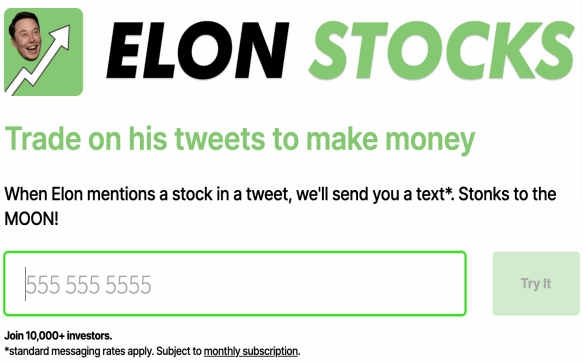
Background:
Cryptocurrency is a form of currency without any physical intrinsic value. It was created as a decentralised currency which was fully online. The way to obtain the cryptocurrency is by ‘mining’ complex maths equations. The method of mining is by using large computational bruteforcing using powerful computers. There is dedicated hardware called ASIC miners available for purchase with the sole intention of mining cryptocurrency. The cryptocurrency boom occurred when the first cryptocurrency referred to as ‘Bitcoin’ was created by a team of programmers. Whilst initially almost valueless, soon the Bitcoin rose to become worth 25,391 British Pounds. During this time, other crypto currencies such as ‘DogeCoin’ came into fruition. Recently in modern events, big technology moguls and CEO’s such as Elon Musk recently expressed their interest and admiration for Cryptocurrencies by tweeting various different tweets praising crypto. By these CEO’s mentioning the cryptocurrencies, the price of the cryptocurrencies fluctuates greatly. Something which can be desirable/undesirable. With the future of exchange and trade leaning towards crypto currency, due to its decentralised nature preventing banks and governments from tampering with it, an app to assist the trading of the currency would be very beneficial to the savvy investor.

Identification of problem:
Due to the volatility of the crypto to tweets by large influencers such as Elon Musk, a tool to monitor the price of a chosen crypto as well as a live feed to show the price of the crypto as well as alert the user when Elon Musk tweets something could prove beneficial to investors. In addition to that, being able to do other functions could prove beneficial such as viewing certain tweets about specific crypto to gain valuable information surrounding the interest and hype behind doge or being able to get graphs to view historical trends in the crypto would be effective if the solution was simple to use and have a simple user interface and display the information in an effective manner.

Description of current system:



Currently, there are few solutions available that do both the functions of a Twitter API as well as a Stock Market application. However, there is an online application called ‘Elon Stocks’ which can do both. The Elon Stocks app sends you a text message whenever Elon Musk tweets a message regarding Doge. It is a paid service consisting of a simple website where you input your phone number and it charges you a dollar a week. The application is of a similar type to mine, but it does not display the stock price live, is paid, and finally it cannot function if your mobile phone is switched off or out of range for cellular

communication, such as in a building or in an office.

Identification of Prospective Users:

Prospective users would be stock market brokers as well as investors both professional and casual. They can use the application to decide when to buy and sell crypto as well as monitor the current price. In addition to that, stock market analysts and other researchers can use the application to study and analyze the connection between tweets from large influencers and stock prices, specifically, Elon Musk. My specific user is my friend, Robert Carter, who is an investor in cryptocurrency, specifically Dogecoin. He has invested a large sum of equity into Dogecoin and has been influenced directly by large company CEOs such as Elon Musk. This application will allow him to monitor his investment and allow him to make stock exchange moves in an educated manner.

Identification of user needs:

1.) “Should show graphs of the doge cryptocurrency real-time, stating their price and various other information”.
2.) “It should show you the history of the currency from days ago to thus observe trends in stock price”.
3.) “It should be able to show the tweets Elon Musk made, and alert the user of such a tweet”.
4.) “It should store the data acquired by the API’S into an SQL database to be accessed”.
5.) “It should be simple to use and should have instructions on how to set up the application”.

6.) “It should be displayed in a bright and clear manner with simple-to-read elements”.

7.) “It should display tweets regarding crypto in a live feed where the user can see who tweeted what as well as the time and date of the tweet”.

8.) “It should alert the user either through TTS or an email that Elon Musk has tweeted as well as the current price of dogecoin”.

Objects Of the Project…

1.) User Need: The program should display the historical price of the Dogecoin cryptocurrency in the form of a graph that is embedded into the user interface. It should be hidden behind a button that says ‘Historical Price’.

1.) The graph should have the time on the x-axis as well as the price on the y-axis.
2.) The graph should be titled with the dates of the start as well as the end of the time period of the data.
3.) It should be plotted in the standard cartesian format.
4.) The program should generate the data when a button is pressed.
5.) The button should be labelled with ‘Historical Price’

2.) User Need: The program should display each element of the program into one GUI which is how the user will interact with the program.

1.) The program should display a window that is interactive to the user’s input.
2.) The program should have buttons that can click and respond to user input.
3.) Elements of the program should be hidden in other windows and be dynamically generated.
4.) The program should have live updating elements into it such as stock price graphs as well as a tweet feed.

3.) User Need: The program should have a live feed of tweets made by people in regards to DogeCoin as well as cryptocurrency.

1.) There should be a live feed of tweets being written at that moment.
2.) The live feed should be embedded in the GUI as an element of the GUI.
3.) The tweets should have a date/time stamp on them.

4.) User Need: The program should alert the user if Elon Musk has tweeted about anything regarding crypto or dogecoin.

1.) The program should send an email to the user in the format of a message.
2.) The program should be able to successfully send an email so that the message appears in the user’s inbox.
3.) The program should connect successfully to an email server to send the email successfully.
4.) The program should send the email automatically in the background and should not be embedded into the GUI.
5.) The user should be able to enter their information and login such as their email address and email server of their choosing.

5.) User Need: The program should display every tweet created by Elon Musk’s Twitter account in the form of a standard tweet.

1.) The displayed tweet should have the full length of the tweet.
2.) There should be a time/date stamp of the time the tweet was made.
3.) The tweet should be embedded into the GUI in the main dashboard.
4.) The displayed tweet should be formatted in the style of a standard tweet for ease of readability. Graphically.

6.) User Need: The program should store all the collected data on tweets as well as stock prices in a MySQL database for ease of analysis as well as data storage.

1.) Data collected from the Elon Musk tweet parser should be stored in the database
2.) Data collected from the Live Stock Price Parser should be in the database.
3.) The data should be formatted in the format DD/MM/YYYY with a timestamp.
4.) The data should be able to be accessed concurrently.
5.) Each module should have its own login to allow for ease of use.
6.) The data should be stored in individual tables for each module.

7.) User Need: The program should show the live graphs of the DogeCoin cryptocurrency. The graphs should have the price of the crypto on the graph and should be displayed as an element in the GUI.

1.) The graphs should update in regular intervals of 1-2 seconds.
2.) The graphs should be formatted with the time on the x-axis and the stock price on the y axis.
3.) The graph should be titled and be integrated into the Graphical User Interface.
4.) The graph should be laid out in the standard cartesian format.

``````````````````

**Chosen solution:**

For ease of use as well as documentation reasons, Python would be best suited for the project. Its ease of use in terms of interacting with APIs as well as its handling of complex OOP concepts is far more organised and concise than other languages. Also, its plethora of graphical user interface design applications such as Tkinter will become useful in designing the application. Whilst a console application could be used in creating the project, it would not be able to encompass all of the requirements such as 5.) and 6.). This is because it will be displayed in a text-only view instead of in a GUI format. Due to this, it is not suitable for this project.

GUI toolkits such as Tkinter or PyQt or wxPython are well established and provide a lot of documentation and support for creating a variety of applications. In addition to that, it supports all of the elements described in the overview of the project. Due to this, instead of writing a GUI layer from scratch, using such a toolkit would be much more effective and is used more by industry professionals.

For Twitter handling, the official Twitter API can be accessed using Python which allows for the features mentioned above. This can be done using the TwitterAPI package as well as the Tweepy package. Due to restrictions in the API, only a certain quota of tweets can be pulled per month however for demonstration purposes the limit should be fine, as the program will not be scaled up to industrial use.

For the graphical elements, they can be managed with the graphing package Matplotlib, which allows for data to be plotted easily. Without it, the complexity would be exponentially harder as you would have to create a package from scratch which would still be less efficient than the free package which is used frequently by professional applications.

For the stock market information, the library Yahoo Finance can be used to analyze the stock market and get the required information such as the historical price of certain stocks. The free library called yfinance (Yahoo Finance)  is available to anybody for free. In addition to this, web scraping will allow us to gain the live price of tickers.
``````````````````

Design Section

General explanation, high level.

The program will be developed in Python due to the availability of various API's such as the Twitter, stock and Graphing ones which I will use. In addition to the email API. Approaching this problem, I will be using object-oriented programming instead of functional or other modes of programming due to its high-level nature, ease of maintenance, greater modularity as well as the fact that applications such as these that require graphics elements are more suited to object-oriented programming. The program will be run on an X86 or ARM computer, and the type of computer, whether or not it is Linux, Mac or Windows is irrelevant as all major operating systems and hardware architectures within the last 10 years support python, which is an interpreted language. There is a speed penalty in using python over other compiled languages such as C# or java, but the access to APIs as well as suitable documentation outweighs any speed penalty in my opinion.

Explanation of the Graphing Algorithm:

The graphing algorithm is divided into two parts, firstly the live price, and secondly the historical price. The historical price can be done by the module connecting to the yahoo finance API, and requesting data on the desired ticker, 'DOGE'. The data is then returned as a pandas data frame which can be analyzed for the information desired. This can be plotted with time on the x-axis and the price on the y axis, using the Matplotlib package mentioned prior. Unlike the historical price algorithm, the live price algorithm achieves a live price by first querying the database for the information desired which it then plots. A call is made to the stock price module which inserts the latest time and prices of doge into the database, which is then requested by the graphing algorithm and plotted real-time using the animation package found in the Matplotlib package. Both of these are built into the main GUI, in separate windows each with a thread due to Matplotlib being unsafe for multithreading.

Explanation of the Twitter Feed Algorithm:

The algorithm, firstly connects to the Twitter API, using the required credentials and then it requests a certain number of tweets about a topic from all tweets made, it is then parsed and formatted into a readable correct format, which is then added to the ListBox widget present in the Homescreen. The Listbox is updated periodically in the main thread. This demonstrates the ability to use and handle API's.

Explanation of the Elon Musk Tweet Algorithm.

Initially, Elon's latest tweet is pulled from the Twitter API, which is used as a comparator. It then pulls the latest tweet Elon Musk made periodically, which it inserts into the database, the module then gets the latest database entry and compares it to the comparator. If it is different, then it can be assumed Elon Musk has tweeted. It then updates the homescreen with the new tweet, and emails the user with the stock price of DOGE. If the tweet has not

changed, it can be assumed that he has not tweeted something new and thus no email is sent, but the homescreen is still updated. It runs indefinitely until the program is closed. This shows another example of using API's to parse data.

Explanation of the Email Algorithm.

The email algorithm, unlike the other algorithms, is not coded in an 'object oriented' manner due to the fact that it is so small it is not necessary to. It would increase complexity, increase memory requirements and slow performance if by a small amount. The module connects to the sendgrid API, with the assistance of the OS import. When prompted, it runs a method which connects to the API using the API key and sends the latest price in the database to the desired email address in the form of an email. This also shows an example of using APIs to handle data.

Explanation of the GUI algorithm.

The GUI algorithm encompasses all of the other subclasses and methods of the other algorithms. There are three windows, each of which have a class to themselves and their own constructor etc etc. There is the main window, the live stock price window, as well as the historical price window. The main window has the twitter feed, the latest Elon tweet, as well as the live price non graphically. There is also the live price graphically in a separate window, with the historical price in another window. In addition to this, the main menu has buttons to the other windows. When a button is clicked, a class is created dynamically, which is created in the main thread. This is due to Matplotlib being very thread unsafe which is a limitation I had to overcome. The main menu is updated in a different thread, where a loop recursively calls itself which contains the methods to update the main menu. The update loop is stopped when a new window button is clicked. The GUI is created using Tkinter in an object oriented manner, with assistance from the threading modules. This demonstrates multithreading, complex OOP paradigms as well as various other higher level techniques.

Explanation of Data Structures.

The only data structures in the actual program are the standard ones built into python. There are uses of special 'datetime' data structures, in addition to other time/date data structures. The information from the API's appear in the form of a dictionary which has to be parsed for the information required. The database class has a query in the form of a List/Dictionary object which is sent to the SQL SERVER.

Explanation of Databases.

There is one database, with two tables, one is called Stock Price, and the other is called elon_musk_tweet. The stock price table contains the stockprice and the time and date of when that price was recorded. The Elon Musk Tweet table has the date/time the tweet was tweeted along with the tweet itself. There is a primary key present in each table which is referenced to get the latest entry by sorting the table by the highest primary key number. The database has concurrent access enabled to allow the modules to work in tandem, and for this, each module has two users, one for reading and one for writing to lower stress on the

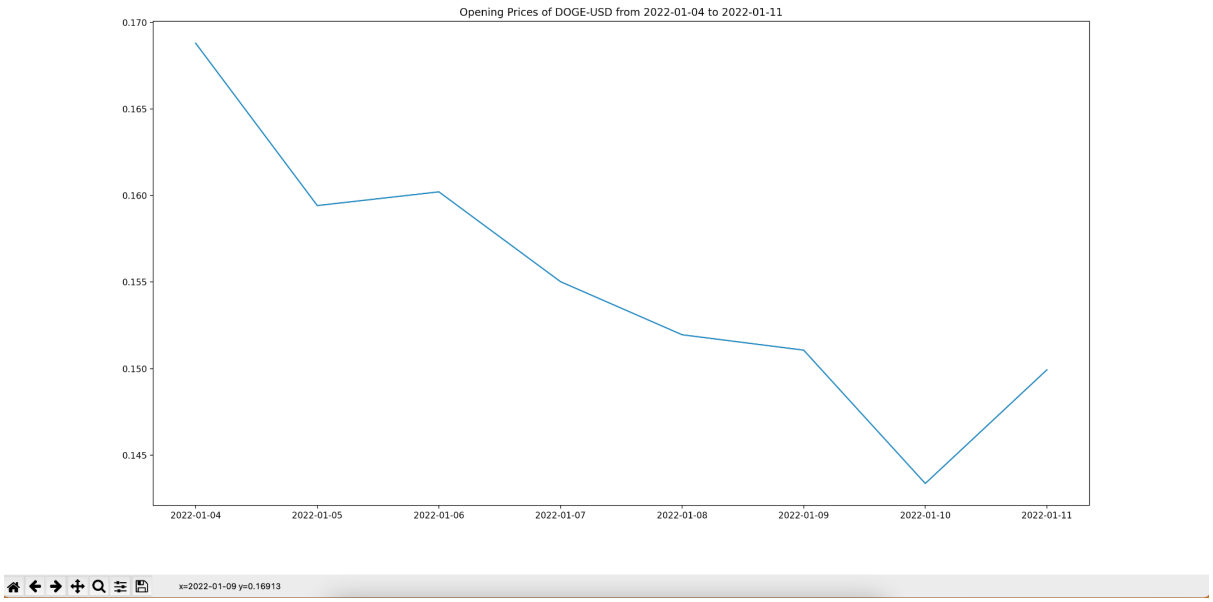
database. The database is run using MYSQL, locally on the machine. The database is initialised with the required tables and columns on startup of the program.

