setting the value of labels using object of CSV Reader*

So, this is basically how our page looks and works. And yes, one of the most important things is to show user the current date and time and this is done using the following code:

```
DATE d1 = new DATE();  
date.setText(d1.datecalculator());
```

*Figure 20: Setting date using object of DATE Class*

## **CLASS DATE**

And the **DATE** class looks something like this:

```

import java.time.format.DateTimeFormatter;
import java.time.LocalDateTime;

12 usages  SYED HASAN ABBAS
public class DATE {

    6 usages  SYED HASAN ABBAS
    public String datecalculator () {
        DateTimeFormatter dtf = DateTimeFormatter.ofPattern("yyyy/MM/dd HH:mm:ss");
        LocalDateTime now = LocalDateTime.now();
        return (dtf.format(now));
    }
}

```

Figure 21: DATE Class

## CLASS CSVREADER

Now, let me also show you how does the class of CSVReader looks like.

```

public class CSVReader {
    public String open;
    public String high;
    public String low;
    public String current_price;
    1 usage
    public String company_name;
}

```

Figure 22: Data Members of CSVReader

This figure shows the data members of the CSVReader and all of them are declared as public and strings.

Now, let's move to the constructor of this class.

```

CSVReader(String filename) {
    File file = new File( pathname: "C:\\Users\\HPP\\ProjectCrypto\\src\\main\\java" +
        "\\com\\project\\projectcrypto\\" + filename);
    readFromLast(file, lines: 2);
}

```

Figure 23: Constructor of CSVReader

The figure shows the constructor of this class where it only receives one parameter only and which is the file name and then calls a method to read that file from the last.

One of the methods in this class is `readFromLast` and it looks something like this.

```
public void readFromLast(File file, int lines){
    int readLines = 0;
    StringBuilder sb = new StringBuilder();
    RandomAccessFile randomAccessFile = null;
    try {
        randomAccessFile = new RandomAccessFile(file, "r");
        long fileLength = file.length() - 1;
        // Set the pointer at the last of the file
        randomAccessFile.seek(fileLength);
        for(long pointer = fileLength; pointer >= 0; pointer--){
            randomAccessFile.seek(pointer);
            char c;
            // read from the last one char at the time
            c = (char)randomAccessFile.read();
            // break when end of the line
            if(c == '\n'){
                readLines++;
                if(readLines == lines)
                    break;
            }
        }
    }
}
```

Figure 24: `readFromLast` method in `CSVReader`

In the figure above you can see one of the snippets of the method `readFromLast`. What it does is that it basically reads the file from the last using the `RandomAccessFile`.

```

        sb.append(c);
    }
    sb.reverse();
    String nb = new String(sb);
    String[] info = nb.split(regex: ",");
    setCompany_name(info[0]);
    setCurrent_price(info[2]);
    setHigh(info[3]);
    setLow(info[4]);
    setOpen(info[5]);
} catch (FileNotFoundException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
}
catch (IOException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
}

```

Figure 25: Second snippet of the readFromLast method

And we also know that in CSV the values are separated by a comma so to store different values we used a delimiter and set it equal to a comma.

```

    }finally{
        if(randomAccessFile != null){
            try {
                randomAccessFile.close();
            } catch (IOException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
        }
    }
}

```

Figure 26: Third snippet of the readFromLast method

This snippet runs only if there exists an error in the RandomAccessFile.

## TSLA FXML

The FXML file for the Tesla Incorporation looks something like this. You get to know the current price of currently one of the biggest companies in the world. And the latest news

related to it. Tesla, Inc. is an American multinational automotive and clean energy company headquartered in Austin, Texas. Tesla designs and manufactures electric vehicles, battery energy storage from home to grid-scale, solar panels and solar roof tiles, and related products and services.



Figure 27: TESLA FXML

## APPLE FXML

The company Apple has been around since. Founded by one of the technology aesthetic person Steve Jobs, the company plays a major role in this technological advancement of humanity. From Macintosh to one of the slimmest PC the Apple has got it all. So, one would want to know the trend of its stock. The FXML file for the APPLE Incorporation looks something like this.



Figure 28: APPLE FXML

## GOOGLE FXML

Google LLC is an American multinational technology company that focuses on artificial intelligence, search engine technology, online advertising, cloud computing, computer software, quantum computing, e-commerce, and consumer electronics. The FXML file for the GOOGLE is shown below:



Figure 29: GOOGLE FXML

## FORD FXML

Ford Motor Company is an American multinational automobile manufacturer headquartered in Dearborn, Michigan, United States. It was founded by Henry Ford and incorporated on June 16, 1903. The company sells automobiles and commercial vehicles under the Ford brand, and luxury cars under its Lincoln luxury brand. And its FXML file is as follows:



Figure 30: FORD FXML

## HP FXML

The Hewlett-Packard Company, commonly shortened to Hewlett-Packard or HP, was an American multinational information technology company headquartered in Palo Alto, California.





Figure 31: HP FXML

## MICROSOFT FXML

Microsoft Corporation, commonly known as Microsoft, is an American multinational technology corporation which produces computer software, consumer electronics, personal computers, and related services headquartered at the Microsoft Redmond campus located in Redmond, Washington, United States. The FXML file is shown below:



Figure 32: MICROSOFT FXML

## TRADE FXML

TRADE FXML is another important window that is used to trade different stocks based on the selection of the customer. This is a minimalistic window so that customer would not be overwhelmed and can make trade easily. The user needs to select options from four fields and then click the DealIn button. TRADE WINDOW is shown below.

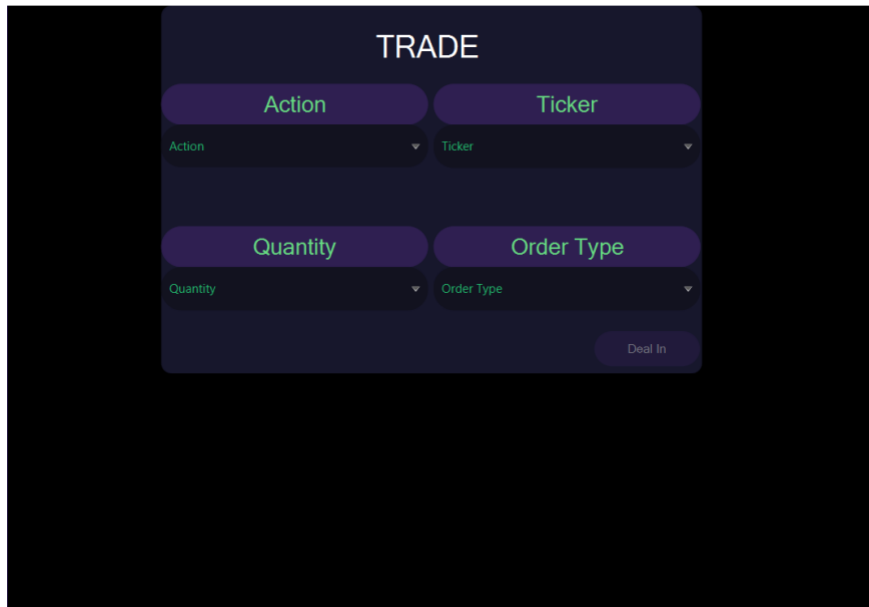


Figure 33: TRADE FXML

Figure 33 shows the TRADE FXML file which is used to buy and sell stocks in the market.

## DESIGNING TRADE

One of the most important things that we kept in mind while designing this window is to keep it minimalistic and easy to use for the customer. Just select four options from the given four combo boxes and then click on the Deal In button. You can see your order details either in the PORTFOLIO window or on the ALPACA website on the OVERVIEW PAPER TRADE page. To make the experience user friendly four different combo boxes were given, and the user just needs to select the options for the combo boxes and then click on the deal In button. Some of the code Snippets of the FXML file is shown below which is used to give different colour schemes to the layout and to give specific FX ID to different labels and combo boxes. Afterwards, relative combo boxes are accessed with reference to their names.

```
</Label>
</children>
</AnchorPane>
<ComboBox fx:id="action" layoutY="117.0" prefHeight="41.0" prefWidth="262.0" promptText="Action" style=
<ComboBox fx:id="ticker" layoutX="268.0" layoutY="117.0" prefHeight="41.0" prefWidth="262.0" promptText
<AnchorPane layoutY="217.0" prefHeight="40.0" prefWidth="262.0" style="-fx-background-color: rgb(47,31,
  <children>
    <Label layoutX="90.0" layoutY="7.0" text="Quantity" textFill="#68db83">
```

Figure 34: Code snippet of TRADE FXML

Figure 34 shows the code for designing the FXML file. Though, it is not the complete code, but you can see the whole code on the files attached with this word file.