System integrity and security:

The database does not store any critical information, which needs to be encrypted, however the database is created locally, with a secure username and password. In addition to this, the Key's presented by the API's as well as credentials are stored securely locally, and in case of a leak, can be regenerated which nullifies the prior keys. The keys are also not able to be accessed after viewing them thus they must be remembered. Due to this, even if the perpetrator was to have access to the python file, it is still completely secure.

Design Of User Interface (annotate)

Home Screen



Historical Price

Live Price

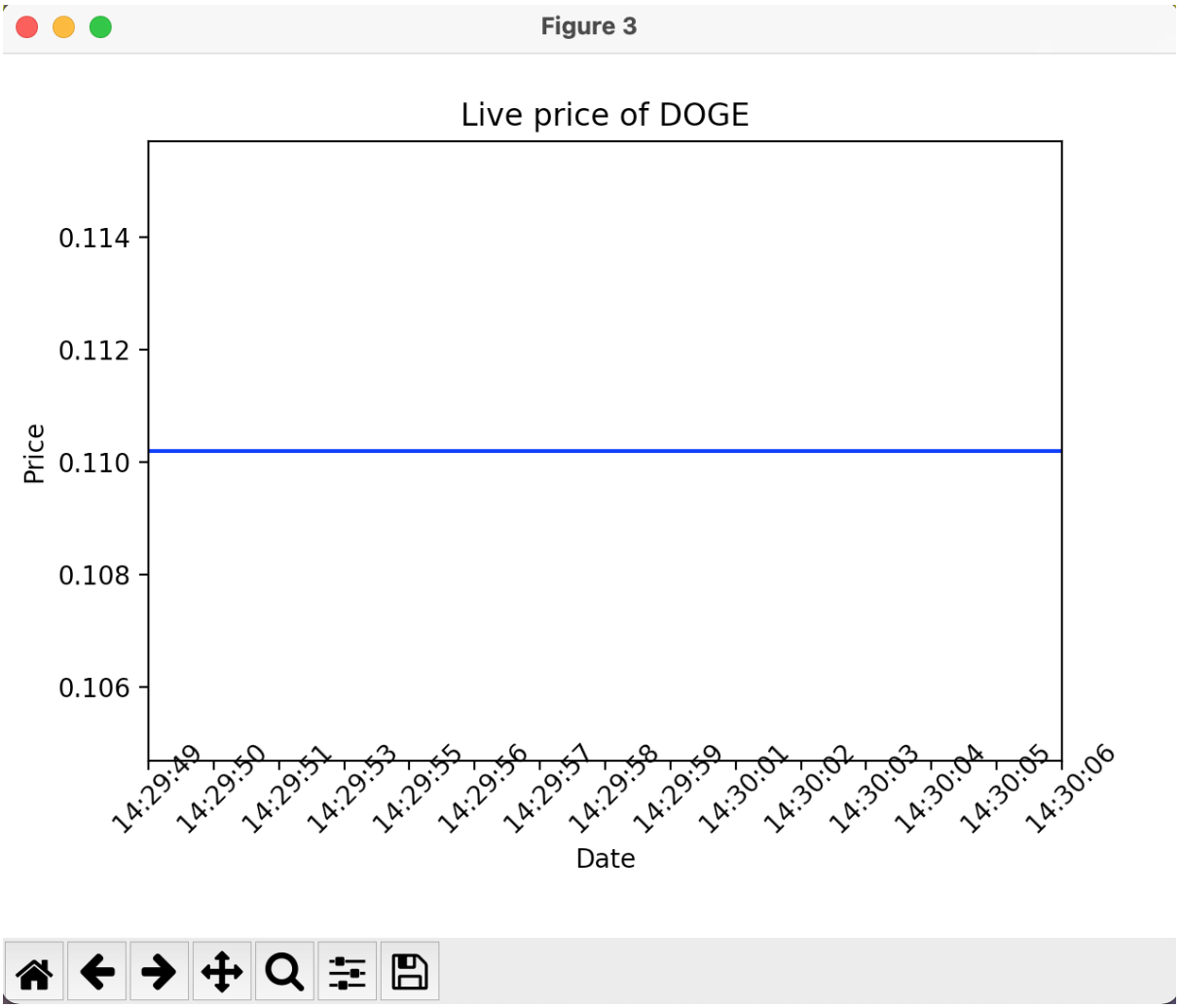
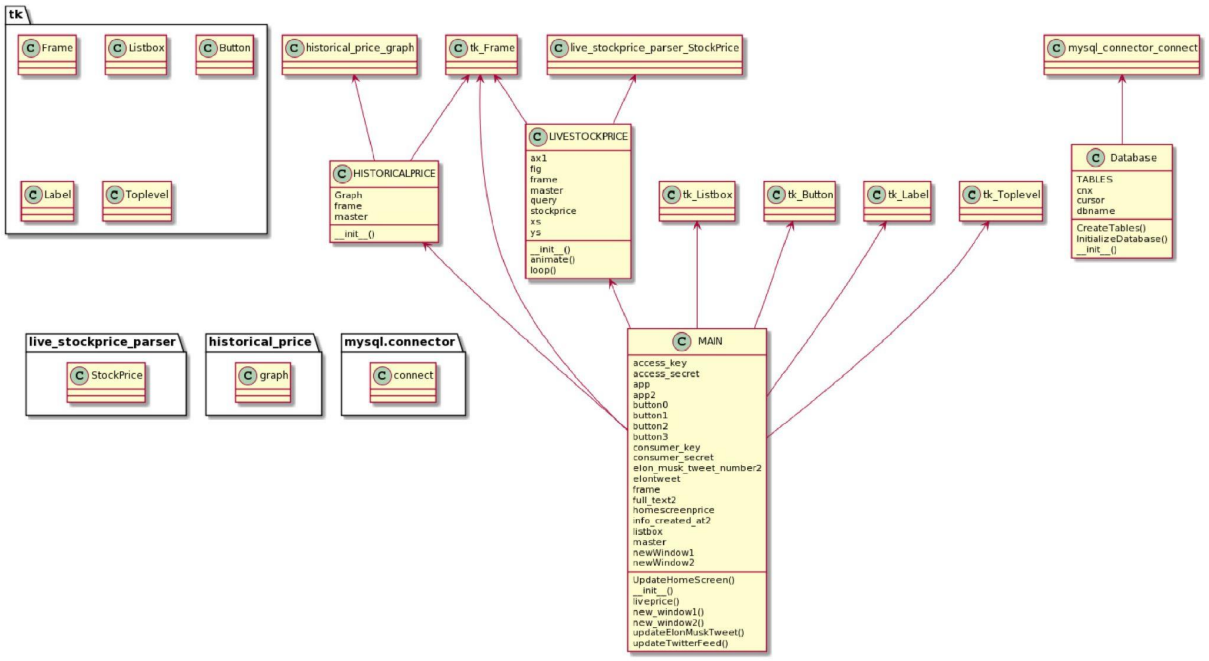


Figure 3
Live price of DOGE
Price
0.114
0.112
0.110
0.108
0.106
14:29:49
14:29:50
14:29:51
14:29:53
14:29:55
14:29:56
14:29:57
14:29:58
14:29:59
14:30:01
14:30:02
14:30:03
14:30:04
14:30:05
14:30:06
Date

Explanation of GUI

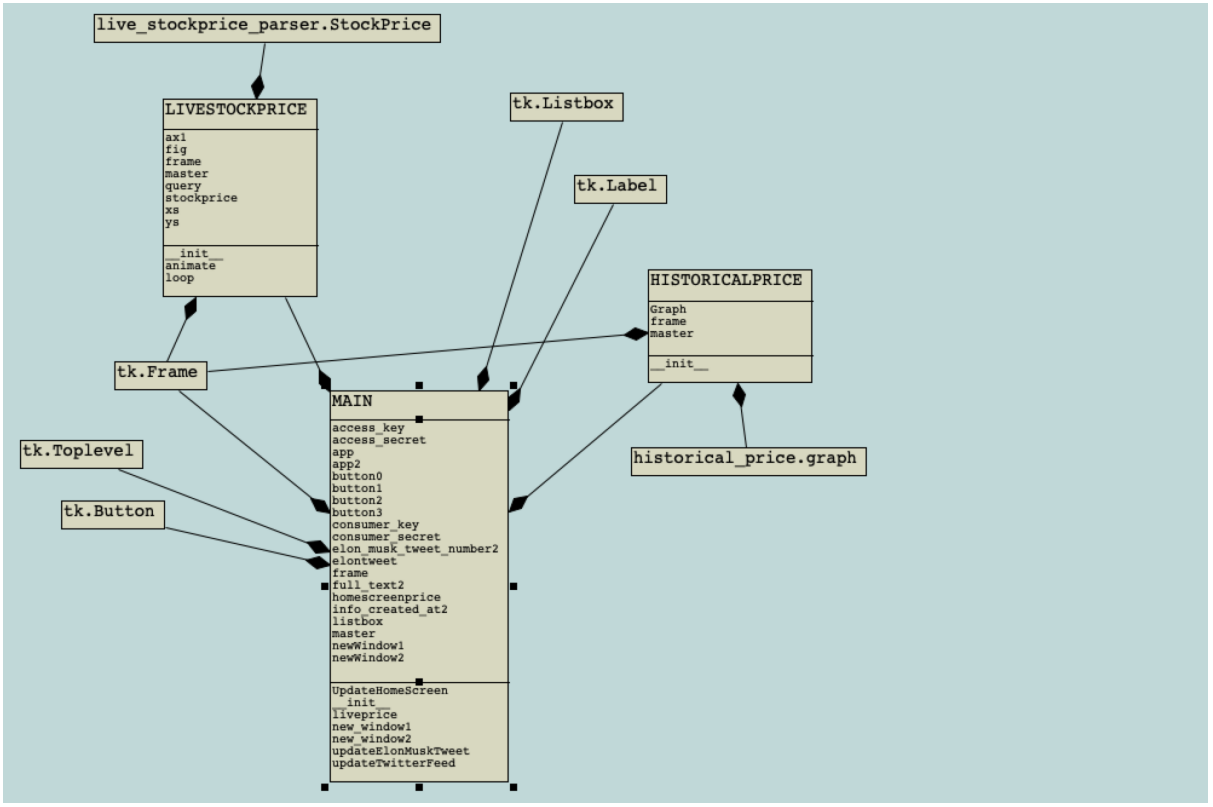
The main menu comprises a grey window, with a neutral background for ease of readability. At the top of the window, there are the tweets that Elon Musk made, and it updates in real time. Below this, there is a quick preview of the price of DOGE, which also updates real time. Below this, the initialize database button is present, which creates the tables and columns in the database. In the middle of the screen, there is the twitter feed which updates real time with tweets being made live about DOGE. The background is a dark grey which contrasts the yellow font for ease of readability whilst not being distracting as compared to the background colour. Below this, the button to display the historical price is present which displays the historical price of the stock over the past day. Below this, is the live price button, which displays the live price of the stock real time graphically as opposed to the price reading on the homescreen. Just below this, the home screen elements button is present. Clicking this, will turn on all the homescreen elements.

The live price and historical price windows are both a matplotlib window. They both have a white background, with a blue line in the middle of the graph. The legends on the side and the text is black to contrast the white background. There are buttons on the bottom. The home screen button resets the graph to default. The left and right buttons move the graph left and right respectively. The four arrow button allows you to move around the graph however you wish. The zoom button, allows you to select a section of the graph which gets zoomed into. The configuration button allows you to change the properties of the graph such as the axis size etc etc. The save button allows you to save the plot as a .png file.

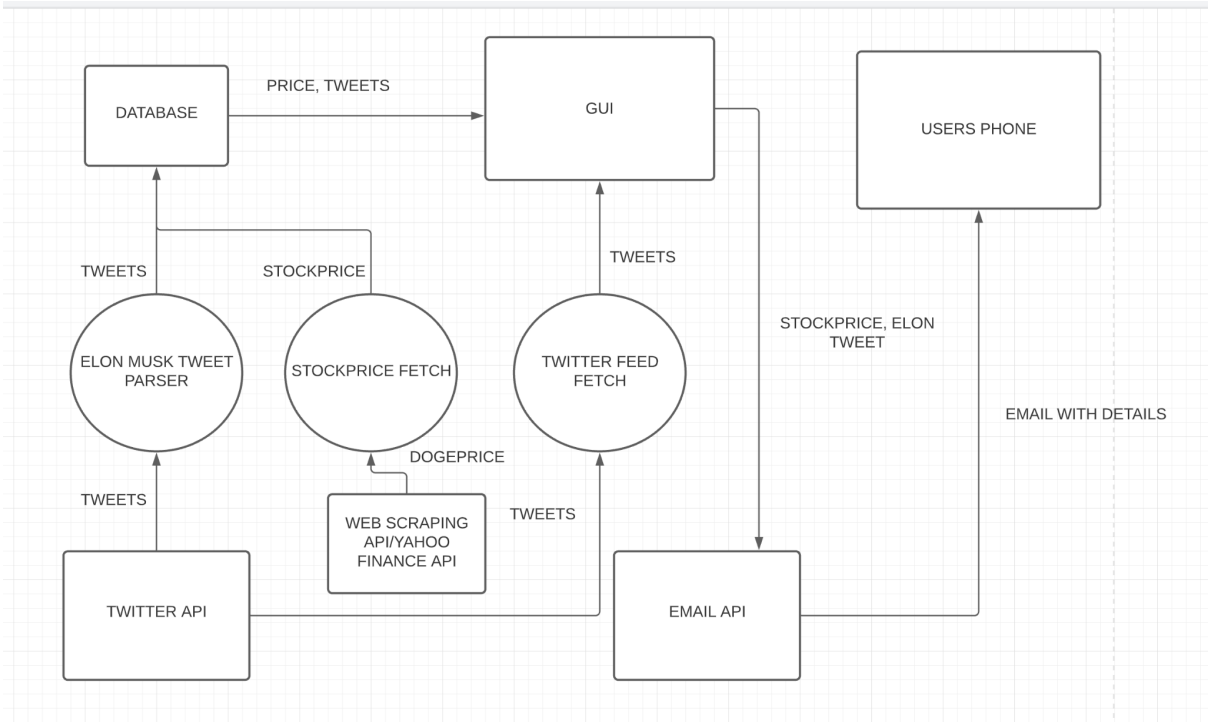


Class Diagram

Alternative Class Diagram



Data Flow Diagram



PSUEDOCODE OF KEY ALGORITHMS, (Explanations Above)

GUI MODULE

```
GET MODULE TKINTER AS TK
FROM TKINTER GET MODULE MAIN
GET MODULE TIME
GET MODULE MYSQL
GET MODULE MYSQL
GET MODULE DATABASE
GET MODULE EMAIL
GET MODULE ELON_MUSK_TWEET_PARSER
FROM TWITTERAPI GET MODULE TWITTERAPI
FROM MATPLOTLIB GET MODULE ANIMATION, PYPLOT AS PLT
GET MODULE HISTORICAL_PRICE
GET MODULE LIVE_STOCKPRICE_PARSER
GET MODULE THREADING
CLASS MAIN:
    FUNCTION INITIALIZE (SELF, MASTER):
        SELF.ELON_MUSK_TWEET_NUMBER2 ← NONE
        SELF.INFO_CREATED_AT2 ← NONE
        SELF.FULL_TEXT2 ← NONE

       ##THIS SECTION OF CODE BELOW, FINDS THE INITIAL TWEET ELON TWEETED, BY PULLING FROM THE DATABASE, IT USES THIS TO COMPARE IF ELON
TWEETED SOMETHING NEW.
        CNX ← MYSQL.CONNECTOR.CONNECT(USER←'username of user', PASSWD←'password of user', DATABASE←'INFORMATION')
        CURSOR ← CNX.CURSOR()
        CURSOR.EXECUTE("SELECT * FROM ELON_MUSK_TWEET ORDER BY ELON_MUSK_TWEET_NUMBER DESC LIMIT 1;")
        FOR (ELON_MUSK_TWEET_NUMBER, INFO_CREATED_AT, FULL_TEXT) IN CURSOR:
            SELF.ELON_MUSK_TWEET_NUMBER2 ← ELON_MUSK_TWEET_NUMBER
            SELF.INFO_CREATED_AT2 ← INFO_CREATED_AT
            SELF.FULL_TEXT2 ← FULL_TEXT
        CURSOR.CLOSE()
        CNX.CLOSE()
        ##THIS ENTIRE SECTION BELOW, CREATES THE WIDGETS AND ELEMENTS ON THE HOME SCREEN, AND 'PACKS' THEM INTO THE MAIN WINDOW.
        SELF.MASTER ← MASTER
        SELF.FRAME ← TK.FRAME(SELF.MASTER)
        SELF.BUTTON0 ← TK.BUTTON(SELF.FRAME, TEXT ← 'INITIALIZE DATABASE', WIDTH ← 25, COMMAND ← DATABASE.DATABASE)
        SELF.BUTTON1 ← TK.BUTTON(SELF.FRAME, TEXT ← 'HISTORICAL PRICE', WIDTH ← 25, COMMAND ← SELF.NEW_WINDOW1)
        SELF.BUTTON2 ← TK.BUTTON(SELF.FRAME, TEXT ← 'LIVE PRICE', WIDTH ← 25, COMMAND ← SELF.NEW_WINDOW2)
        SELF.BUTTON3 ← TK.BUTTON(SELF.FRAME, TEXT ← 'HOME SCREEN ELEMENTS', WIDTH ←25,
COMMAND←THREADING.THREAD(TARGET←SELF.UPDATEHOMESCREEN).START())
        SELF.LISTBOX ← TK.LISTBOX(SELF.FRAME, HEIGHT ← 20, WIDTH ← 150, BG ← "GREY", ACTIVESTYLE ← 'DOTBOX', FONT ← "HELVETICA", FG ← "YELLOW")
        SELF.HOMESCREENPRICE ← TK.LABEL(SELF.FRAME, TEXT ← "")
        SELF.ELONTWEET ← TK.LABEL(SELF.FRAME, TEXT ← "")
        SELF.ELONTWEET.PACK()
        SELF.HOMESCREENPRICE.PACK()
        SELF.BUTTON0.PACK()
        SELF.LISTBOX.PACK()
        SELF.BUTTON1.PACK()
        SELF.BUTTON2.PACK()
        SELF.BUTTON3.PACK()
        SELF.FRAME.PACK()

        ##THIS IS THE TWITTER CREDENTIALS OF THE FINAL USER.
        SELF.CONSUMER_KEY ← "userconsumerkey"
        SELF.CONSUMER_SECRET ← "userconsumersecret"
        SELF.ACCESS_KEY ← "useraccesskey"
        SELF.ACCESS_SECRET ← "useraccesssecret"


    FUNCTION UPDATETWITTERFEED(SELF):
        ##THIS METHOD OF THE CLASS, UPDATES THE TWITTER FEED
        TRY:
            ##THIS BLOCK OF CODE, CREATES A CONNECTION TO THE TWITTER API USING THE CREDENTIALS AND REQUESTS TWEETS ABOUT THE KEY WORD 'DOGE'.
            API ← TWITTERAPI(SELF.CONSUMER_KEY, SELF.CONSUMER_SECRET, SELF.ACCESS_KEY, SELF.ACCESS_SECRET)
            R ← API.REQUEST('SEARCH/TWEETS', {'Q':'DOGE'})
            FOR ITEM IN R:
                CREATED_AT2 ← STR(ITEM["CREATED_AT"])
                FULL_TEXT2 ← STR(ITEM["TEXT"])
                FORMATTED_STRING ← "CREATED AT: {CREATED_AT2}, TEXT: {FULL_TEXT2}".FORMAT(CREATED_AT2←CREATED_AT2, FULL_TEXT2←FULL_TEXT2)
                SELF.LISTBOX.INSERT('END', FORMATTED_STRING) ##THIS INSERTS THE FOUND TWEETS INTO THE BOTTOM OF THE LISTBOX PRESENT IN THE MAIN
MENU.
        EXCEPT:
            PRINT("THE TWITTER FEED CANNOT BE UPDATED")

    FUNCTION UPDATEELONMUSKTWEET(SELF):
        ##THIS SECTION OF CODE, UPDATES THE ELON MUSK TWEET LABEL ON THE HOMESCREEN, AS WELL AS THIS, IT CALLS THE EMAIL METHOD TO ALERT THE
USER IF ELON MUSK HAS TWEETED.
        TRY:
            ##INSTANTIATING AN ELONMUSKLISTENER CLASS TO GET THE TWEETS
            ELONMUSKLISTENER ← ELON_MUSK_TWEET_PARSER.ELONMUSKLISTENER()
            ELONMUSKLISTENER.GETTWEET()
            PRINT("TWEET FETCHED AND INSERTED INTO DATABASE")
            DEL ELONMUSKLISTENER
            CNX2 ← MYSQL.CONNECTOR.CONNECT(USER←'ELONMUSKTWEETREADER', PASSWD←'KK123459!', DATABASE←'INFORMATION')
            CURSOR2 ← CNX2.CURSOR()
            CURSOR2.EXECUTE("SELECT * FROM ELON_MUSK_TWEET ORDER BY ELON_MUSK_TWEET_NUMBER DESC LIMIT 1;")
            ##GETS THE LATEST TWEET FROM THE DATABASE
            FOR (ELON_MUSK_TWEET_NUMBER, INFO_CREATED_AT, FULL_TEXT) IN CURSOR2:
                FULL_TEXT2 ← FULL_TEXT
                INFO_CREATED_AT2 ← INFO_CREATED_AT
            CURSOR2.CLOSE()
            CNX2.CLOSE()
            ##THIS CHECKS IF THE TWEET MADE IS ACTUALLY NEW, AND THEN REPLACES THE TWEET STORED IN MEMORY AND SENDS AN EMAIL TO THE FINAL USER.
            IF FULL_TEXT2 ←← SELF.FULL_TEXT2:
                CONCATENATED_TEXT ← "ELON TWEETED {FULL_TEXT2}, AT {INFO_CREATED_AT2}".FORMAT(FULL_TEXT2←FULL_TEXT2,
INFO_CREATED_AT2←INFO_CREATED_AT2)
                SELF.ELONTWEET.CONFIGURE(TEXT ← CONCATENATED_TEXT)
                SELF.FULL_TEXT2 ← FULL_TEXT2
            ELIF FULL_TEXT2 !← SELF.FULL_TEXT2:
                CONCATENATED_TEXT ← "ELON TWEETED {FULL_TEXT2}, AT {INFO_CREATED_AT2}".FORMAT(FULL_TEXT2←FULL_TEXT2,
INFO_CREATED_AT2←INFO_CREATED_AT2)
```

```
                SELF.ELONTWEET.CONFIGURE(TEXT ← CONCATENATED_TEXT)
                EMAILSERVICE.SENDEMAIL()
                SELF.FULL_TEXT2 ← FULL_TEXT2
        EXCEPT:
            PRINT("THE TWEET COULD NOT BE FETCHED FROM THE DATABSE, PLEASE ENSURE MYSQL IS ENABLED AND IS ONLINE. AND THE CREDENTIALS ARE
CORRECT.")

    FUNCTION LIVEPRICE(SELF):
        TRY:
            ##THIS METHOD, UPDATES THE PRICE LABEL ON THE MAIN MENU WITH THE CURRENT PRICE AS WELL AS THE TIME.
            STOCKPRICEPARSER ← LIVE_STOCKPRICE_PARSER.STOCKPRICE('HTTPS://WWW.GOOGLE.COM/FINANCE/QUOTE/DOGE-GBP', [redacted])
            ##CALLS THE METHOD TO GET THE LATEST STOCKPRICE AND INSERT IT INTO THE DATABASE
            STOCKPRICEPARSER.PRICELOOP()
            STOCK_NUMBER2 ← NONE
            TIME2 ← NONE
            PRICE2 ← NONE
            CNX ← MYSQL.CONNECTOR.CONNECT(USER←'STOCKPRICEREADER', PASSWD←'KK123459!', DATABASE←'INFORMATION')
            CURSOR ← CNX.CURSOR()
            CURSOR.EXECUTE("SELECT * FROM STOCKPRICE ORDER BY STOCK_NUMBER DESC LIMIT 1;")
            FOR (STOCK_NUMBER, TIME, PRICE) IN CURSOR:
                STOCK_NUMBER2 ← STOCK_NUMBER
                TIME2 ← TIME
                PRICE2 ← PRICE
            CURSOR.CLOSE()
            CNX.CLOSE()
            CONCATENATED_STRING ← "PRICE ← {PRICE2}    TIME ← {TIME2}".FORMAT(PRICE2 ← PRICE2, TIME2 ← TIME2)
            SELF.HOMESCREENPRICE.CONFIGURE(TEXT ← CONCATENATED_STRING)
            ##UPDATES THE HOMESCREEN WITH THE CURRENT PRICE
        EXCEPT:
            PRINT("THE LIVE PRICE COULD NOT BE FETCHED FROM THE DATABASE, PLEASE ENSURE THAT MYSQL IS ENABLED AND IS ONLINE. AND THE
CREDENTIALS ARE CORRECT.")

    FUNCTION UPDATEHOMESCREEN(SELF):
        TRY:
            ##THIS METHOD, RUNS ALL THE SUBMETHODS TO ENSURE THAT THE HOME MENU UPDATES, IT DOES THIS IN A NEW THREAD TO PREVENT SLOWDOWN.
            WHILE(TRUE):
                TIME.SLEEP(5)
                SELF.UPDATETWITTERFEED()
                SELF.UPDATEELONMUSKTWEET()
                SELF.LIVEPRICE()
        EXCEPT:
            PRINT("THE MAIN FUNCTIONS TO UPDATE THE HOMESCREEN ARE NOT FUNCTIONAL, PLEASE RESTART THE PROGRAM OR REVERT IT TO A PRIOR STATE AND
RE-ENTER YOUR CREDENTIALS.")

    FUNCTION NEW_WINDOW1(SELF):
        ##THIS METHOD, CREATES A NEW CLASS DYNAMICALLY WHEN THE BUTTON CORRESPONDING TO THIS BUTTON IS PRESSED.
        TRY:
            SELF.NEWWINDOW1 ← TK.TOPLEVEL(SELF.MASTER)
            SELF.APP ← HISTORICALPRICE(SELF.NEWWINDOW1)
        EXCEPT:
            PRINT("THE HISTORICAL PRICE WINDOW COULD NOT BE OPENED, PLEASE REVERT THE PROGRAM TO ITS ORIGINAL STATE.")

    FUNCTION NEW_WINDOW2(SELF):
        ##THIS METHOD, CREATES A NEW CLASS DYNAMICALLY WHEN THE BUTTON CORRESPONDING TO THIS BUTTON IS PRESSED.
        TRY:
            SELF.NEWWINDOW2 ← TK.TOPLEVEL(SELF.MASTER)
            SELF.APP2 ← LIVESTOCKPRICE(SELF.NEWWINDOW2)
        EXCEPT:
            PRINT("THE LIVE STOCKPRICE WINDOW CANNOT BE OPENED, PLEASE REVERT THE PROGRAM TO ITS ORIGINAL STATE")

##THESE CLASSES, ARE GENERATED DYNAMICALLY, DEPENDING ON WHICH BUTTON ON THE HOMESCREEN IS CLICKED AND THEN DESTROYED WHEN THE WINDOW IS
CLOSED.
CLASS HISTORICALPRICE:
    FUNCTION INITIALIZE (SELF, MASTER):
        SELF.MASTER ← MASTER
        SELF.FRAME ← TK.FRAME(SELF.MASTER)
        SELF.GRAPH ← HISTORICAL_PRICE.GRAPH('DOGE-USD','DOGE PRICE','TIME','PRICE $')
        TRY:
            SELF.GRAPH.INITLIALIZEGRAPH()
        EXCEPT:
            PRINT("THE GRAPH COULD NOT BE CREATED UNFORTUNATELY, PLEASE REVERT THE CODE TO ITS ORIGINAL FORM")
        SELF.MASTER.DESTROY()
        ##THIS METHOD, GENERATES THE HISTORICAL PRICE GRAPH BY CALLING THE METHOD IN THE GRAPH CLASS.

CLASS LIVESTOCKPRICE:
     FUNCTION INITIALIZE (SELF, MASTER):
         SELF.MASTER ← MASTER
         SELF.FRAME ← TK.FRAME(SELF.MASTER)
         SELF.QUERY ← ("SELECT * FROM STOCKPRICE ORDER BY STOCK_NUMBER DESC LIMIT 1;")
         SELF.STOCKPRICE ← LIVE_STOCKPRICE_PARSER.STOCKPRICE('HTTPS://WWW.GOOGLE.COM/FINANCE/QUOTE/DOGE-GBP',
                                                             'YMLKEC FXKBKC')
         SELF.STOCKPRICE.INITIALIZE()
         SELF.FIG ← PLT.FIGURE()
         SELF.AX1 ← SELF.FIG.ADD_SUBPLOT(1, 1, 1)
         SELF.XS ← []
         SELF.YS ← []
         SELF.LOOP()

     FUNCTION ANIMATE(SELF, I):
         ##THIS ANIMATION LOOP IS CALLED IN THE SPECIAL FUNCTION CALLED ANIMATION BUILT INTO THE MATPLOTLIB PACKAGE WHICH IT UTILIZES FOR
ANIMATING THE PLOT.
         TRY:
             ##THIS FETCHES THE LATEST PRICES AND INSERTS THEM INTO THE DATABASE
             SELF.STOCKPRICE.PRICELOOP()
             CNX ← MYSQL.CONNECTOR.CONNECT(USER←[redacted], PASSWD←'[redacted]', DATABASE←'INFORMATION')
             CURSOR ← CNX.CURSOR()
             CURSOR.EXECUTE(SELF.QUERY)
             FOR (STOCK_NUMBER, TIME, PRICE) IN CURSOR:
                 TIME2 ← TIME
                 PRICE2 ← PRICE
            ##THE CODE ABOVE FETCHES THE PRICE FROM THE DATABASE.
             SELF.XS.APPEND(STR(TIME2))
             SELF.YS.APPEND(PRICE2)
             PRINT("TIME IS,", STR(TIME2))
             PRINT("PRICE IS,", PRICE2)
             SELF.AX1.CLEAR()
             SELF.AX1.AUTOSCALE(ENABLE←TRUE, AXIS←'BOTH', TIGHT←TRUE)
```

```
            SELF.AX1.PLOT(SELF.XS, SELF.YS, SCALEY←TRUE, SCALEX←TRUE, COLOR←"BLUE")
            PLT.XTICKS(ROTATION←45, ROTATION_MODE←"ANCHOR")
            PLT.SUBPLOTS_ADJUST(BOTTOM←0.2, TOP←0.9)
            PLT.XLABEL('DATE')
            PLT.YLABEL('PRICE')
            PLT.TITLE('LIVE PRICE OF DOGE')
            CURSOR.CLOSE()
            CNX.CLOSE()
            SELF.MASTER.DESTROY()
        EXCEPT:
            PRINT("THE LIVE GRAPH CANNOT BE CREATED, PLEASE ENSURE THAT ALL THE MODULES ARE INSTALLED, PLEASE REVERT THE CODE TO ITS ORIGINAL
FORM")

    FUNCTION LOOP(SELF):
        ##BELOW IS THE BUILT IN ANIMATION FUNCTION IN THE MATPLOTLIB PACKAGE.
        ANI ← ANIMATION.FUNCANIMATION(SELF.FIG, SELF.ANIMATE, INTERVAL←1000)
        PLT.SHOW()

FUNCTION MAIN():
    TRY:
        ##A TKINTER INSTANCE IS CREATED, AND A TARGET RESOLUTION IS SET WHICH CAN BE ADJUSTED BELOW.
        ROOT ← TK.TK()
        ROOT.GEOMETRY("1280X720")
        APP ← MAIN(ROOT)
        ROOT.MAINLOOP()
    EXCEPT:
        PRINT("CATASTROPHIC ERROR PLEASE REVERT CODE TO ITS ORIGINAL FORM")
IF __NAME__ ←← '__MAIN__':
    MAIN()
```