## TRADE CONTROLLER

TRADE controller is used to implement functions in the GUI so that the user can buy/sell stock easily without interacting with the command prompt. Some of the important code snippets for TRADE CONTROLLER is shown below.

```
import javafx.collections.FXCollections;
import javafx.fxml.FXML;
import javafx.fxml.Initializable;
import javafx.scene.control.Button;
import javafx.scene.control.ComboBox;
import javafx.scene.input.MouseEvent;
import java.io.IOException;
import java.net.URL;
import java.util.Formatter;
import java.util.ResourceBundle;
```

Figure 35: Import statements used in TRADE controller

Figure 35 shows some of the important imports of TRADE CONTROLLER files. These imports are important part of each controller file there is in this project.

```
public class TRADEController implements Initializable{

    @FXML
    private Button dealInButton;

    @FXML
    private ComboBox<String> action;

    @FXML
    private ComboBox<String> ticker;
    @FXML
    private ComboBox<String> quantity;

    @FXML
    private ComboBox<String> order_type;
```

Figure 36: Declaration of Combo Boxes in TRADE

Figure 36 shows the declaration of variables for the four combo boxes used in this window. These combo boxes were named as action, ticker, quantity, order\_type.

```

@FXML
void controlTrade(MouseEvent event) throws IOException, InterruptedException {

    Formatter output;

    String Action = action.getValue();
    String Ticker = ticker.getValue();
    String Quantity = quantity.getValue();
    String Type = order_type.getValue();

    int q = Integer.parseInt(Quantity);

    if (Action.equals("buy")){
        System.out.println("buy done");
        output = new Formatter( fileName: "buy.txt");
        output.format("%s\n%s\n%s\n%s", Ticker, Quantity, Type, Action);
        output.close();
        buy b1 = new buy(Ticker,q,Type,Action);
        b1.Buy();
    }
}

```

Figure 37: controlTrade method in TRADE Controller

Figure 37 shows the method controlTrade which is used to place the order when the deal in button is pressed. If user selects the buy option, then based on these options buy method will be called else sell method will be called. The details of buy and sell method are shown below.

```

else{
    output = new Formatter( fileName: "sell.txt");
    output.format("%s\n%s\n%s\n%s", Ticker, Quantity, Type, Action);
    output.close();
    sell s1 = new sell(Ticker,q,Type,Action);
    s1.Sell();
    System.out.println("buy done");
}

```

Figure 38: else block of controlTrade method

Figure 38 shows else block for the method being called. In else block Sell method will be called.

```

public void initialize(URL url, ResourceBundle resourceBundle) {
    ticker.setItems(FXCollections.observableArrayList( ...es: "AAPL", "TSLA", "GOOG", "FORD", "HPCQ", "MSET"));
    quantity.setItems(FXCollections.observableArrayList( ...es: "1", "2", "3", "4", "5", "6", "7", "8", "9", "10"));
    action.setItems(FXCollections.observableArrayList( ...es: "buy", "sell"));
    order_type.setItems(FXCollections.observableArrayList( ...es: "market", "limit"));
}

```

Figure 39: initialize method in TRADE

Figure 39 shows the initialize method for the TRADE CONTROLLER. In this method the options for the four combo boxes are being set. User can make selection from the options given in the combo boxes.

## PORTFOLIO FXML

Portfolio FXML or window was made to show user all the previous transactions he made, what he bought and what he sold for what price and which stock. This helps the user make remember that how is he spending all his money. Further, below this view is the profit calculator. One may ask what the functionality of this box is. The simple answer is, it tells you which stock to buy based on how many days you have and how much money want. It asks the user for two inputs, the amount he wants to get as output and the number of days in which he wants to have that profit. So, our profit calculator shows the stock name in which the user should invest in.

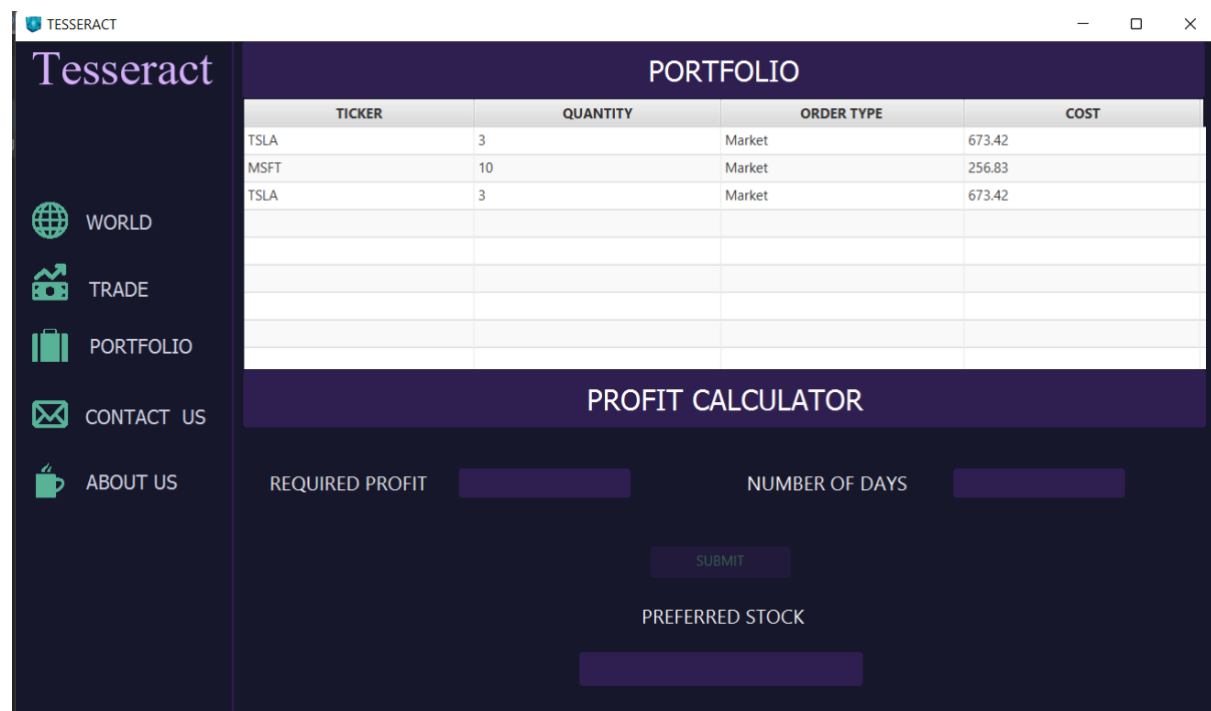


Figure 40: Portfolio Page

The above figure shows the whole Portfolio page. And if now the user entered his data in the below profit calculator it would be something like this:

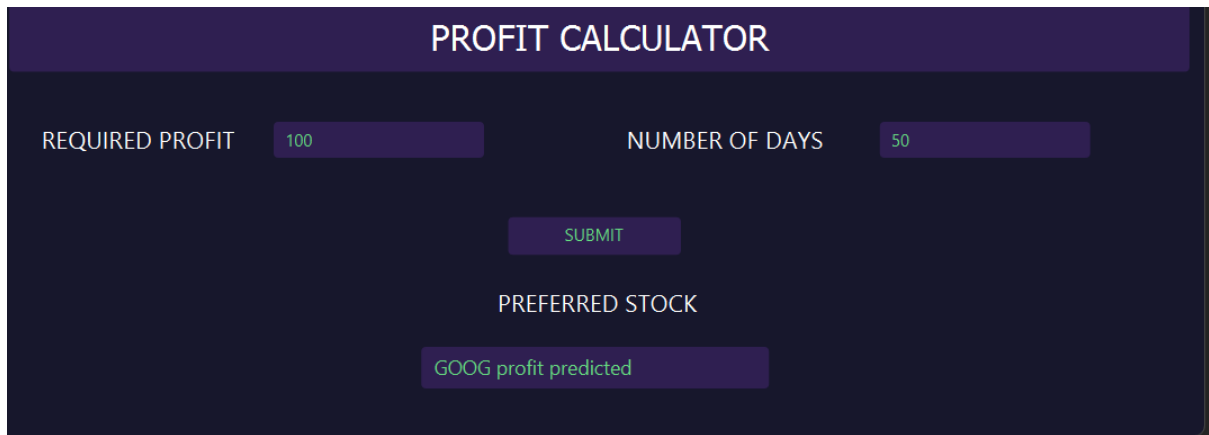


Figure 41: Profit Calculator

In the above figure, the user first entered the data and then clicked the submit button and then as an output “GOOG profit predicted” showed up telling the user that it is the best time to invest some money in the Google.

## DESIGNING PORTFOLIO

As in the above section, you saw the whole portfolio page the designing was never an easy task. First, to show user all the transactions he made we opted for using Label to show the previous transactions but that made us limited to show only a few transactions. Then we chose for the table view, but it wasn’t working well so after giving it hours we finally managed to work with the table view. This enabled us to show user all the previous transactions he made and keep them in record. The code for the table view in FXML file is as:

```
<TableView fx:id="table" layoutX="8.0" layoutY="50.0" prefHeight="236.0" prefWidth="839.0" style="-fx-
<columns>
  <TableColumn fx:id="ticker" editable="false" prefWidth="200.79999083280563" text="TICKER" />
  <TableColumn fx:id="quantity" prefWidth="215.1999969482422" text="QUANTITY" />
  <TableColumn fx:id="order_type" prefWidth="212.0" text="ORDER TYPE" />
  <TableColumn fx:id="cost" prefWidth="204.800048828125" text="COST" />
</columns>
</TableView>
```

Figure 42: Table View Code in Portfolio FXML

Now, coming to the profit predictor we used labels and text fields to ask for the input and show the required output.