DATABASE MODULE

```
FROM __FUTURE__ GET PRINT_FUNCTION
GET MYSQL.CONNECTOR
FROM MYSQL.CONNECTOR GET ERRORCODE

CLASS DATABASE:
    FUNCTION INITIALIZE(SELF):
        SELF.DBNAME ← 'INFORMATION'
        SELF.TABLES ← {}
        SELF.CNX ← MYSQL.CONNECTOR.CONNECT(USER='ROOT', PASSWD='KK123459!')
        SELF.CURSOR ← SELF.CNX.CURSOR()
        SELF.CREATETABLES()
        SELF.INITIALIZEDATABASE()

    FUNCTION CREATETABLES(SELF):

        SELF.TABLES['STOCKPRICE'] ← (
            "CREATE TABLE `STOCKPRICE` ("
            "  `STOCK_NUMBER` INT(6) NOT NULL AUTO_INCREMENT,"
            "  `TIME` TIME NOT NULL,"
            "  `PRICE` FLOAT(64) NOT NULL,"
            "  PRIMARY KEY (`STOCK_NUMBER`)"
            ") ENGINE←INNODB")

        SELF.TABLES['ELON_MUSK_TWEET'] ← (
            "CREATE TABLE `ELON_MUSK_TWEET` ("
            "  `ELON_MUSK_TWEET_NUMBER` INT(6) NOT NULL AUTO_INCREMENT,"
            "  `INFO_CREATED_AT` CHAR(100) NOT NULL,"
            "  `FULL_TEXT` VARCHAR(500) NOT NULL,"
            "  PRIMARY KEY (`ELON_MUSK_TWEET_NUMBER`)"
            ") ENGINE←INNODB")


    FUNCTION INITIALIZEDATABASE(SELF):
        TRY:
            SELF.CURSOR.EXECUTE(
                "CREATE DATABASE {} DEFAULT CHARACTER SET 'UTF8'".FORMAT(SELF.DBNAME))
        EXCEPT MYSQL.CONNECTOR.ERROR AS ERR:
            PRINT("FAILED CREATING DATABASE: {}".FORMAT(ERR))
            EXIT(1)

        TRY:
            SELF.CURSOR.EXECUTE("USE {}".FORMAT(SELF.DBNAME))
        EXCEPT MYSQL.CONNECTOR.ERROR AS ERR:
            PRINT("DATABASE {} DOES NOT EXISTS.".FORMAT(SELF.DBNAME))
            IF ERR.ERRNO == ERRORCODE.ER_BAD_DB_ERROR:
                SELF.CREATE_DATABASE(SELF.CURSOR)
                PRINT("DATABASE {} CREATED SUCCESSFULLY.".FORMAT(SELF.DBNAME))
                SELF.CNX.DATABASE ← SELF.DBNAME
            ELSE:
                PRINT(ERR)
                EXIT(1)

        FOR TABLE_NAME IN SELF.TABLES:
            TABLE_DESCRIPTION ← SELF.TABLES[TABLE_NAME]
            TRY:
                PRINT("CREATING TABLE {}: ".FORMAT(TABLE_NAME), END='')
                SELF.CURSOR.EXECUTE(TABLE_DESCRIPTION)
            EXCEPT MYSQL.CONNECTOR.ERROR AS ERR:
                IF ERR.ERRNO == ERRORCODE.ER_TABLE_EXISTS_ERROR:
                    PRINT("ALREADY EXISTS.")
                ELSE:
                    PRINT(ERR.MSG)
            ELSE:
                PRINT("OK")

        SELF.CURSOR.CLOSE()
        SELF.CNX.CLOSE()
```

ELON MUSK TWEET PARSER MODULE

```
FROM __FUTURE__ GET PRINT_FUNCTION
GET TIME
GET SCHEDULE
GET TWEEPY
FROM DATETIME GET DATE, DATETIME, TIMEDELTA
GET MYSQL.CONNECTOR
CLASS ELONMUSKLISTENER:
    FUNCTION INITIALIZE(SELF):
        SELF.CNX ← MYSQL.CONNECTOR.CONNECT(USER←[redacted]'PASSWD←[redacted]DATABASE←"INFORMATION")
        SELF.CURSOR ← SELF.CNX.CURSOR()
        SELF.USER ← "ELONMUSK"
        SUPER(ELONMUSKLISTENER, SELF).__INIT__()
        SELF.TEXT ← ""
        SELF.CONSUMER_KEY ← [redacted]
        SELF.CONSUMER_SECRET ← [redacted]
        SELF.ACCESS_KEY ← [redacted]
        SELF.ACCESS_SECRET ← [redacted]

        SELF.ADD_DATA ← ("INSERT INTO ELON_MUSK_TWEET "
                    "(INFO_CREATED_AT, FULL_TEXT) "
                    "VALUES (%(INFO_CREATED_AT)S, %(FULL_TEXT)S)")

    FUNCTION GETTWEET(SELF):
        AUTH ← TWEEPY.OAUTHHANDLER(SELF.CONSUMER_KEY, SELF.CONSUMER_SECRET)
        AUTH.SET_ACCESS_TOKEN(SELF.ACCESS_KEY, SELF.ACCESS_SECRET)
        API ← TWEEPY.API(AUTH)
        TWEETS ← API.USER_TIMELINE(SCREEN_NAME←SELF.USER,COUNT←200,INCLUDE_RTS←FALSE,TWEET_MODE←'EXTENDED')
        FOR INFO IN TWEETS[:1]:
            ENCODED_STRING ← INFO.FULL_TEXT.ENCODE("ASCII", "IGNORE")
            DECODE_STRING ← ENCODED_STRING.DECODE()
            SELF.TEXT ← DECODE_STRING
            DATA_ENTRY ← {

                'INFO_CREATED_AT': INFO.CREATED_AT,
                'FULL_TEXT': SELF.TEXT,
            }
            SELF.CURSOR.EXECUTE(SELF.ADD_DATA, DATA_ENTRY)
            SELF.CNX.COMMIT()
            PRINT(INFO.CREATED_AT)
            PRINT(INFO.FULL_TEXT)
            PRINT("\N")
```

EMAIL SERVICE MODULE

```
GET SENDGRID
GET MYSQL.CONNECTOR
GET OS
FROM SENDGRID.HELPERS.MAIL GET MAIL, EMAIL, TO, CONTENT

FUNCTION  SENDEMAIL():
    CNX ← MYSQL.CONNECTOR.CONNECT(USER←'username', PASSWD←'password', DATABASE←'INFORMATION')
    CURSOR ← CNX.CURSOR()
    CURSOR.EXECUTE("SELECT * FROM STOCKPRICE ORDER BY STOCK_NUMBER DESC LIMIT 1;")
    FOR (STOCK_NUMBER, TIME, PRICE) IN CURSOR:
        STOCK_NUMBER2 ← STOCK_NUMBER
        TIME2 ← TIMES
        PRICE2 ← PRICE
    CURSOR.CLOSE()
    CNX.CLOSE()
    MY_SG ← SENDGRID.SENDGRIDAPICLIENT('SG[redacted])
    FROM_EMAIL ← EMAIL([redacted])
    TO_EMAIL ← TO[redacted])
    SUBJECT ← "ELON MUSK HAS TWEETED"
    STR_PRICE ← STR(PRICE2)
    STRING_FORMATTED ← "THE PRICE OF DOGE IS {}".FORMAT(STR_PRICE)
    CONTENT ← CONTENT("TEXT/PLAIN", STRING_FORMATTED)
    MAIL ← MAIL(FROM_EMAIL, TO_EMAIL, SUBJECT, CONTENT)
    MAIL_JSON ← MAIL.GET()
    RESPONSE ← MY_SG.CLIENT.MAIL.SEND.POST(REQUEST_BODY←MAIL_JSON)
```

LIVE STOCKPRICE PARSER MODULE:

```
GET REQUESTS
FROM BS4 GET BEAUTIFULSOUP
GET TIME
GET SCHEDULE
GET THREADING
GET MYSQL.CONNECTOR
GET MATPLOTLIB.PYPLOT AS PLT
GET MYSQL.CONNECTOR
GET _THREAD



CLASS STOCKPRICE:

    FUNCTION INITIALISE(SELF,URL,CLASS)
        SELF.FIG ← PLT.FIGURE()
        SELF.CNX ← MYSQL.CONNECTOR.CONNECT(USER←[redacted], PASSWD←'[redacted], DATABASE←[redacted]
        SELF.CURSOR ← SELF.CNX.CURSOR()
        SELF.COUNT ← 0
        SELF.URL ← URL
        SELF.PAGE ← NONE
        SELF.SOUP ← NONE
        SELF.PRICE ← 0.00
        SELF.CLASS ← CLASS
        SELF.QUERY_TEMPLATE ← ("INSERT INTO STOCKPRICE "
                               "(TIME, PRICE) "
                               "VALUES (%(CURRENT_TIME)S, %(NEWSTR5)S)")
        SELF.INITIALIZE()

    FUNCTION INITIALIZE(SELF):
        SELF.PAGE ← REQUESTS.GET(SELF.URL)
        SELF.SOUP ← BEAUTIFULSOUP(SELF.PAGE.TEXT, 'LXML')

    FUNCTION PARSEPRICE(SELF):
        SELF.INITIALIZE()
        PRICE ← SELF.SOUP.FIND('DIV', CLASS_←SELF.CLASS).TEXT
        RETURN PRICE

    FUNCTION PRICELOOP(SELF):
            VALUE ← SELF.PARSEPRICE()
            PRINT("VALUE PARSED IS NOW:", VALUE)
            T ← TIME.LOCALTIME()
            CURRENT_TIME ← TIME.STRFTIME("%H:%M:%S", T)
            COMPOSED ← STR(VALUE)
            NEWSTR ← COMPOSED.REPLACE("'", "")
            NEWSTR2 ← NEWSTR.REPLACE(" ","")
            NEWSTR3 ← NEWSTR2.REPLACE("$", "")
            NEWSTR4 ← NEWSTR3.REPLACE("(", "")
            NEWSTR5 ← NEWSTR4.REPLACE(")", "")
            DATA_ENTRY ← {

                'CURRENT_TIME': CURRENT_TIME,
                'NEWSTR5': STR(NEWSTR5),
            }
            PRINT("TIME INSERTED", CURRENT_TIME)
            PRINT("PRICE INSERTED", NEWSTR5)

            SELF.CURSOR.EXECUTE(SELF.QUERY_TEMPLATE, DATA_ENTRY)
            SELF.CNX.COMMIT()
```

HISTORICAL PRICE MODULE

```
GET YFINANCE AS YF
GET MATPLOTLIB.PYPLOT AS PLT
FROM DATETIME GET DATE, TIMEDELTA

CLASS GRAPH():
    INITIALIZE(SELF,TAG,TITLE,X_AXIS,Y_AXIS):
        SELF.TITLE ← TITLE
        SELF.X_AXIS ← X_AXIS
        SELF.Y_AXIS ← Y_AXIS
        SELF.TAG ← TAG
        SELF.START_DATE ← NONE
        SELF.END_DATE ← NONE
        SELF.DATA ← []
        SELF.DOWNLOADSTOCKDATA()
        SELF.PLT ← PLT


    FUNCTION DOWNLOADSTOCKDATA(SELF):
        SELF.START_DATE ← DATE.TODAY() - TIMEDELTA(DAYS←7)
        SELF.END_DATE ← DATE.TODAY()
        SELF.DATA ← YF.DOWNLOAD(SELF.TAG, STR(SELF.START_DATE), STR(SELF.END_DATE))
        PRINT("HELLO", SELF.DATA['ADJ CLOSE'][0])

    FUNCTION INITLIALIZEGRAPH(SELF):
        SELF.PLT.FIGURE(FIGSIZE← (20,10))
        SELF.PLT.TITLE('OPENING PRICES OF {} FROM {} TO {}'.FORMAT(SELF.TAG,SELF.START_DATE,SELF.END_DATE))
        SELF.PLT.PLOT(SELF.DATA['ADJ CLOSE'])
        SELF.PLT.SHOW()
```

```
IF NAME == "MAIN":
    GRAPH ← GRAPH('DOGE-USD','DOGE PRICE','TIME','PRICE $')
    GRAPH.INITLIALIZEGRAPH()
```

Code Screenshot

MAIN MODULE

Title

```python
import tkinter as tk
from tkinter.tix import MAIN
import time
import mysql.connector
import mysql
import DATABASE
import EmailService
import elon_musk_tweet_parser
from TwitterAPI import TwitterAPI
from matplotlib import animation, pyplot as plt
import historical_price
import live_stockprice_parser
import threading
class MAIN:
    def __init__(self, master):
        self.elon_musk_tweet_number2 = None
        self.info_created_at2 = None
        self.full_text2 = None

        ##This section of code below, finds the initial tweet Elon tweeted, by pulling from the database, it uses this to compare if Elon Tweeted something new.
        cnx = mysql.connector.connect(user='elonmusktweetReader', passwd=[redacted], database='[redacted]')
        cursor = cnx.cursor()
        cursor.execute("SELECT * FROM elon_musk_tweet ORDER BY elon_musk_tweet_number DESC LIMIT 1;")
        for (elon_musk_tweet_number, info_created_at, full_text) in cursor:
            self.elon_musk_tweet_number2 = elon_musk_tweet_number
            self.info_created_at2 = info_created_at
            self.full_text2 = full_text
        cursor.close()
        cnx.close()
        ##This entire section below, creates the widgets and elements on the home screen, and 'packs' them into the main window.
        self.master = master
        self.frame = tk.Frame(self.master)
        self.button0 = tk.Button(self.frame, text = 'Initialize Database', width = 25, command = DATABASE.Database)
        self.button1 = tk.Button(self.frame, text = 'Historical Price', width = 25, command = self.new_window1)
        self.button2 = tk.Button(self.frame, text = 'Live Price', width = 25, command = self.new_window2)
        self.button3 = tk.Button(self.frame, text = 'HOME SCREEN ELEMENTS', width =25, command = threading.Thread(target=self.UpdateHomeScreen).start())
        self.listbox = tk.Listbox(self.frame, height = 20, width = 150, bg = "grey", activestyle = 'dotbox', font = "Helvetica", fg = "yellow")
        self.homescreenprice = tk.Label(self.frame, text = "")
        self.elontweet = tk.Label(self.frame, text = "")
        self.elontweet.pack()
        self.homescreenprice.pack()
        self.button0.pack()
        self.listbox.pack()
        self.button1.pack()
        self.button2.pack()
        self.button3.pack()
        self.frame.pack()

        ##This is the twitter credentials of the final user.
        self.consumer_key = [redacted]
        self.consumer_secret = "8[redacted]
        self.access_key = "12[redacted]
        self.access_secret = "BC[redacted]


    def updateTwitterFeed(self):
        ##This method of the class, updates the twitter feed
        try:
            ##This block of code, creates a connection to the Twitter API using the credentials and requests tweets about the key word 'Doge'.
            api = TwitterAPI(self.consumer_key, self.consumer_secret, self.access_key, self.access_secret)
            r = api.request('search/tweets', {'q':'DOGE'})
            for item in r:
                created_at2 = str(item["created_at"])
                full_text2 = str(item["text"])
                formatted_string = "CREATED AT: {created_at2}, TEXT: {full_text2}".format(created_at2=created_at2, full_text2=full_text2)
                self.listbox.insert('end', formatted_string) ##This inserts the found tweets into the bottom of the listbox present in the main menu.
        except:
            print("The Twitter feed cannot be updated")

    def updateElonMuskTweet(self):
        ##This section of code, updates the elon musk tweet label on the homescreen, as well as this, it calls the email method to alert the user if Elon Musk has tweeted.
        #try:
            ##Instantiating an ElonMuskListener class to get the tweets
            elonmusklistener = elon_musk_tweet_parser.ElonMuskListener()
            elonmusklistener.getTweet()
```

```
            print("Tweet Fetched and inserted into database")
            del elonmusklistener
            cnx2 = mysql.connector.connect(user='elonmusktweetReader', passwd='Kk123459!', database='Information')
            cursor2 = cnx2.cursor()
            cursor2.execute("SELECT * FROM elon_musk_tweet ORDER BY elon_musk_tweet_number DESC LIMIT 1;")
            ##Gets the latest tweet from the database
            for (elon_musk_tweet_number, info_created_at, full_text) in cursor2:
                full_text2 = full_text
                info_created_at2 = info_created_at
            cursor2.close()
            cnx2.close()
            ##This checks if the tweet made is actually new, and then replaces the tweet stored in memory and sends an email to the final user.
            if full_text2 == self.full_text2:
                concatenated_text = "Elon Tweeted {full_text2}, at {info_created_at2}".format(full_text2=full_text2,
info_created_at2=info_created_at2)
                self.elontweet.configure(text = concatenated_text)
                self.full_text2 = full_text2
            elif full_text2 != self.full_text2:
                print("New Tweet")
                concatenated_text = "Elon Tweeted {full_text2}, at {info_created_at2}".format(full_text2=full_text2,
info_created_at2=info_created_at2)
                self.elontweet.configure(text = concatenated_text)
                EmailService.sendemail()
                self.full_text2 = full_text2
        #except:
            #print("The tweet could not be fetched from the databse, please ensure MYSQL is enabled and is online. And the credentials are
correct.")

    def liveprice(self):
        try:
            ##This method, updates the price label on the main menu with the current price as well as the time.
            stockpriceparser = live_stockprice_parser.StockPrice('https://www.google.com/finance/quote/DOGE-GBP', 'YMlKec fxKbKc')
            ##Calls the method to get the latest stockprice and insert it into the database
            stockpriceparser.priceLoop()
            stock_number2 = None
            time2 = None
            price2 = None
            cnx = mysql.connector.connect(user='StockPriceReader', passwd='Kk123459!', database='Information')
            cursor = cnx.cursor()
            cursor.execute("SELECT * FROM Stockprice ORDER BY stock_number DESC LIMIT 1;")
            for (stock_number, time, price) in cursor:
                stock_number2 = stock_number
                time2 = time
                price2 = price
            cursor.close()
            cnx.close()
            concatenated_string = "Price = {price2}    Time = {time2}".format(price2 = price2, time2 = time2)
            self.homescreenprice.configure(text = concatenated_string)
            ##Updates the homescreen with the current price
        except:
            print("The live price could not be fetched from the database, please ensure that MYSQL is enabled and is online. And the
credentials are correct.")

    def UpdateHomeScreen(self):
        try:
            ##This method, runs all the submethods to ensure that the home menu updates, it does this in a new thread to prevent slowdown.
            while(True):
                time.sleep(5)
                self.updateTwitterFeed()
                self.updateElonMuskTweet()
                self.liveprice()
        except:
            print("The main functions to update the homescreen are not functional, please restart the program or revert it to a prior state and
re-enter your credentials.")

    def new_window1(self):
        ##This method, creates a new class dynamically when the button corresponding to this button is pressed.
        try:
            self.newWindow1 = tk.Toplevel(self.master)
            self.app = HISTORICALPRICE(self.newWindow1)
        except:
            print("The historical price window could not be opened, please revert the program to its original state.")

    def new_window2(self):
        ##This method, creates a new class dynamically when the button corresponding to this button is pressed.
        try:
            self.newWindow2 = tk.Toplevel(self.master)
            self.app2 = LIVESTOCKPRICE(self.newWindow2)
        except:
            print("The live stockprice window cannot be opened, please revert the program to its original state")

##These classes, are generated dynamically, depending on which button on the homescreen is clicked and then destroyed when the window is
closed.
```

```
class HISTORICALPRICE:
    def __init__(self, master):
        self.master = master
        self.frame = tk.Frame(self.master)
        self.Graph = historical_price.graph('DOGE-USD','DOGE PRICE','Time','Price $')
        try:
            self.Graph.initlializeGraph()
        except:
            print("The graph could not be created unfortunately, please revert the code to its original form")
        self.master.destroy()
        ##This method, generates the historical price graph by calling the method in the graph class.

class LIVESTOCKPRICE:
    def __init__(self, master):
        self.master = master
        self.frame = tk.Frame(self.master)
        self.query = ("SELECT * FROM Stockprice ORDER BY stock_number DESC LIMIT 1;")
        self.stockprice = live_stockprice_parser.StockPrice('https://www.google.com/finance/quote/DOGE-GBP',
                                                            'YMlKec fxKbKc')
        self.stockprice.initialize()
        self.fig = plt.figure()
        self.ax1 = self.fig.add_subplot(1, 1, 1)
        self.xs = []
        self.ys = []
        self.loop()

    def animate(self, i):
        ##This animation loop is called in the special function called animation built into the matplotlib package which it utilizes for animating the plot.
        try:
            ##This fetches the latest prices and inserts them into the database
            self.stockprice.priceLoop()
            cnx = mysql.connector.connect(user='[redacted] passwd=[redacted] database='[redacted]
            cursor = cnx.cursor()
            cursor.execute(self.query)
            for (stock_number, time, price) in cursor:
                time2 = time
                price2 = price
            ##The code above fetches the price from the database.
            self.xs.append(str(time2))
            self.ys.append(price2)
            print("time is,", str(time2))
            print("price is,", price2)
            self.ax1.clear()
            self.ax1.autoscale(enable=True, axis='both', tight=True)
            self.ax1.plot(self.xs, self.ys, scaley=True, scalex=True, color="blue')
            plt.xticks(rotation=45, rotation_mode="anchor")
            plt.subplots_adjust(bottom=0.2, top=0.9)
            plt.xlabel('Date')
            plt.ylabel('Price')
            plt.title('Live price of DOGE')
            cursor.close()
            cnx.close()
            self.master.destroy()
        except:
            print("The live graph cannot be created, please ensure that all the modules are installed, please revert the code to its original form")

    def loop(self):
        ##below is the built in animation function in the matplotlib package.
        ani = animation.FuncAnimation(self.fig, self.animate, interval=1000)
        plt.show()

def main():
    try:
        ##a tkinter instance is created, and a target resolution is set which can be adjusted below.
        root = tk.Tk()
        root.geometry("1280x720")
        app = MAIN(root)
        root.mainloop()
    except:
        print("CATASTROPHIC ERROR PLEASE REVERT CODE TO ITS ORIGINAL FORM")
if __name__ == '__main__':
    main()
```

HISTORICAL PRICE MODULE

Title

```python
import yfinance as yf
import matplotlib.pyplot as plt
from datetime import date, timedelta

class graph():
    def __init__(self, tag, title, x_axis, y_axis):
        self.title = title
        self.x_axis = x_axis
        self.y_axis = y_axis
        self.tag = tag
        self.start_date = None
        self.end_date = None
        self.data = []
        self.downloadStockData()
        self.plt = plt
        ##the variables are initialized.

    def downloadStockData(self):
        self.start_date = date.today() - timedelta(days=7)
        self.end_date = date.today()
        self.data = yf.download(self.tag, str(self.start_date), str(self.end_date))
        ##The data is downloaded from the day today to the last 7 days using the date time module. As well as this, the yfinance module.



    def initlializeGraph(self):
        self.plt.figure(figsize= (20,10))
        self.plt.title('Opening Prices of {} from {} to {}'.format(self.tag,self.start_date,self.end_date))
        self.plt.plot(self.data['Adj Close'])
        self.plt.show()

    ##The module above, initializes the graph and shows the plot when called.



if __name__ == "__main__":
    Graph = graph('DOGE-USD','DOGE PRICE','Time','Price $')
    Graph.initlializeGraph()

    ##the above is for testing purposes.
```