## PORTFOLIO CONTROLLER

The packages and classes that were imported in this class are:

```

import javafx.beans.binding.Bindings;
import javafx.beans.property.Property;
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.fxml.Initializable;
import javafx.scene.control.*;
import javafx.scene.control.TableColumn.CellEditEvent;
import javafx.scene.control.cell.PropertyValueFactory;
import javafx.scene.control.cell.TextFieldTableCell;

import java.io.*;
import java.net.URL;
import java.util.InputMismatchException;
import java.util.NoSuchElementException;
import java.util.ResourceBundle;
import java.util.Scanner;

```

Figure 43: import statements in Portfolio Controller

The data members used in this controller were as follows:

```

public class portfolioController implements Initializable {

    1 usage
    public static final int daysReq = 0;

    1 usage
    public static final int profitReq = 0;

    6 usages
    ordersReader ord = new ordersReader();

```

Figure 44: Data Members in Portfolio

The **daysReq** store the value of how much days the user has and **profitReq** stores the value of how much profit the user wants. Whereas an object of **ordersReader** has been initiated which calculates all the previous transaction the user made to show it in the portfolio section of the Portfolio page. The labels, buttons, and the text field used are as below:



```
3 usages
@FXML
private TableColumn<Portfolio, String> cost;

5 usages
@FXML
private TextField no_of_days;

3 usages
@FXML
private TableColumn<Portfolio, String> order_type;

3 usages
@FXML
private TableColumn<Portfolio, String> quantity;
```

*Figure 45: Table Columns and Text Field in Portfolio Controller*

The cost column shows the value each of the stock cost. Number of days is a text field where the user enters his/her input. Order type is a column in the table view which shows that either the stock bought or sold was market or limit. Quantity column shows the number of stocks bought or sold.

```

5 usages
@FXML
private TextField req_profit;

4 usages
@FXML
private Button submit;

4 usages
@FXML
private TableView<Portfolio> table;

3 usages
@FXML
private TableColumn<Portfolio, String> ticker;

7 usages
@FXML
private TextField preferredStock;

```

Figure 46: Text Field, button, and Table Column in Portfolio Controller

Required profit asks for the user inputs. Submit is a button. The name of the table view is set to table. The ticker column asks shows the name of the stock bought or sold. Preferred Stock shows the user which stock to buy. One method is checkFields which checks if the user has entered the values in the text fields or not. If he hasn't entered, then the submit button will remain disable else will get enabled. This is shown as:

```

1 usage  HASAN-SYED1090 +1
void checkFields(){
    if(req_profit.getText().trim().equals("") && no_of_days.getText().trim().equals("") ) {

        submit.disableProperty().bind(req_profit.textProperty().isEmpty());
        submit.disableProperty().bind(no_of_days.textProperty().isEmpty());
        submit.disableProperty().bind(Bindings.or(no_of_days.textProperty()
            .lessThanOrEqualTo(String.valueOf(daysReq)),req_profit.textProperty().
            lessThanOrEqualTo(String.valueOf(profitReq))));
    }
}

```

Figure 47: checkField method in Portfolio

One method that was used to show values in the table is as follows:

```

public ObservableList<Portfolio> getPortfolio() throws IOException {
    ObservableList<Portfolio> portfolio = FXCollections.observableArrayList();

    System.out.println(ord.size);
    for (int i = 0; i < ord.size; i++) {
        portfolio.add(new Portfolio(ord.ticker[i],ord.quantity[i],ord.order_type, ord.cost[i]));
    }

    return portfolio;
}

```

Figure 48: getPortfolio method

This method sets the value in the table using a for loop and using the variable **ord** of class ordersReader.

## CONTACT US FXML

Contact us FXML was basically made so that the customer can interact with the faculty and report their problems through the email provided in the contact us window. So that we can work on these problems and resolve them. It is just a basic window so only thing that mattered was the colour scheme which would make it easy for the customer to read and access our email correctly without any mistake. Snippet is shown below.

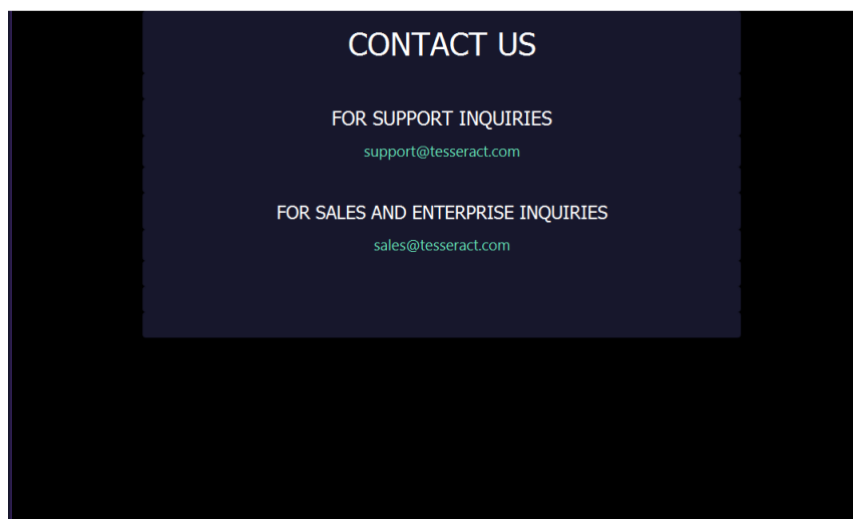


Figure 49: Contact US Page

Figure 49 shows the contact us window which helps the user report any issues or bugs with this desktop application.

## ABOUT US FXML

About us FXML file is just an introduction of who we are and why we made this app. This window actually talks about the background information, and it also makes a good impression on the customer and makes the user believe that we are here to stay and conquer.



*Figure 50: ABOUT US Page*

Figure 50 shows the about us window which shows our company's legacy and motivation.

## FLOW CONTROL

Flow control in java basically means that you can control the flow of the execution of methods depending upon the task. For example, some of the methods in TRADE window will only be executed when the user fills in the combo boxes and then click on the trade in method otherwise the flow of the program will omit that method. IF ELSE statements can be used to control the flow in your program. Code snippet given below will help you understand the concept of program's control flow.

```

if (ticker[0:4] == "FORD" ):
    ticker = "F"

elif (ticker[0:4] == "HPCQ" ):
    ticker = "HPQ"

elif (ticker[0:4] == "TSLA" ):
    ticker = "TSLA"

elif (ticker[0:4] == "GOOG" ):
    ticker = "GOOG"

elif (ticker[0:4] == "MSFT" ):
    ticker = "MSFT"
else:
    ticker = "AAPL"

```

*Figure 51: IF ELSE example*

Figure 51 shows how program's flow control can be changed based on the conditions implemented in the IF ELSE conditions. This figure shows the selection of the ticker you want to buy or sell based on the selection of the customer.

## COMPOSITE DATA TYPES

Composite data types of data members related to classes and arrays mainly. Composite data types are being used at different places in the project. Composite data types are important when you have to pass an entire object of a specific class as a parameter to the constructor or methods. Some of the code snippets of composite data types are shown below.

```

@FXML
private Button dealInButton;

@FXML
private ComboBox<String> action;

@FXML
private ComboBox<String> ticker;
@FXML
private ComboBox<String> quantity;

```

*Figure 52: Composite Data Types Example*

Figure 52 shows the composite data types. In this case the combo boxes data members are declared which is also a class of JAVA FX.

```
public String order_type = "Market";  
public String[] quantity = new String[10];  
public String[] ticker = new String[10];  
public String[] cost = new String[10];
```

*Figure 53: String type Arrays*

Figure 53 also shows the arrays of type String. Such data members are known as data members of composite types.

## Method (Function) Overloading

We have used method overloading in making a prediction of profit as shown below:

```
profitprediction(double requiredprofit)  
{  
    this(requiredprofit, days: 30);  
}  
profitprediction(double requiredprofit, int days){  
    setDays(days);  
    setRequiredprofit(requiredprofit);  
}
```

## SETTERS, GETTERS, DEFAULT AND PARAMETERIZED CONSTRUCTORS

The use of setters and getters as well as default and no argument constructors cannot be denied as it is the most common requirements in almost all OOP related projects in JAVA. Some of the code snippets showing setters and getters as well as default and parameterized constructors are shown below.

```

public String getTicker() { return ticker; }

public void setTicker(String ticker) { this.ticker = ticker; }

public int getQuantity() { return quantity; }

public void setQuantity(int quantity) { this.quantity = quantity; }

```

*Figure 54: setters and getters example*

Figure 54 shows the use of setters and getters in our project. These setters and getters are used to retrieve the stocks from the user selection in combo box and is used through getter methods to place orders of the stock selected and then show it in the PORTFOLIO window.

```

public Trade(){
    this( ticker: "", quantity: 0, ordertype: "", side: "");
}

public Trade(String ticker, int quantity, String ordertype, String side) {
    setTicker(ticker);
    setQuantity(quantity);
    setOrdertype(ordertype);
    setSide(side);
}

```

*Figure 55: Default and Parametrized Constructor*

Figure 55 shows the default and parameterized constructors that was used in our project in the profitprediction.java file.

## METHOD OVERRIDING, CONSTRUCTORS IN SUBCLASSES

We have used method overriding in trading stocks. Method overriding is involved in Trade and Buy classes. I have used constructors in subclasses of Trade class as shown below:

```

public class buy extends Trade {

    buy(String ticker, int quantity, String ordertype, String side){

        super(ticker,quantity,ordertype,side);

    }
}

```

## POLYMORPHISM

We have used Polymorphism in Trade controller class when we were making an object of buy class. You can see the use of Run time polymorphism in below snippet of code:

```
Trade b1 = new buy(Ticker,q,Type,Action);  
b1.Buy();
```

## this KEYWORD

this keyword is also very important in OOP. “this” keyword can be used for constructor chaining as well as to point to the current object reference. It can also be used to call the parameterized constructor as shown in figure 54. Moreover, this keyword has also been used in making setters as you can see in figure 54.

## STATIC CLASS MEMBERS

Static class members are not used very often in our project, but few examples are there from which you can understand the concept of static class members.

```
public class portfolioController implements Initializable {  
  
    public static final int daysReq = 0;  
    public static final int profitReq = 0;  
}
```

*Figure 56: static class members*

Figure 56 shows the use of static class members in portfolioController where it is necessary for the user not to enter any value in the text fields which is zero or less than zero otherwise error will be thrown while calculated the preferred stock.



```

void checkFields(){
    if(req_profit.getText().trim().equals("") && no_of_days.getText().trim().equals("") ) {

        submit.disableProperty().bind(req_profit.textProperty().isEmpty());
        submit.disableProperty().bind(no_of_days.textProperty().isEmpty());
        submit.disableProperty().bind(Bindings.or(no_of_days.textProperty()
            .lessThanOrEqualTo(String.valueOf(daysReq)), req_profit.textProperty()
            .lessThanOrEqualTo(String.valueOf(profitReq))));
    }
}
}

```

*Figure 57: usage of static class members*

Figure 57 shows where the static class members have been used. This checkfields method makes the submit button disabled in case wrong values are answered.

## INTERFACES, ABSTRACT CLASSES, AND METHODS

In this program we have also use the concept of abstract class in which method needs to be declared abstract (no body) (unless declared final) and no object can be made of abstract class. Basically, abstract class is made for general purposes and can be extended by concrete super-classes, but each abstract method needs to be implemented in concrete classes. In this Program TRADE class is abstract class whose snippets are shown below.

```

public abstract class Trade {
    String ticker;
    int quantity;
    String ordertype;
    String side;
}

```

*Figure 58: abstract class Trade*

Figure 58 shows the abstract class Trade and all of its data members

```

abstract void Buy()throws IOException, InterruptedException;
abstract void Sell()throws IOException, InterruptedException;

```

*Figure 59: data members of abstract class Trade*

Figure 59 shows the abstract methods which do not have any body. If other class extends this super class, then these methods need to have a body.

Abstract methods play a very pivotal role in our project. These methods are used to implement the functionality to buy and sell the stocks.

```
public class buy extends Trade {  
  
    buy(String ticker, int quantity, String ordertype, String side) { super(ticker, quantity, ordertype, side); }  
  
    public void Buy() throws IOException, InterruptedException {  
        try {  
            String s = null;  
  
            Process p = Runtime.getRuntime().exec("python C:\\Users\\HPP\\ProjectCrypto\\src\\main" +  
                "\\java\\com\\project\\projectcrypto\\buy.py");  
            BufferedReader in = new BufferedReader(new InputStreamReader(p.getInputStream()));  
            while((s = in.readLine()) != null){  
                System.out.println(s);  
            }  
        } catch (IOException ex) {  
            ex.printStackTrace();  
        }  
    }  
  
    public void Sell(){}  
}
```

Figure 60: Sub Class of Trade

Figure 60 shows that the class buy extends the abstract class Trade. The abstract method buy was also given a body. The main purpose of this method is to call the buy.py file and execute the code written in that file.

```
public class sell extends Trade {  
    sell(String ticker, int quantity, String ordertype, String side) { super(ticker, quantity, ordertype, side); }  
  
    public void Sell(){  
        try {  
            String s = null;  
  
            Process p = Runtime.getRuntime().exec("python C:\\Users\\HPP\\ProjectCrypto\\" +  
                "src\\main\\java\\com\\project\\projectcrypto\\sell.py");  
            BufferedReader in = new BufferedReader(new InputStreamReader(p.getInputStream()));  
            while((s = in.readLine()) != null){  
                System.out.println(s);  
            }  
        } catch (IOException ex) {  
            ex.printStackTrace();  
        }  
    }  
  
    public void Buy(){}  
}
```

Figure 61: Sub Class of Trade

Figure 61 shows the implementation of the sell abstract method. The description is the same as in figure 60.

## ARRAYS AND FUNCTIONS

Arrays and functions are an important part of any program. Arrays are widely used in our project to set up the options in combo boxes. Snippets are shown below.

```
public String[] quantity = new String[10];  
public String[] ticker = new String[10];  
public String[] cost = new String[10];
```

*Figure 62: Arrays*

Figure 62 shows the use of arrays, and they are used so that the user selection can be stored in these String type Arrays.

As far as functions are concerned, there are many methods in our project but the most important of them all is the BUY/SELL method used to buy or sell the stocks based upon the user selection. The details of these functions are given above in figure 6(C & D).

## EXCEPTIONAL HANDLING AND DATA VALIDATION

Most of the exceptional handling can be done by the built-in exceptional handling classes that contains different exceptions such as InputMismatchException or ArithmeticException or IOException. We have also made use of the same concepts. The Exceptional handling code Snippets are shown below.

```

try {

    int req = Integer.parseInt(req_profit.getText());
    int days = Integer.parseInt(no_of_days.getText());
    ratereader fetchrates = new ratereader();
    double ratetsla = fetchrates.getRatetsla();
    double rateaapl = fetchrates.getRateaapl();
    double ratemsft = fetchrates.getRatemsft();
    double ratef = fetchrates.getRatef();
    double ratehpg = fetchrates.getRatehpg();
    double rategoog = fetchrates.getRategoog();

```

Figure 63: try block

Figure 63 shows that if user enters invalid input that exception will be thrown, and it cannot calculate the preferred stock. This code has been extracted from portfolio controller.

```

    } catch (FileNotFoundException e) {
        e.printStackTrace();
    } catch (NumberFormatException e) {
        System.err.println("Enter valid no.");
    } catch (InputMismatchException e) {
        System.err.println("Enter digits only");
    }

```

Figure 64: catch block

Figure 64 shows the exception handling catch block which forces the user to enter the correct input into the text field.

As far as data validation is concerned, there is a method of checkfields which disables the deal in button in Trade window and also the submit button in PORTFOLIO window if wrong input is given. The code snippets are shown below.

```

void checkFields(){
    if(req_profit.getText().trim().equals("") && no_of_days.getText().trim().equals("") ) {

        submit.disableProperty().bind(req_profit.textProperty().isEmpty());
        submit.disableProperty().bind(no_of_days.textProperty().isEmpty());
        submit.disableProperty().bind(Bindings.or(no_of_days.textProperty()
            .lessThanOrEqualTo(String.valueOf(daysReq)), req_profit.textProperty()
            .lessThanOrEqualTo(String.valueOf(profitReq))));
    }
}

```

Figure 65: check Fields method

This figures show the code that disables the button if wrong or negative input is given in the text fields.

## OUTPUTS

Some of the outputs of our project are as follows.

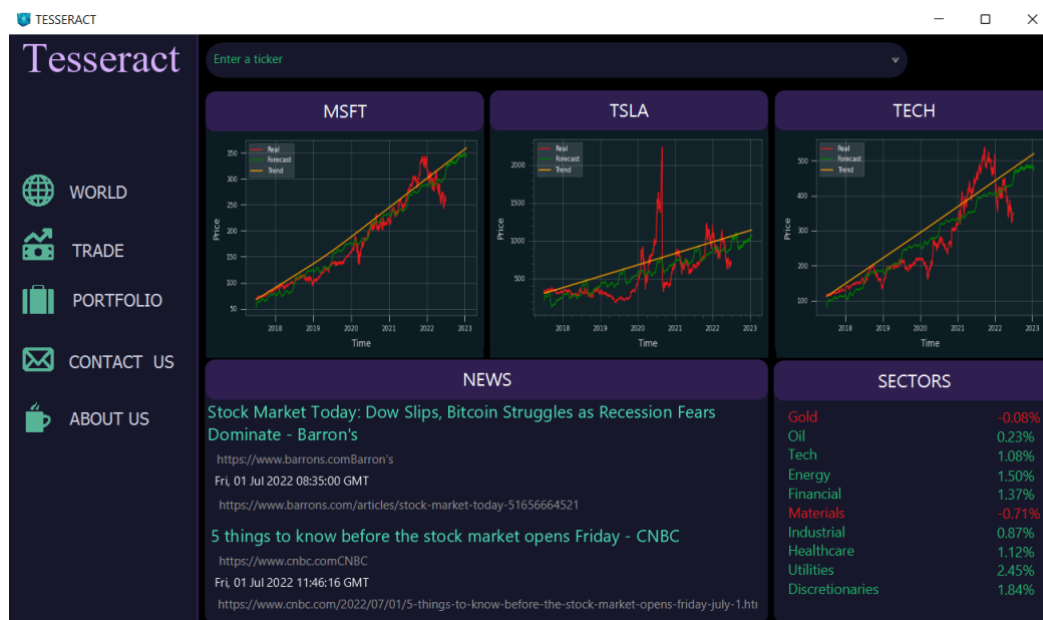


Figure 66: World Page

This figure shows the window that will appear on the screen when you run the program. This window is also the WORLD window.

When you open the GOOGLE PAGE:

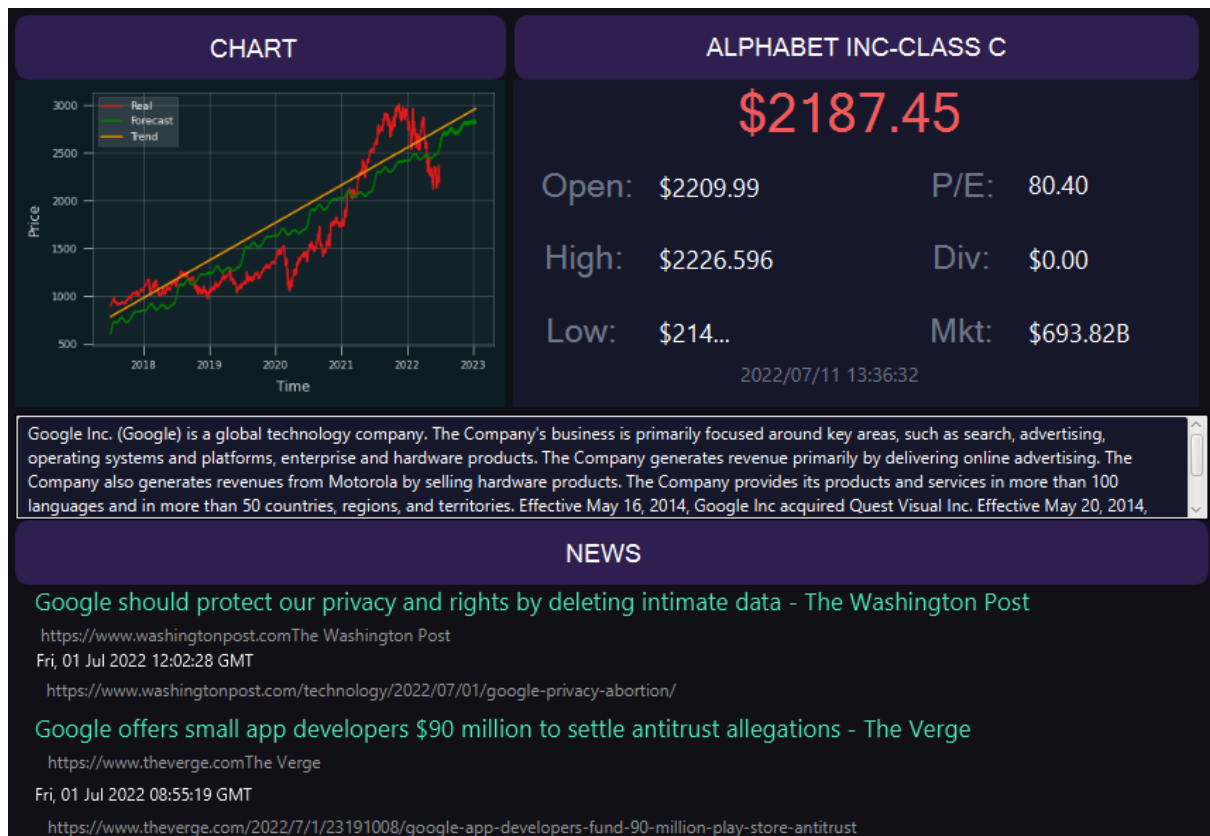


Figure 67: Google Page

## APPLE PAGE:



Figure 68: Apple Page

## FORD PAGE

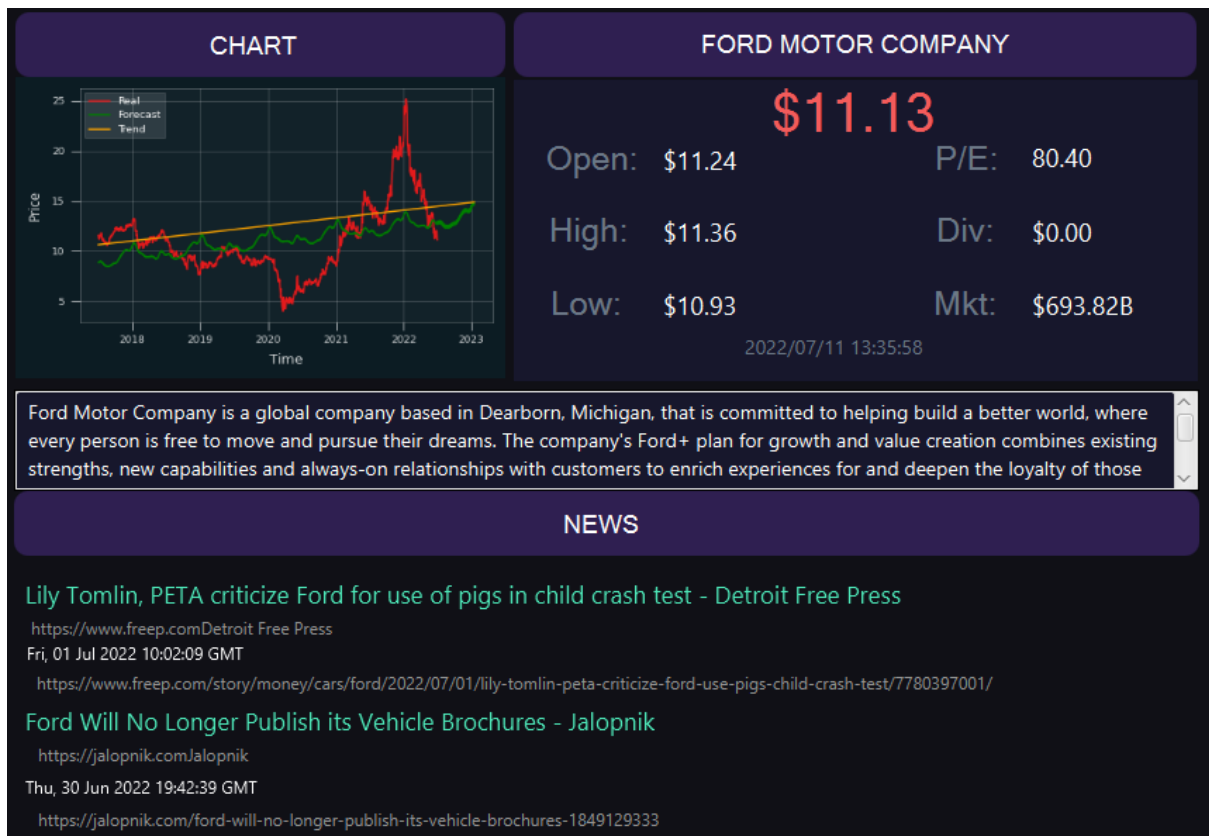


Figure 69: Ford Page



## HP PAGE:

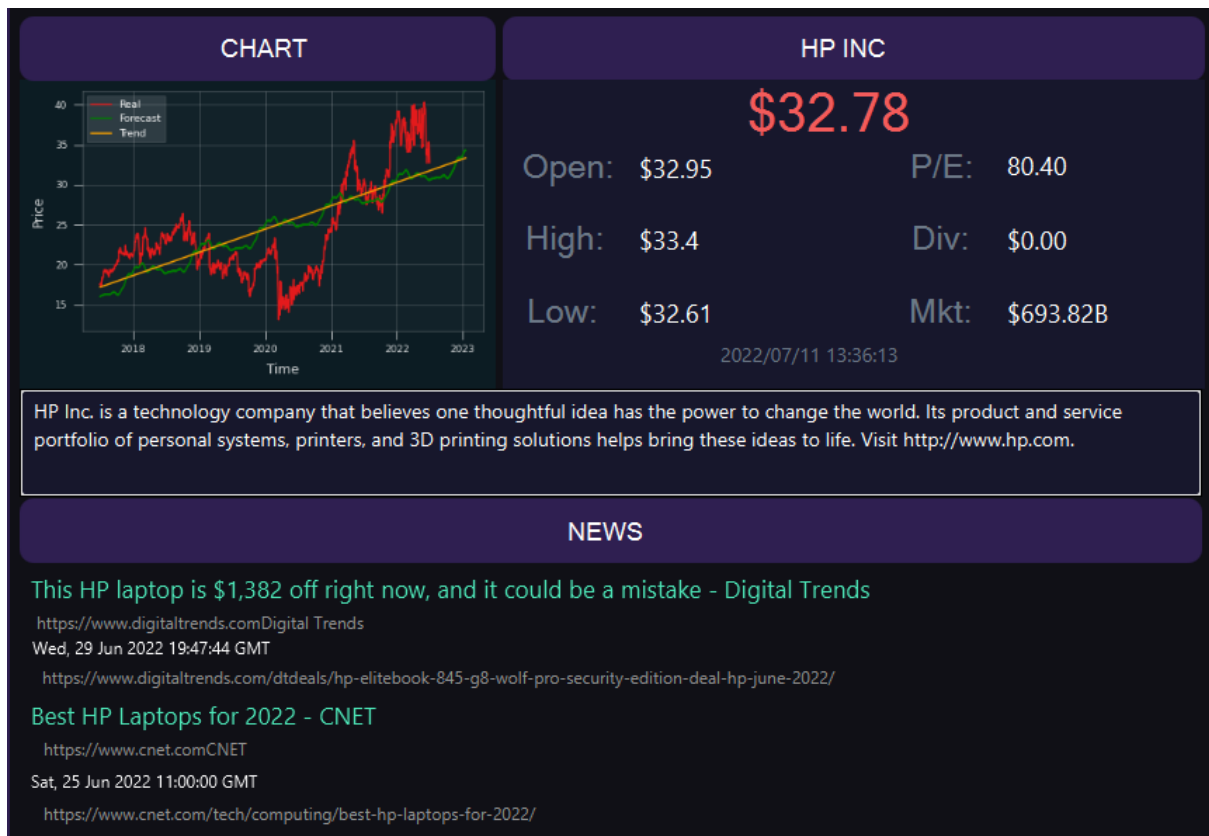


Figure 70: HP Page

## MICROSOFT PAGE



Figure 71: Microsoft Page

## TESLA PAGE



Figure 72: Tesla Page

## TRADE PAGE

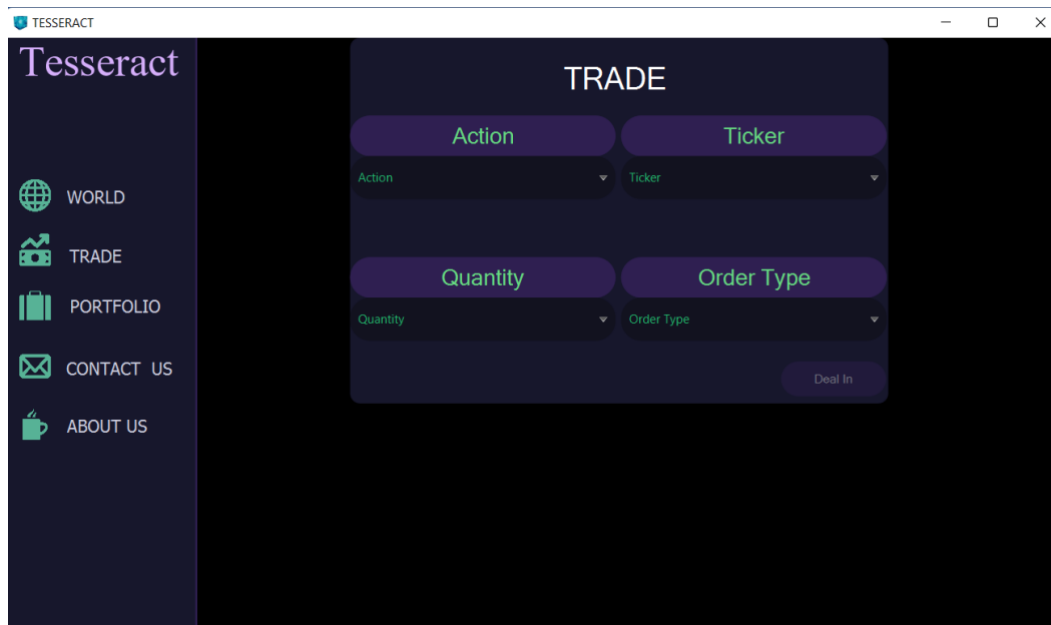


Figure 73: Trade Page

This window appears on the screen when you press the TRADE button on left hand side of the menu. As you can see no option is selected in the combo boxes therefore user cannot click on the deal in button.