LIVE STOCKPRICE PARSER

Title

```python
import requests
from bs4 import BeautifulSoup
import time
import schedule
import threading
import mysql.connector
import matplotlib.pyplot as plt
import mysql.connector
import _thread



class StockPrice:

    def __init__(self, url, cLass):
        ##All of the local variables declared.
        self.fig = plt.figure()
        self.cnx = mysql.connector.connect(user='[redacted]s', passwd=[redacted]database="Information")
        self.cursor = self.cnx.cursor()
        ##Cursor is created which is used to connect to the database with.
        self.count = 0
        self.url = url
        self.page = None
        self.soup = None
        self.price = 0.00
        self.cLass = cLass
        self.query_template = ("INSERT INTO Stockprice "
                               "(time, price) "
                               "VALUES (%(current_time)s, %(newstr5)s)")
        self.initialize()

    def initialize(self):
        self.page = requests.get(self.url)
        self.soup = BeautifulSoup(self.page.text, 'lxml')

        ##The beautful soup page is created with the parameter 'lxml'

    def parsePrice(self):
        self.initialize()
        price = self.soup.find('div', class_=self.cLass).text
        return price

    ##The module above finds the specific class in the beautiful soup page which it references. it returns the live price from the soup site.
    def priceLoop(self):
            value = self.parsePrice()
            print("VALUE PARSED IS NOW:", value)
            t = time.localtime()
            current_time = time.strftime("%H:%M:%S", t)
            composed = str(value)
            newstr = composed.replace("'", "")
            newstr2 = newstr.replace(" ","")
            newstr3 = newstr2.replace("$", "")
            newstr4 = newstr3.replace("(", "")
            newstr5 = newstr4.replace(")", "")
            data_entry = {

                'current_time': current_time,
                'newstr5': str(newstr5),
            }
            print("TIME INSERTED", current_time)
            print("PRICE INSERTED", newstr5)

            self.cursor.execute(self.query_template, data_entry)
            self.cnx.commit()

        ##The section above commits the data to the database.
```

EMAIL SERVICE

Title

```python
import sendgrid
import mysql.connector
import os
from sendgrid.helpers.mail import Mail, Email, To, Content

def sendemail():
    ##The below section of code connects to the database to parse the latest time and price from the database using MYSQL.
    cnx = mysql.connector.connect(user='[redacted]', database='Information')
    cursor = cnx.cursor()
    cursor.execute("SELECT * FROM Stockprice ORDER BY stock_number DESC LIMIT 1;")
    for (stock_number, time, price) in cursor:
        stock_number2 = stock_number
        time2 = times
        price2 = price
    cursor.close()
    cnx.close()

    ##It then connects to the sendgrid api which allows the program to send an email to the client.
    my_sg = sendgrid.SendGridAPIClient('[redacted]')
    ##The key to connect to the API.
    from_email = Email("[redacted]")
    to_email = To("[redacted]")
    subject = "ELON MUSK HAS TWEETED"
    str_price = str(price2)
    string_formatted = "The Price Of Doge Is {}".format(str_price)
    content = Content("text/plain", string_formatted)
    mail = Mail(from_email, to_email, subject, content)
    mail_json = mail.get()
    response = my_sg.client.mail.send.post(request_body=mail_json) ##Sends web request.
```

ELON MUSK PARSER

Title

```python
from __future__ import print_function
import time
import schedule
import tweepy
from datetime import date, datetime, timedelta
import mysql.connector
class ElonMuskListener:
    def __init__(self):
        ##This module inserts the elon musk tweet into the database.
        self.cnx = mysql.connector.connect(user='[redacted]', passwd='[redacted]', database="Information")
        self.cursor = self.cnx.cursor()
        self.user = "elonmusk"
        super(ElonMuskListener, self).__init__()
        self.text = ""
        self.consumer_key = [redacted]
        self.consumer_secret = [redacted]
        self.access_key = [redacted]
        self.access_secret = [redacted]
        self.add_data = ("INSERT INTO elon_musk_tweet "
                    "(info_created_at, full_text) "
                    "VALUES (%(info_created_at)s, %(full_text)s)")


        ##The subroutine converts the parsed tweets into an ascii format which it inserts into the database using a query using the cursor.
    def getTweet(self):
        auth = tweepy.OAuthHandler(self.consumer_key, self.consumer_secret)
        auth.set_access_token(self.access_key, self.access_secret)
        api = tweepy.API(auth)
        tweets = api.user_timeline(screen_name=self.user,count=200,include_rts=False,tweet_mode='extended')
        for info in tweets[:1]:
            encoded_string = info.full_text.encode("ascii", "ignore")
            decode_string = encoded_string.decode()
            self.text = decode_string
            data_entry = {

                'info_created_at': info.created_at,
                'full_text': self.text,
            }
            self.cursor.execute(self.add_data, data_entry)
            self.cnx.commit()
            ##The print statements are simply for debugging.
            print(info.created_at)
            print(info.full_text)
            print("\n")


```

DATABASE MODULE

Test No	Form	Objective	Test Data	Expected Result	Type	Result
1.1	**Historical Price**	The graph should have the time on the x-axis as well as the price on the y-axis.	**N/A**	There should be a graph with time on the x axis and price on the y axis	**Normal**	**Pass**
1.2	**Historical Price**	The graph should be titled with the dates of the start as well as the end of the time period of the data.	**N/A**	There should be graphs titled with the dates of the start and end period of the data.	**Normal**	**Pass**
1.3	**Historical Price**	It should be plotted in the standard cartesian format.	**N/A**	The graph should be plotted in the standard cartesian format	**Normal**	**Pass**
1.4	**Historical Price**	The program should generate the data when a button is pressed	**Button Press**	The program should generate the data when the button is clicked	**Normal**	**Pass**
1.5	**Historical Price**	The button should be labelled with ‘Historical Price’	**Button Press**	There should be a button labelled with historical price	**Normal**	**Pass**
2.1	**Main GUI**	The program should display a window that is interactive to the users input	**N/A**	There should be a window that is interactive to user input.	**Normal**	**Pass**

2.2	**Main GUI**	The program should have buttons that can click and respond to user input.	**Button Press**	The program should have a button which can be clicked and responded to.	**Normal**	**Pass**
2.3	**Main GUI**	Elements of the program should be hidden in other windows and be dynamically generated.	**N/A**	The program should have elements be hidden in other windows and be dynamically generated.	**Normal**	**Pass**
2.4	**Main GUI**	The program should have live updating elements into it such as stock price graphs as well as a twitter feed.	**N/A**	The program should have live updating elements such as stock price graphs as well as twitter feed.	**Normal**	**Partial Fail, a graph has not been implemented on the homepage.**
3.1	**Main GUI**	There should be a live feed of tweets being written at that moment	**N/A**	There should appear a live feed on the home form.	**Normal**	**Pass**
3.2	**Main GUI**	The live feed should be embedded in the GUI as an element of the GUI.	**N/A**	It should be embedded into the home form.	**Normal**	**Pass**
3.3	**Main GUI**	The tweets should have a date/time stamp on them.	**N/A**	The tweets should have the data and time of	**Normal**	**Pass**

				them being posted displayed.		
4.1	**No Form**	The program should send an email to the user in the format of a message.	**N/A**	There should be an email in the users inbox sent by the program in the format of a message	**Normal**	**Pass**
4.2	**No Form**	The program should be able to successfully send an email so that the message appears in the user's inbox.	**N/A**	There should be an email in the user's inbox.	**Normal**	**Pass**
4.3	**No Form**	The program should connect successfully to an email server to send the email successfully.	**N/A**	A connection should be made to an email server to send the email correctly.	**Normal**	**Pass**
4.4	**No Form**	The program should send the email automatically in the background and should not be embedded into the GUI.	**N/A**	The email should be sent automatically in the background and not be embedded into the GUI.	**Normal**	**Pass**
4.5	**No Form**	The user should be able to enter their information and login such as their email address and email server of their choosing.	**N/A**	The user should enter their information and login such as their email address and email	**Normal**	**Fail, the user cannot enter their credentials in the GUI and cannot choose their email service**

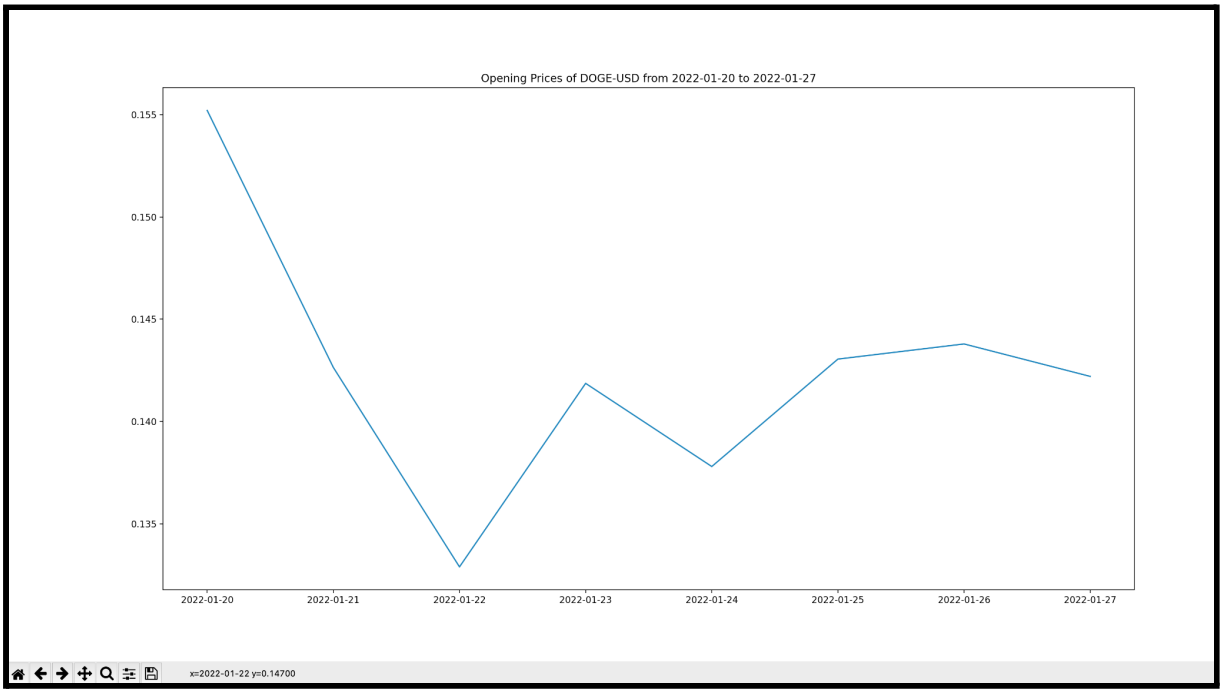
				server of their choosing.		
5.1	**Main Menu**	The displayed tweet should have the full length of the tweet.	**N/A**	The tweet displayed should have the length of the full tweet.	**Normal**	**Pass**
5.2	**Main Menu**	There should be a time/date stamp of the time the tweet was made.	**N/A**	The tweet should have the time and date of when the tweet was made	**Normal**	**Pass.**
5.3	**Main Menu**	The tweet should be embedded into the GUI in the main dashboard.	**N/A**	The tweet should be embedded into the main menu GUI.	**Normal**	**Pass.**
5.4	**Main Menu**	The displayed tweet should be formatted in the style of a standard tweet for ease of readability. Graphically.	**N/A**	The tweet should be formatted in the style of a standard tweet for ease of readability.	**Normal**	**Fail, it is simply text and is not displayed as a graphical tweet.**
6.1	**No Form**	Data collected from the Elon Musk tweet parser should be stored in the database	**N/A**	The data collected from elon musks twitter page should be stored in the database.	**Normal**	**Pass**
6.2	**No Form**	Data collected from the Live Stock Price Parser should be in the database.	**N/A**	The data collected from the stock price module should be stored in	**Normal**	**Pass**

				the database.		
6.3	**No Form**	The data should be formatted in the format DD/MM/YYYY with a timestamp.	**N/A**	The data should be formatted with a timestamp in the british format.	**Normal**	**Pass**
6.4	**No Form**	The data should be able to be accessed concurrently.	**N/A**	The data should be concurrenty accessible	**Normal**	**Pass**
6.5	**No Form**	Each module should have its own login to allow for ease of use.	**N/A**	Each module should have its own login info.	**Normal**	**Pass**
6.6	**No Form**	The data should be stored in individual tables for each module.	**N/A**	There should be individual tables for each module.	**Normal**	**Pass**
7.1	**Live Price**	The graphs should update in regular intervals of 1-2 seconds.	**N/A**.	There should be graphs which update every 1-2 seconds	**Normal**	**Pass**
7.2	**Live Price**	The graphs should be formatted with the time on the x-axis and the stock price on the y axis.	**N/A**	The graphs should have time on the x-axis and the stock price on the y-axis	**Normal**	**Pass**
7.3	**Live Price**	The graph should be titled and be integrated into the Graphical User Interface.	**N/A**	The graph should be integrated into the graphical user	**Normal**	**Pass**

				interface.		
7.4	**Live Price**	The graph should be laid out in the standard cartesian format.	**N/A**	The graph should	**Normal**	**Pass**

EVIDENCE FOR TESTING

1.123)