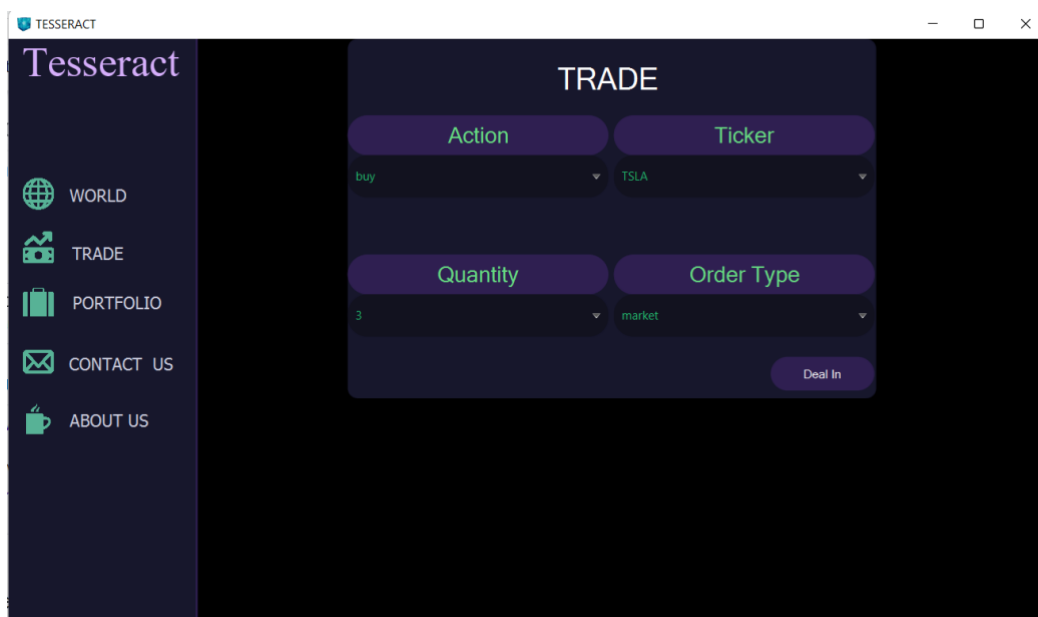


Figure 74: dealIn button only works if all the combo boxes are selected

Figure shows that user can now click on the deal in button after he/she select the options from the four combo boxes.

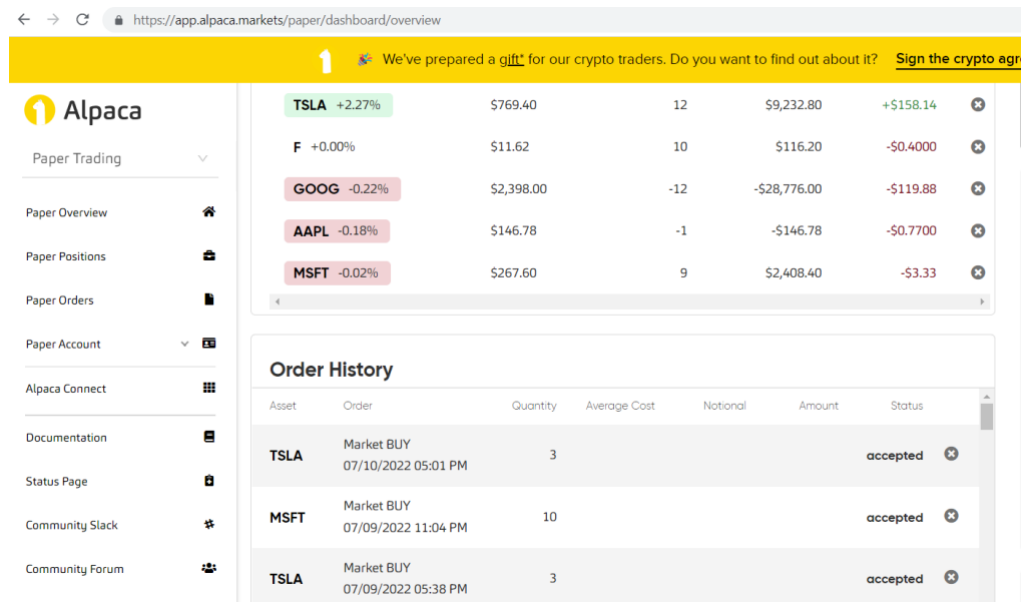


Figure 75: Alpaca Trading Website

Figure shows the website ALPACA TRADING WEBSITE where you can see the details of the order placed along with time at which the order took place. You can also confirm your order from the portfolio window.

## ASSOCIATED PYTHON FILES

There are some Python files that were integrated with Java code for some purposes like buying stocks, selling stocks, retrieving data from APIs, and forecasting the prices of stock in future. The following python files serve these purposes:

### BUY.PY FILE

Buy.py file contains code for placing buy order to the Alpaca API.

```
import alpaca_trade_api as tradeapi
API_KEY = 'PKGGM516KZYPV0I7U9L9'
SECRET_TOKEN = 'Z191P8Ey5ckdrRcjTXyMVVcBooyFypRSQtnwi5ci'
BASE_URL = 'https://paper-api.alpaca.markets'
api = tradeapi.REST(
    base_url = BASE_URL,
    key_id = API_KEY,
    secret_key = SECRET_TOKEN
)
```

Figure 76: Configuring Alpaca API for buying stocks

In the above snippet of code, we are importing Alpaca Trade API and configuring API key, Secret token and Base URL with the API. This API provides an easy-to-use trading interface to users to trade stocks. Each user is provided with a unique API key and a Secret token. A person can use this API key in his programs to make buy and sell stocks.

```
file = open("buy.txt", "r")
text = file.readlines()
ticker=text[0]
quantity = text[1]
ordertype= text[2]
side= text[3]
file.close()
```

Figure 77: Reading contents of Buy.txt file

In the above code, we are opening a buy.txt file for reading. The contents of this file are shown below:

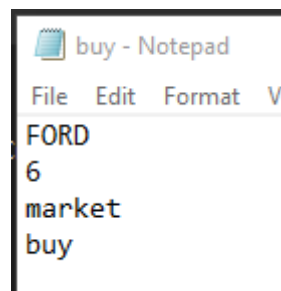


Figure 78: Buy.txt file

So, we are reading the contents of this file and storing in variables like ticker, quantity, order type and side. Then we are closing the file.

```

order = api.submit_order(symbol=ticker,
                          qty = quantity,side=side, type=ordertype,
                          time_in_force= 'day')

```

Figure 79: Placing buy order

In the above snippet, we are submitting the order to the API with arguments as ticker (Stock), quantity, side, and order type. The API will further place the order in the real market.

```

f = open("orders.txt", "a")
f.write("Buy \n")
f.write(ticker+ "\n")
f.write(quantity+ "\n")
f.write(ordertype+ "\n")
f.write(side+ "\n")
f.write("\n")
f.close()

```

Figure 80: Writing buy order in Orders.txt

In the above code, we are appending to an orders.txt file the details regarding the buying order that was just placed with the help of the API. We are writing the name of ticker, its quantity, order type and side. Then we close this file.

## SELL.PY FILE

Sell.py file contains code for placing buy order to the Alpaca API.

```

import alpaca_trade_api as tradeapi
API_KEY = 'PKGGM516KZYPVOI7U9L9'
SECRET_TOKEN = 'Z191P8Ey5ckdrRcjTXyMVVcBooyFypRSQtnwi5ci'
BASE_URL = 'https://paper-api.alpaca.markets'
api = tradeapi.REST(
    base_url = BASE_URL,
    key_id = API_KEY,
    secret_key = SECRET_TOKEN
)

```

Figure 81: Configuring Alpaca API for selling stocks

In the above snippet of code, we are importing Alpaca Trade API and configuring API key, Secret token and Base URL with the API. This API provides an easy to use trading interface to users to trade stocks. Each user is provided with a unique API key and a Secret token. A person can use this API key in his programs to make buy and sell stocks.

```

file = open("sell.txt", "r")
text = file.readlines()
ticker=text[0]
quantity = text[1]
ordertype= text[2]
side= text[3]

```

Figure 82: Reading contents of Sell.txt file

In the above code, we are opening a sell.txt file for reading. The contents of this file are shown below:

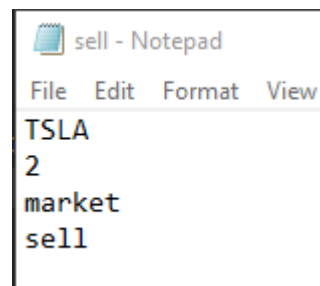


Figure 83: Sell.txt file

So, we are reading the contents of this file and storing in variables like ticker, quantity, order type and side. Then we are closing the file.

```

order = api.submit_order(symbol=ticker,
    qty = quantity,side=side, type=ordertype,
    time_in_force= 'day')