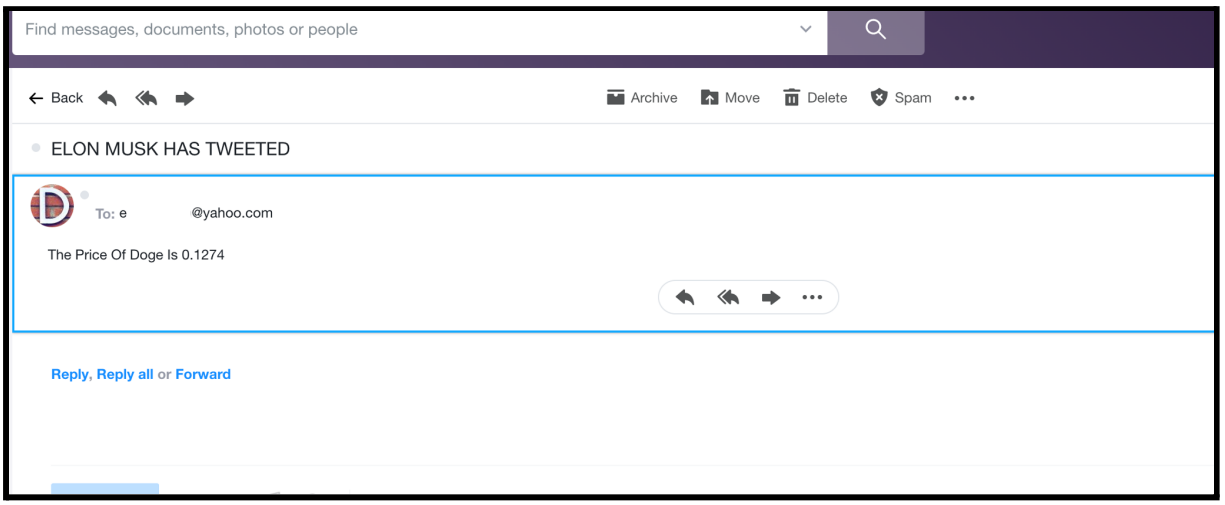


1.45)shorturl.at/cetIN - Youtube Video

2.1234) shorturl.at/kquwR - Youtube Video

3.123)shorturl.at/mzL38 - Youtube Video

4.123)

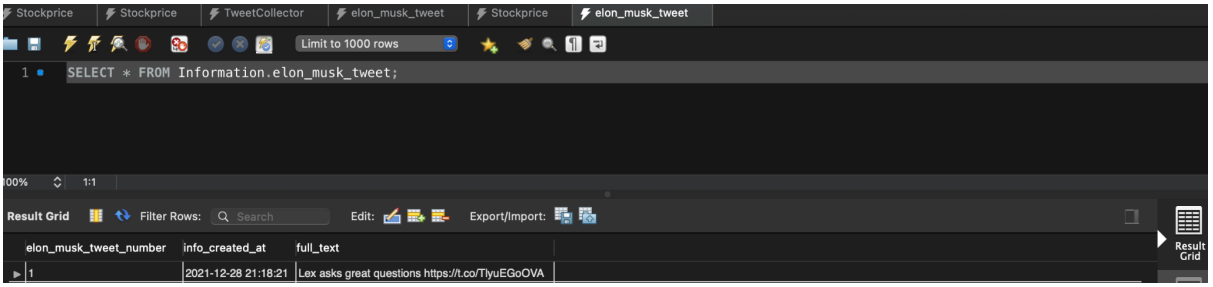


4.4.) shorturl.at/cetIN - Youtube Video

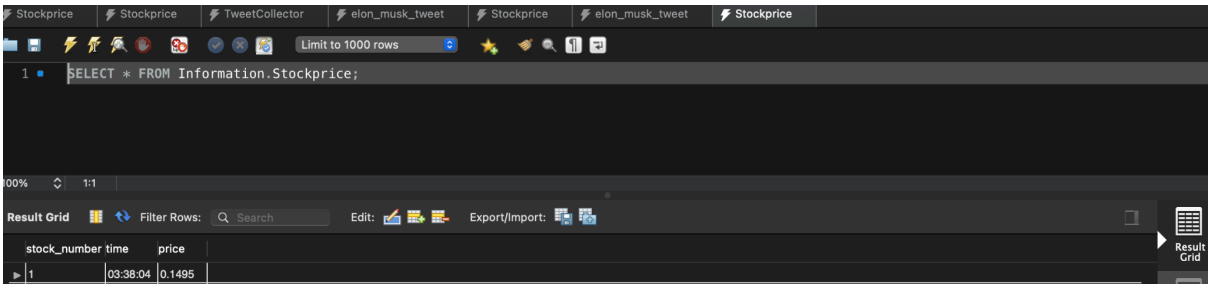
5.123)

Elon Tweeted @cleantechnica Most people have no idea how fast sustainable energy is growing!, at 2022-01-11 04:20:18
Price = 0.1102 Time = 14:23:31

6.13)

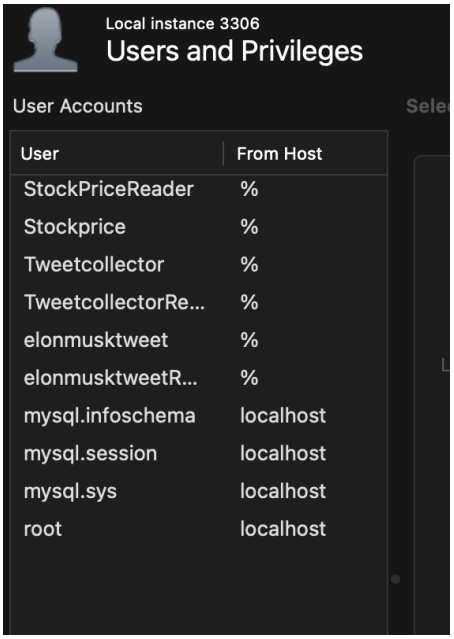


6.2)



6.45)

6.6)

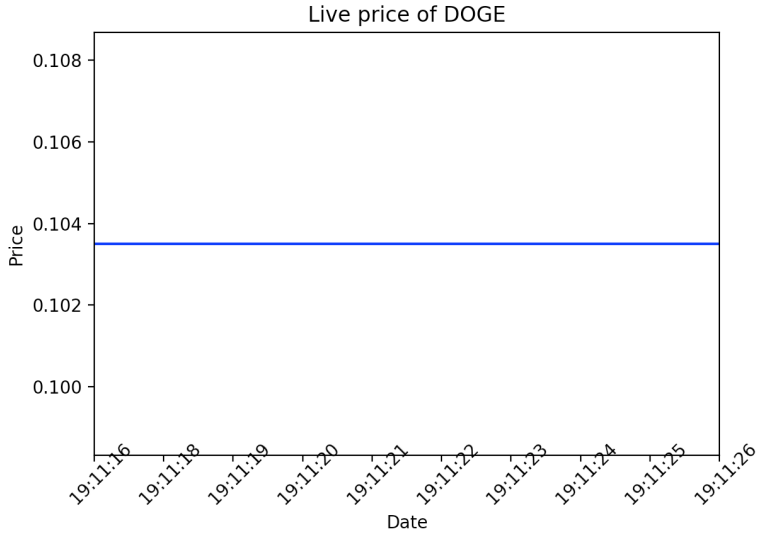


stock_number	time	price
1	03:38:04	0.1495
2	03:38:07	0.1495
3	03:38:10	0.1495
4	03:38:13	0.1495
5	03:38:16	0.1495
6	03:38:19	0.1495
7	03:38:22	0.1495
8	03:38:25	0.1495
9	03:38:28	0.1495
10	03:38:31	0.1495
11	03:38:34	0.1495
12	03:38:37	0.1495

7.1)

```
PRICE INSERTED 0.1030
time is, 18:22:35
price is, 0.103
VALUE PARSED IS NOW: 0.1030
TIME INSERTED 18:22:36
PRICE INSERTED 0.1030
time is, 18:22:36
price is, 0.103
VALUE PARSED IS NOW: 0.1030
TIME INSERTED 18:22:37
PRICE INSERTED 0.1030
time is, 18:22:37
price is, 0.103
```

7.24)



7.3) shorturl.at/cxyAD

Evaluation

In conclusion, the objectives that I have set myself have been partially met. In my project, I have integrated all of the core features that I originally intended on implementing, with only small details being left out. In my opinion, I believe my client Robert Carter will be satisfied with the final product. In addition to this, I believe that my chosen solution was optimal considering my final product which allows for expandability into larger domains. This is due to the fact that I have used data structures such as databases as well as commercial API's. Whilst I have had to make compromises in terms of features I have originally conceived, limitations which were beyond my scope and skillset did not permit me to implement them.

User Need: The program should display the historical price of the Dogecoin cryptocurrency in the form of a graph that is embedded into the user interface. It should be hidden behind a button that says 'Historical Price'.

I have fully achieved all of the objectives regarding this user need. I have chosen a solution of MatPlotLib, which if I were to do again, would use a different software. This is due to the fact that Matplotlib is not thread safe and thus I encountered many problems and difficulties when developing the software which there was not a lot of information on.

User Need: The program should display each element of the program into one GUI which is how the user will interact with the program.

I have partially achieved all of the objectives in this section. I have created a main home window with interactive elements which link to other forms. In addition to this, the windows are generated dynamically, and has live updating features such as the stock price graphs as well as the tweet feed. Due to the matplotlibs issues with multithreading as well as animated graphs not being fully integrated into the package, I encountered errors which I had to modify the code of the package to fix. Due to this, as well as due to the fact that matplotlib is not thread safe it was not possible to implement an animated graph into the homescreen.

User Need: The program should have a live feed of tweets made by people in regards to DogeCoin as well as cryptocurrency.

I have fully achieved the objectives set by myself in this section. There is a live tweet feed embedded into the main home screen which has a date/time stamp on it. Overall in the

development, I have encountered issues with the Tweepy package which I had to scrap an entire module and rewrite due to the fact that the package was not thread safe, as well as the fact that there were twitter API limitations such as not pulling tweets too quickly, too often or interrupting the connection thus I have had to scrap almost all of the twitter package. I however overcame this issue, by using a different package which in combination with careful thread timing as well as purposeful pauses in the process allowed me to bypass this issue entirely.

User Need: The program should alert the user if Elon Musk has tweeted about anything regarding crypto or dogecoin.

I have partially achieved the objectives that I have set out in this section, I have implemented a system alerting the user if Elon Musk has tweeted via an email to the user. Overall during the development, I have encountered some issues creating an email server. Due to this, instead of creating my own email server, I used an online service who's API I connected to. In conclusion, I believe that this was the most efficient approach as it became more friendly to the end user. However, I have not implemented a way to enter your email address or choose which email server to use; however as python is an interpreted language, the user would have to unpack and run the python script which during this stage, parameters such as the email address can be modified.

User Need: The program should display every tweet created by Elon Musk's Twitter account in the form of a standard tweet.

In this section of my program, I have partially completed the objectives I have set initially. Whilst I have created a twitter feed which is implemented into the home screen, It is not formatted graphically in the format of a tweet. This was due to difficulties with the tkinter package, as well as time limitations to implement the graphical elements.

User Need: The program should store all the collected data on tweets as well as stock prices in a MySQL database for ease of analysis as well as data storage.

I have fully completed the objectives of this section. I have implemented an SQL server which the program connects to, to store stock prices and tweets in separate tables. I have implemented this using MYSQL as well as the MYSQL package in python. I have had to create my own SQL queries when creating the program and it has been successful as well as expandable to larger organisations as well as saving local storage space.

User Need: The program should show the live graphs of the DogeCoin cryptocurrency. The graphs should have the price of the crypto on the graph and should be displayed as an element in the GUI.

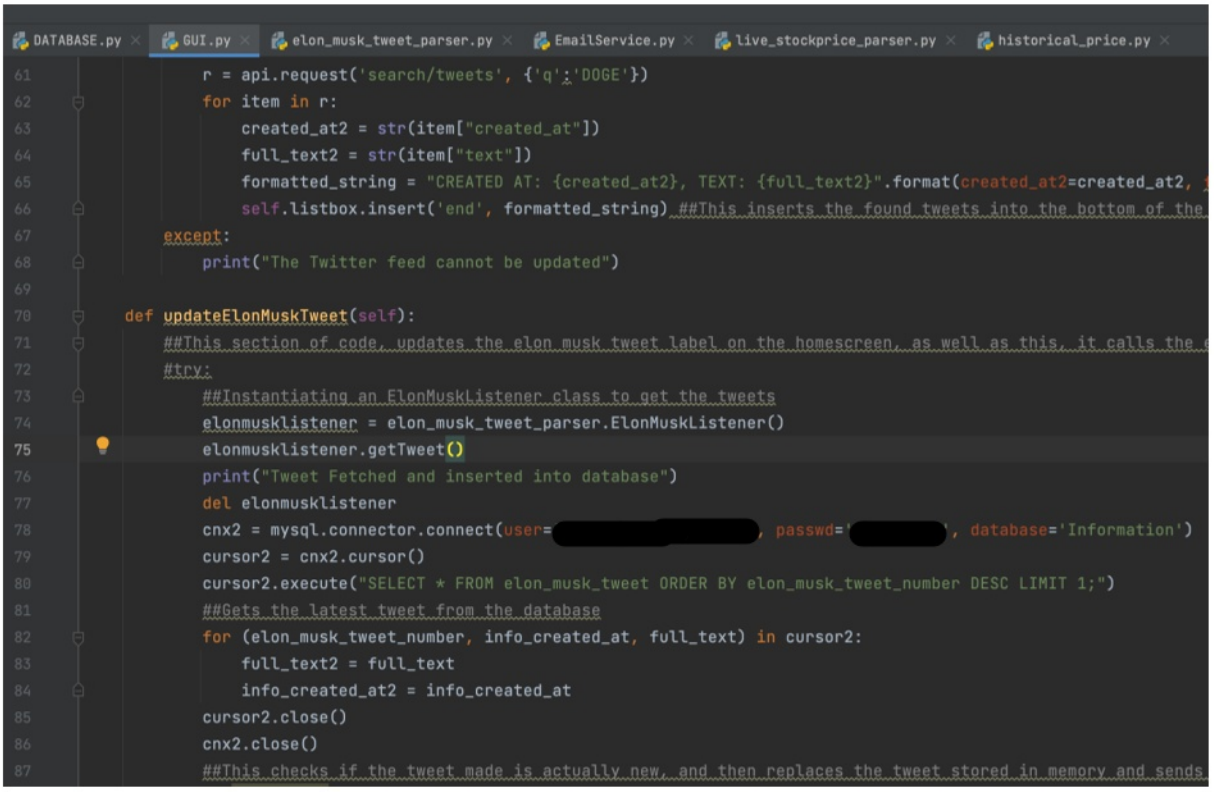
I have met all the requirements set originally in this section of the code. I have implemented a live graph of the dogecoin currency and I have implemented the graph into the GUI as an element which can be viewed as well as manipulated. I faced many issues when coding the live graph, mainly timing the threads to ensure little hang up when looking at the graph. In addition to this, animated graphs in matplotlib are a legacy feature as they are rarely used, however there were few alternatives so I had to use it.

High Level Techniques Used

Group	Model (including data model/structure)	Algorithms
A	Complex data model in database (eg several interlinked tables)	Cross-table parameterised SQL
		Aggregate SQL functions
		User/CASE-generated DDL script
		Graph/Tree Traversal
	Hash tables, lists, stacks, queues, graphs, trees or structures of equivalent standard	List operations
		Linked list maintenance
	Files(s) organised for direct access	Stack/Queue Operations
		Hashing
		Advanced matrix operations
		Recursive algorithms
	Complex scientific/mathematical/robotics/control/business model	Complex user-defined algorithms (eg optimisation, minimisation, scheduling, pattern matching) or equivalent difficulty
		Mergesort or similarly efficient sort
		Dynamic generation of objects based on complex user-defined use of OOP model
		Server-side scripting using request and response objects and server-side extensions for a complex client-server model
	Complex user-defined use of object-orientated programming (OOP) model, eg classes, inheritance, composition, polymorphism, interfaces	Calling parameterised Web service APIs and parsing JSON/XML to service a complex client-server model
	Complex client-server model	

Composition can be seen when the GUI class instantiates other classes such as the ElonMuskListenener class. In addition to this, **Polymorphism** can be demonstrated using my graph class. It can be used for a variety of different graphs and is not specific for a stock or style, these can be defined using the parameters. This allows it to morph into any stock graph object. In addition to this, my Tkinter **interface** is coded in an OOP style with classes for each window which are **Dynamically Generated** depending on if the user has clicked on the button corresponding to that window.. As well as this, I have used **request and response objects as well as server side extensions** for example, for my Yahoo finance price parsing, Twitter API parsing, email API, as well as my database connector. In addition to this, I have a **DDL script** which is run through a mysql connector to create the schemas, tables and database itself.

Composition Example:



```
DATABASE.py | GUI.py | elon_musk_tweet_parser.py | EmailService.py | live_stockprice_parser.py | historical_price.py

            r = api.request('search/tweets', {'q':'DOGE'})
            for item in r:
                created_at2 = str(item["created_at"])
                full_text2 = str(item["text"])
                formatted_string = "CREATED AT: {created_at2}, TEXT: {full_text2}".format(created_at2=created_at2, f
                self.listbox.insert('end', formatted_string) ##This inserts the found tweets into the bottom of the
        except:
            print("The Twitter feed cannot be updated")

    def updateElonMuskTweet(self):
        ##This section of code, updates the elon musk tweet label on the homescreen, as well as this, it calls the
        #try:
            ##Instantiating an ElonMuskListener class to get the tweets
            elonmusklistener = elon_musk_tweet_parser.ElonMuskListener()
            elonmusklistener.getTweet()
            print("Tweet Fetched and inserted into database")
            del elonmusklistener
            cnx2 = mysql.connector.connect(user=[redacted], passwd='[redacted]', database='Information')
            cursor2 = cnx2.cursor()
            cursor2.execute("SELECT * FROM elon_musk_tweet ORDER BY elon_musk_tweet_number DESC LIMIT 1;")
            ##Gets the latest tweet from the database
            for (elon_musk_tweet_number, info_created_at, full_text) in cursor2:
                full_text2 = full_text
                info_created_at2 = info_created_at
            cursor2.close()
            cnx2.close()
            ##This checks if the tweet made is actually new, and then replaces the tweet stored in memory and sends
```

Polymorphism:

```
class graph():
    def __init__(self, tag, title, x_axis, y_axis):
        self.title = title
        self.x_axis = x_axis
        self.y_axis = y_axis
        self.tag = tag
        self.start_date = None
        self.end_date = None
        self.data = []
        self.downloadStockData()
        self.plt = plt
        ##the variables are initialized.

    def downloadStockData(self):
        self.start_date = date.today() - timedelta(days=7)
        self.end_date = date.today()
        self.data = yf.download(self.tag, str(self.start_date), str(self.end_date))
        ##The data is downloaded from the day today to the last 7 days using the date time module. As well as this, the yfinance module.



    def initlializeGraph(self):
        self.plt.figure(figsize=(20,10))
        self.plt.title('Opening Prices of {} from {} to {}'.format(self.tag,self.start_date,self.end_date))
        self.plt.plot(self.data['Adj Close'])
        self.plt.show()
```

Interface:

```
def UpdateHomeScreen(self):
    try:
        ##This method, runs all the submethods to ensure that the home menu updates, it does this in a new thread to prevent s
        while(True):
            time.sleep(5)
            self.updateTwitterFeed()
            self.updateElonMuskTweet()
            self.liveprice()
    except:
        print("The main functions to update the homescreen are not functional, please restart the program or revert it to a pr

def new_window1(self):
    ##This method, creates a new class dynamically when the button corresponding to this button is pressed.
    try:
        self.newWindow1 = tk.Toplevel(self.master)
        self.app = HISTORICALPRICE(self.newWindow1)
    except:
        print("The historical price window could not be opened, please revert the program to its original state.")

def new_window2(self):
    ##This method, creates a new class dynamically when the button corresponding to this button is pressed.
    try:
        self.newWindow2 = tk.Toplevel(self.master)
        self.app2 = LIVESTOCKPRICE(self.newWindow2)
    except:
        print("The live stockprice window cannot be opened, please revert the program to its original state")
```

Dynamic Generation Of Objects:

```
    self.button0 = tk.Button(self.frame, text='Initialize Database', width=25, command=DATABASE.Database)
    self.button1 = tk.Button(self.frame, text='Historical Price', width=25, command=self.new_window1)
    self.button2 = tk.Button(self.frame, text='Live Price', width=25, command=self.new_window2)
    self.button3 = tk.Button(self.frame, text='HOME SCREEN ELEMENTS', width=25, command=threading.Thread(target=self.UpdateHomeScreen).start())
```

```
def new_window1(self):
    ##This method, creates a new class dynamically when the button corresponding to this button is pressed.
    try:
        self.newWindow1 = tk.Toplevel(self.master)
        self.app = HISTORICALPRICE(self.newWindow1)
    except:
        print("The historical price window could not be opened, please revert the program to its original state.")

def new_window2(self):
    ##This method, creates a new class dynamically when the button corresponding to this button is pressed.
    try:
        self.newWindow2 = tk.Toplevel(self.master)
        self.app2 = LIVESTOCKPRICE(self.newWindow2)
    except:
        print("The live stockprice window cannot be opened, please revert the program to its original state")
```

```
class LIVESTOCKPRICE:
    def __init__(self, master):
        self.master = master
        self.frame = tk.Frame(self.master)
        self.query = ("SELECT * FROM Stockprice ORDER BY stock_number DESC LIMIT 1;")
        self.stockprice = live_stockprice_parser.StockPrice('https://www.google.com/finance/quote/DOGE-GBP',
                                                            'YMlKec fxKbKc')
        self.stockprice.initialize()
        self.fig = plt.figure()
        self.ax1 = self.fig.add_subplot(1, 1, 1)
        self.xs = []
        self.ys = []
        self.loop()
```

Request and Response Objects as well as Server Side Extensions

```
my_sg = sendgrid.SendGridAPIClient([redacted]
##The key to connect to the API.
```

```
##The below section of code connects to the database to parse the latest time and price from the database using MYSQL.
cnx = mysql.connector.connect(user='[redacted]', passwd='[redacted]database='I[redacted]
cursor = cnx.cursor()
cursor.execute("SELECT * FROM Stockprice ORDER BY stock_number DESC LIMIT 1;")
```

```
def initialize(self):
    self.page = requests.get(self.url)
    self.soup = BeautifulSoup(self.page.text, 'lxml')

    ##The beautful soup page is created with the parameter 'lxml'

def parsePrice(self):
    self.initialize()
    price = self.soup.find('div', class_=self.cLass).text
    return price
```

```
auth = tweepy.OAuthHandler(self.consumer_key, self.consumer_secret)
auth.set_access_token(self.access_key, self.access_secret)
api = tweepy.API(auth)
tweets = api.user_timeline(screen_name=self.user,count=200,include_rts=False,tweet_mode='extended')
for info in tweets[:1]:
    encoded_string = info.full_text.encode("ascii", "ignore")
    decode_string = encoded_string.decode()
    self.text = decode_string
    data_entry = {

        'info_created_at': info.created_at,
        'full_text': self.text,
    }
```

```
    ##The subroutine converts the parsed tweets into an ascii format which it inserts into the database using a query using the cursor.
def getTweet(self):
    auth = tweepy.OAuthHandler(self.consumer_key, self.consumer_secret)
    auth.set_access_token(self.access_key, self.access_secret)
    api = tweepy.API(auth)
    tweets = api.user_timeline(screen_name=self.user,count=200,include_rts=False,tweet_mode='extended')
    for info in tweets[:1]:
        encoded_string = info.full_text.encode("ascii", "ignore")
        decode_string = encoded_string.decode()
        self.text = decode_string
        data_entry = {

            'info_created_at': info.created_at,
            'full_text': self.text,
        }
        self.cursor.execute(self.add_data, data_entry)
        self.cnx.commit()
        ##The print statements are simply for debugging.
        print(info.created_at)
        print(info.full_text)
        print("\n")
```

DDL Script Example

```
def CreateTables(self):
    ##The DDL SCRIPT is created as a dictionary object which is fed to the cursor
    self.TABLES['Stockprice'] = (
        "CREATE TABLE `Stockprice` ("
        "  `stock_number` int(6) NOT NULL AUTO_INCREMENT,"
        "  `time` time NOT NULL,"
        "  `price` float(64) NOT NULL,"
        "  PRIMARY KEY (`stock_number`)"
        ") ENGINE=InnoDB")

    self.TABLES['elon_musk_tweet'] = (
        "CREATE TABLE `elon_musk_tweet` ("
        "  `elon_musk_tweet_number` int(6) NOT NULL AUTO_INCREMENT,"
        "  `info_created_at` char(100) NOT NULL,"
        "  `full_text` varchar(500) NOT NULL,"
        "  PRIMARY KEY (`elon_musk_tweet_number`)"
        ") ENGINE=InnoDB")
        ##The parameters of the database as well as the engine chosen are assigned here.
```

Some other aspects of the code which I am particularly proud of.

For my GUI code, as it is event driven, the need for a completely object oriented structure was not necessarily necessary. Whilst technically it would not be needed and would have been easier to code it in a standard format, the benefits of the code being more modular as well as more thread safe in my opinion outweighed the time and effort required to code the UI in such a manner. Due to this, I spent considerable time understanding and learning how Tkinter could be implemented into a more object oriented manner as well as understanding little tips and tricks to ensure a snappy menu as a lot of python apps can be slow on one thread as well as not thread safe. Speaking of such, I am also very proud of how I implemented dynamic object generation with dynamic thread creation to ensure my program remained snappy by instantiating a class whenever a new window was created into a new thread. I only applied this however to certain windows as in my experience having too many threads could conflict and could cause python to crash or throw errors which cannot be ignored as it is an interpreted language.

Another aspect which I was proud of was my use of SQL to store my data as well as to retrieve some of my data. This allowed me to expand my program into commercial applications as the idea of storing such data locally can often be infeasible. In addition to this, it allowed me to go from understanding very little about SQL to a firm understanding of SQL as well as how to implement it. As well as this, writing my own SQL scripts also allowed me to further develop my knowledge of programming languages which will help me as a software engineer in the future. Learning MySQL as well as SQL in general was quite the challenge for me, but with tutorials as well as self research, my doubt was cleared and it allowed me to code the solution.

Finally, an aspect which I was proud of, was the skills and knowledge I picked up along the way. This coding project has allowed me to develop my skills in object oriented programming, in python specifically, as well as in how to combine different modules together to create a unique product. Prior to this, coming from a C# background, my knowledge of

python was very limited but after my coding project I am as competent, if not more, at Python. In addition to this, coming from simple console applications to building a desktop application has taught me much about how to design an application effectively, using Tkinter.

Appendix and Bibliography

Sources Used When Coding:

https://stackoverflow.com/questions/47740151/how-to-retrieve-live-price-from-yahoo-finance-stock - Research on the yahoo finance API

https://rapidapi.com/blog/how-to-use-the-twitter-api-with-python/ - Research on the twitter API

https://docs.tweepy.org/en/stable/api.html - Research on the twitter API.

https://realpython.com/python-sql-libraries/#using-python-sql-libraries-to-connect-to-a-database - Research on MySQL

https://dev.mysql.com/doc/connector-python/en/connector-python-example-ddl.html - Research on the python MYSQL connector.

https://www.datacamp.com/community/tutorials/mysql-python#CAC - More research on MYSQL.

https://linuxtut.com/en/45953b4d037a62b2042c/ - Research on embedding graphs into Tkinter

https://stackoverflow.com/questions/29158220/tkinter-understanding-mainloop/29158947#29158947 - Research on Tkinter's main loop.

https://docs.python.org/3/library/tk.html - Tkinter documentation

Interview Transcript, BLUE = ME GREEN = CLIENT

1.) "After looking at the final solution, what are your thoughts on it?"

2.) "I like the final product, it's simplistic GUI and ease of readability… um… was my favourite aspect of the product. "

3.) "In your opinion, do you feel it has met the requirements you have laid initially?"

4.) “Yeah… I feel as if you have implemented all the features I asked for initially. A live tweet feed is handy for monitoring patterns in the market as well as for assessing what peoples opinions are you know? I also find that the Elon Musk tweet feed is very useful, I recently invested some more in DOGE and this will definitely help me…”

5.) “Unfortunately I have made some changes due to technical issues. Do you feel they affect you as a user? “

6.) “I feel that perhaps implementing the animated graph into the homescreen would have been nice, however it is definitely not a dealbreaker, more of an issue of convenience… Umm, apart from that I actually now prefer the less graphical elements of the program as it is a bit easier to read and generally feels more professional in my opinion to be honest…”

7.) “In conclusion, can you please summarise what you think of the program?”

8.) “Haha, sure! The final program was, in my opinion, quite good. I found it to be much more tailored to my needs than other programs and the alerting system is quite good. I also like the features such as the historical price as well as the live price. Compared to my original specifications… I think you have delivered a satisfactory product… Umm, out of 10, I would rate it a solid 8. Thanks Man…”

9.) “No problem, thank you for your opinion.”

10.) “No problem man”