```

Figure 84: Placing sell order

In the above snippet, we are submitting the order to the API with arguments as ticker (Stock), quantity, side, and order type. The API will further place the order in the real market.

```

f = open("orders.txt", "a")
f.write("Sell \n")
f.write(ticker+ "\n")
f.write(quantity+ "\n")
f.write(ordertype+ "\n")
f.write(side+ "\n")
f.write("\n")
f.close()

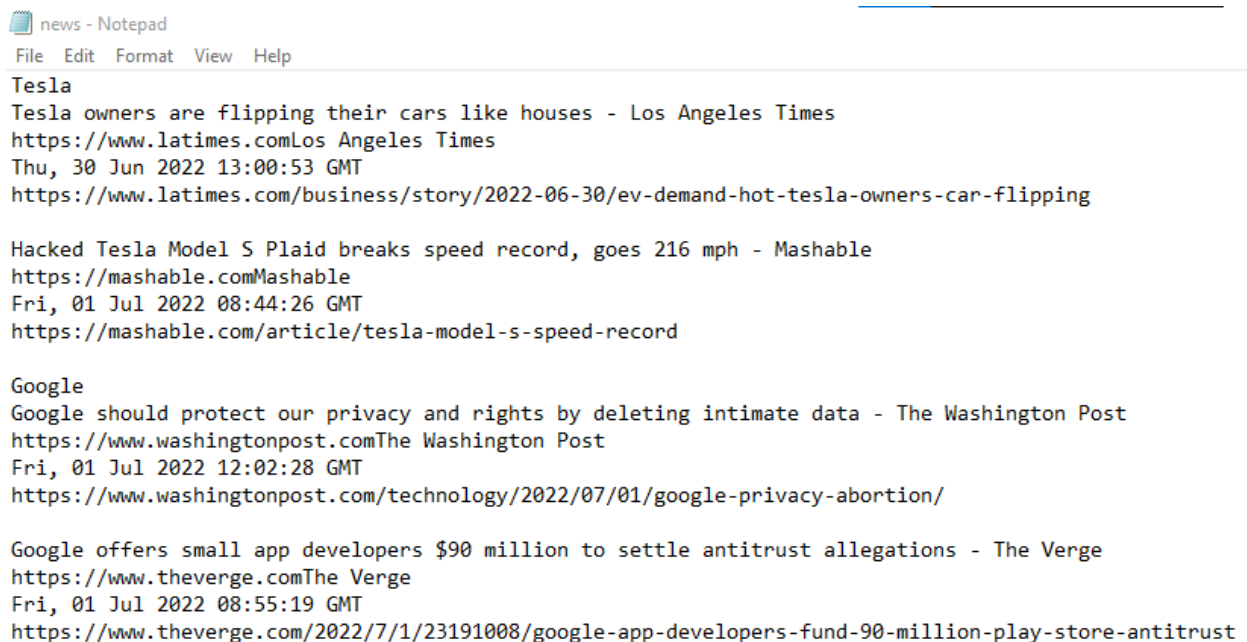
```

Figure 85: Writing details of sell order in Orders.txt

In the above code, we are appending to an orders.txt file the details regarding the selling order that was just placed with the help of the API. We are writing the name of ticker, its quantity, order type and side. Then we close this file.

## NEWS.PY FILE

The python script in news.py is retrieving news from google news API and writing in the news.txt file. The contents of the news.txt file can be seen below:



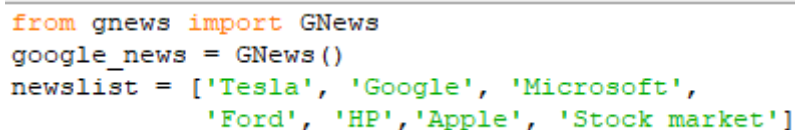
```
news - Notepad
File Edit Format View Help
Tesla
Tesla owners are flipping their cars like houses - Los Angeles Times
https://www.latimes.comLos Angeles Times
Thu, 30 Jun 2022 13:00:53 GMT
https://www.latimes.com/business/story/2022-06-30/ev-demand-hot-tesla-owners-car-flipping

Hacked Tesla Model S Plaid breaks speed record, goes 216 mph - Mashable
https://mashable.comMashable
Fri, 01 Jul 2022 08:44:26 GMT
https://mashable.com/article/tesla-model-s-speed-record

Google
Google should protect our privacy and rights by deleting intimate data - The Washington Post
https://www.washingtonpost.comThe Washington Post
Fri, 01 Jul 2022 12:02:28 GMT
https://www.washingtonpost.com/technology/2022/07/01/google-privacy-abortion/

Google offers small app developers $90 million to settle antitrust allegations - The Verge
https://www.theverge.comThe Verge
Fri, 01 Jul 2022 08:55:19 GMT
https://www.theverge.com/2022/7/1/23191008/google-app-developers-fund-90-million-play-store-antitrust
```

Figure 86: News.txt file



```
from gnews import GNews
google_news = GNews()
newslist = ['Tesla', 'Google', 'Microsoft',
            'Ford', 'HP', 'Apple', 'Stock market']
```

Figure 87: Importing Google news API

In the above code, we are importing google news API and initializing it. Then I have formed a list of topics regarding which I want to extract news about.



```

f = open("news.txt", "w")
for i in newslst:
    news = google_news.get_news(i)
    f.write(i + "\n")
    y=0
    for x in news[0:2]:
        f.write(news[y]['title'] + "\n")
        f.write(news[y]['publisher']['href'])
        f.write(news[y]['publisher']['title'] + "\n")
        f.write(news[y]['published date'] + "\n")
        f.write(news[y]['url'] + "\n")
        f.write("\n")
        y =y+1
f.close()

```

Figure 88: Retrieving news from API and writing in news.txt

In the above snippet, we have opened news.txt file for writing. Then we have looped through news list and extracted recent news regarding it like title, publisher, URL, and publishing date. Then I have closed this file.

## RATES.PY:

The python script in rates.py is calculating the rate of every stock and writing in rates.txt file. The contents of the rates.py can be seen below:

```

import pandas as pd
stocklist=['TSLA','AAPL','MSFT','F','HPQ','GOOG']
rates = []

```

Figure 89: Pandas and stocks list

In the above code, we are importing pandas a library which is used for dealing with data frames. Then I have formed a list of stocks and rates.

```

def slope(data):
    first = data.loc[1228, 'close']
    last = data.loc[1255, 'close']
    diff = (last - first)/30
    diff = round(diff,2)
    rates.append(diff)
    return diff

```

Figure 90: Slope function

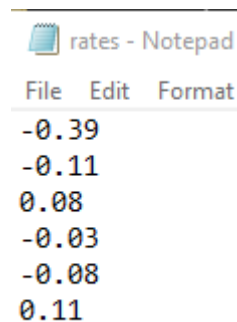
In the above snippet, we are defining a function slope which takes a data frame as a parameter and stores stock prices at different points. Then it finds the difference and divides

by 30 to get the increase or decrease in value of stock per day. Then the rates are appended in rates list.

```
f = open("rates.txt", "w")
for i in stocklist:
    df = pd.read_csv(i+'.csv')
    df.drop(['symbol', 'high', 'low', 'open', 'volume'
            , 'adjClose', 'adjHigh', 'adjLow', 'adjOpen'
            , 'adjVolume', 'divCash', 'splitFactor'],
            axis=1, inplace=True)
    df['Year'] = df['date'].apply(lambda x: str(x)[:21])
    df['Month'] = df['date'].apply(lambda x: str(x)[5:7])
    df['Day'] = df['date'].apply(lambda x: str(x)[8:10])
    df['ds'] = pd.DatetimeIndex(df['Year']+'-'+df['Month']+'-'+df['Day'])
    df.drop(['Year', 'Month', 'Day', 'date'], axis = 1, inplace = True)
    data = df.iloc[-40:]
    rate = slope(data)
    f.write(str(rate) + "\n")
```

Figure 91: Calculating rate of every stock and writing in rates.txt

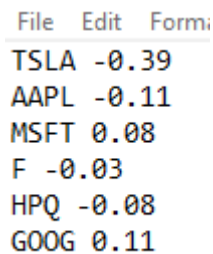
In the above code, we are opening rates.txt file for writing rates of stock in it. Then we have looped over the stocks list and extracted data from the excel files regarding the stock. We have called the slope function for every stock and write its rate in the file. The rates file looks something like this:



```
rates - Notepad
File Edit Format
-0.39
-0.11
0.08
-0.03
-0.08
0.11
```

Figure 92: Rates.txt

The reference for this rate looks something like this:

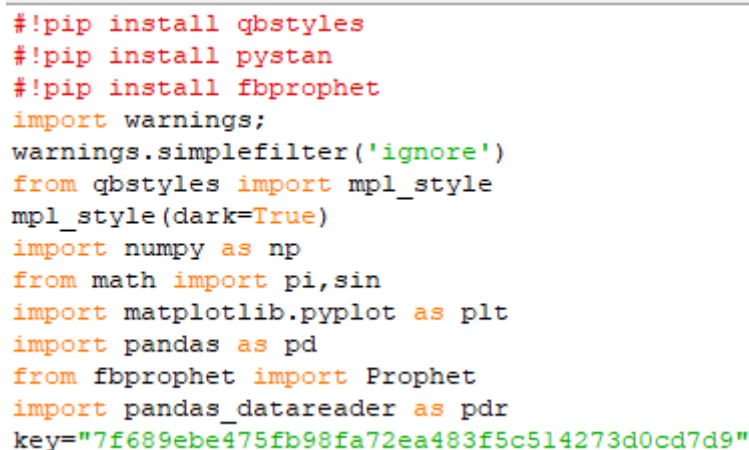


	File	Edit	Form:
TSLA	-0.39		
AAPL	-0.11		
MSFT	0.08		
F	-0.03		
HPQ	-0.08		
GOOG	0.11		

Figure 93: Rates reference

## FORECASTING.PY FILE

The python script in forecasting.py is retrieving stock data from Tingo API and making future predictions in the prices of stock. The contents of this file can be seen below:



```
#!/pip install qbstyles
#!/pip install pystan
#!/pip install fbprophet
import warnings;
warnings.simplefilter('ignore')
from qbstyles import mpl_style
mpl_style(dark=True)
import numpy as np
from math import pi,sin
import matplotlib.pyplot as plt
import pandas as pd
from fbprophet import Prophet
import pandas_datareader as pdr
key="7f689ebe475fb98fa72ea483f5c514273d0cd7d9"
```

Figure 94: Importing required libraries

In the above code, we are installing qbstyles, pystan and Facebook Prophet. We have imported warnings, numpy, math, pandas, pandas data reader and Prophet. Also I have set the API key of Tingo. Tingo API provides data related to stock prices. We are using numpy for dealing with arrays. Pandas for dealing with data frames and pandas data reader for stock data. We are using Facebook Prophet for Time series forecasting.

```
def line_plot(na,df,forecast):
    name=na
    plt.xlabel('Time')
    plt.ylabel('Price')
    plt.plot(df['ds'],df['y'])
    plt.plot(forecast['ds'],forecast['yhat'],color='green')
    plt.plot(forecast['ds'],forecast['trend'],color='orange')
    plt.tight_layout()
    plt.legend(["Real", "Forecast","Trend"], loc ="upper left")
    img=na+'.png'
    #print(img)
    plt.savefig(img,pad_inches = 10,quality = 100)
    plt.clf()
```

Figure 95: Plotting function

In the above snippet, we are defining a function `line_plot()`. This function takes name of the stock, data frame consisting of prices and dates, and forecasted data frame. We are using matplotlib to plot the graph. We are setting Time as label on x axis and Price as y axis. Then we are plotting real values of stock on graph with red colour, forecasted values with green colour, and trend of stock as orange colour. Then we are placing the legend on the graph to help understand it. We save the graph on the system. Then we clear the plot.

```
for i in stocklist:
    df = pdr.get_data_tingo(i, api_key=key)
    name=i+'.csv'
    df.to_csv(name)
    filenames.append(name)
```

Figure 96: Retrieving data from Tingo API

In the above code, we are looping through the stock list and extracting data from the tingo API. We are also saving the data frame in the form of an excel format. Then we append the name to the file names list.

The data retrieved from API is of the following format:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	symbol	date	close	high	low	open	volume	adjClose	adjHigh	adjLow	adjOpen	adjVolum	divCa	splitFac
2	TSLA	2017-07-03	352.6	371.35	351.5	370.2	6297330	70.524	74.27	70.3	74.048	31486650	0	1
3	TSLA	2017-07-05	327.1	347.24	326.3	347.2	1.7E+07	65.418	69.448	65.266	69.44	84417480	0	1
4	TSLA	2017-07-06	308.8	320.79	306.3	317.3	1.9E+07	61.766	64.15798	61.26	63.452	95945975	0	1
5	TSLA	2017-07-07	313.2	317	307.4	313.5	1.4E+07	62.644	63.4	61.476	62.7	70268750	0	1
6	TSLA	2017-07-10	316.1	317.94	303.1	312.9	1.4E+07	63.21	63.588	60.626	62.58	68510400	0	1
7	TSLA	2017-07-11	327.2	327.28	314.3	316	1.1E+07	65.444	65.456	62.86	63.2	57000270	0	1
8	TSLA	2017-07-12	329.5	333.1	324.5	330.4	1E+07	65.904	66.62	64.9	66.08	51489195	0	1
9	TSLA	2017-07-13	323.4	331.6	320	330.1	8540442	64.682	66.32	63.994	66.022	42702210	0	1
10	TSLA	2017-07-14	327.8	328.42	321.2	323.2	5590393	65.556	65.684	64.244	64.638	27951965	0	1

Figure 97: Retrieved data format

```

for i in filenames:
    df = pd.read_csv(i)
    df.drop(['symbol', 'high', 'low', 'open', 'volume',
            'adjClose', 'adjHigh', 'adjLow', 'adjOpen',
            'adjVolume', 'divCash', 'splitFactor'],
            axis=1, inplace=True)
    df['Year'] = df['date'].apply(lambda x: str(x)[-21])
    df['Month'] = df['date'].apply(lambda x: str(x)[5:7])
    df['Day'] = df['date'].apply(lambda x: str(x)[8:10])
    df['ds'] = pd.DatetimeIndex(df['Year']+'-'+df['Month']+'-'+df['Day'])
    df.drop(['Year', 'Month', 'Day', 'date'], axis = 1, inplace = True)
    df.columns = ['y', 'ds']
    name='df_'+i
    df.to_csv(name)
    mod = Prophet(interval_width=0.95,
                  daily_seasonality=True,
                  changepoint_prior_scale=0.001,
                  seasonality_prior_scale=0.01)
    model = mod.fit(df)
    future = mod.make_future_dataframe(periods=200,freq='D')
    forecast = mod.predict(future)
    name2 = 'forecast_'+i
    df.to_csv(name2)
    na=i[:-4]
    #print(na)
    line_plot(na,df,forecast)

```

Figure 98: Forecasting stock values and plotting

In the above snippet of code, we are looping through the filenames and reading the csv file associated with that name. Then we are dropping the columns which are not relevant. Facebook Prophet wants the data in a specific format to make predictions. So, we are making changes in the date column. Then we are renaming the date column as ds and close(price) as y. We are making an instance of Facebook Prophet in next line with various hyper parameters. Then we provide the data to the model for training and later we make future

predictions for next 200 days. We are also saving the forecasted data frame. Later we are calling the plotting function we defined earlier to get the graph.



*Figure 99: Facebook Prophet result*

In the above figure you can view a graph between time and stock price of a company i.e., Microsoft. On the x axis we have time and on y axis we have price. There are three plots present on this graph. The yellow line shows the trend of the stock over the years. The red line shows the real values of stock, and the green line shows the forecasted values of the stock when the model was trained. You can observe that the yellow and green line move ahead of the red line with respect to x axis. This is because this graph shows the predicted price of stock over the next 200 days. You can also observe that predicted values are not always true with respect to real values, this shows low accuracy of the model. Facebook Prophet is an easy-to-use forecasting model but there is a fallback of less accuracy. To solve this problem, we can use Deep learning approach or Statistical approach. Deep learning approach includes sequential models like RNN (Recurrent Neural Network) and LSTM (Long Short-Term Memory). You can observe architecture of LSTM below:

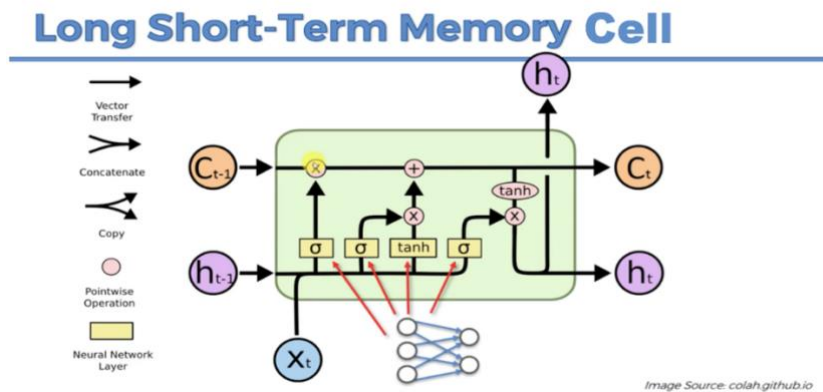


Figure 100: LSTM Architecture

When we tried using LSTM, we got better results. The results were:

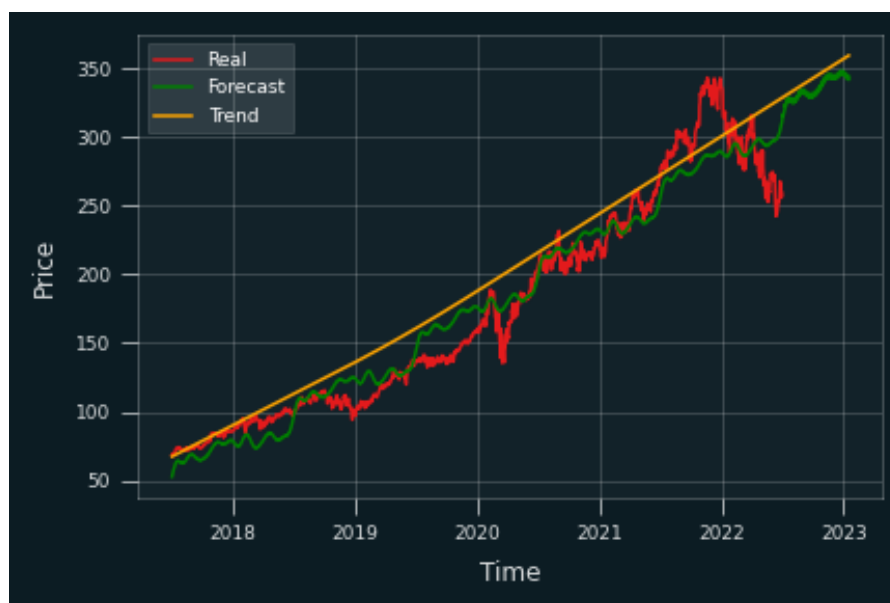


Figure 101: LSTM results

We can observe that there is quite less noise than the previous results. But the fallback of using LSTM is that it is computationally expensive and it takes much time to train model for all stocks individually.

The other approach is Statistical approach which includes models like ARIMA and SARIMAX. These models require a lot of hyper parameter tuning to get good results. You have to apply Ducky Fuller test every time for each stock. You have to care about seasonality

and other parameters. Hence this approach cannot be used if we have to automate the process of producing forecasted graphs.

Hence as a conclusion, we decided to use Facebook Prophet in the project.

## **CONCLUSION**

Now entry level traders can benefit from this program to invest in profitable stocks without fear of losing their hard-earned money. This is a first of its kind project as previously nothing was available to forecast the trends of the stocks in the market.