

# National College of Ireland

Bachelor of Science (Hons) in Computing

Software Development

2021/2022

GitHub

[https://github.com/bryanrose75/Trade\\_King](https://github.com/bryanrose75/Trade_King)

Bryan Rose

x18100180

x18100180@student.ncirl.ie

# TradeKing

## Technical Report

### Contents

<b>Executive summary</b> .....	3
<b>Introduction</b> .....	3
Background .....	3
Aims .....	4
Technology .....	5
Structure .....	5
<b>System</b> .....	6
Use Case Diagram.....	6
<b>Functional Requirements</b> .....	7
Requirement 1 <User log in> .....	7
Requirement 2 < Data communication with exchange >.....	9
Requirement 3 <Add Strategies> .....	10
Requirement 4 < Close stream connection when exiting application > .....	13
Requirement 5 < Connect to CoinGecko API > .....	13
Requirement 6 < Synchronised system time> .....	14
Requirement 7 < Page navigation > .....	14
<b>Non-Functional Requirements</b> .....	14
User Trading knowledge .....	14
Performance/Response time requirement.....	17
Availability requirement .....	18
Maintainability requirement.....	18
Usability Requirements.....	18
Save Workspace requirement.....	19
<b>Design &amp; Architecture</b> .....	19

<b>Implementation .....</b>	<b>21</b>
Connecting to An Exchange .....	21
Creating the Breakout and technical strategies.....	24
Executing Trades .....	35
Exiting a Trade.....	38
Creating the Crypto comparison graphs .....	40
<b>Graphical User Interface (GUI) .....</b>	<b>44</b>
<b>Testing .....</b>	<b>54</b>
Unit tests .....	55
Manual Tests .....	59
<b>Conclusions .....</b>	<b>62</b>
<b>Further Development or Research .....</b>	<b>63</b>
<b>References .....</b>	<b>64</b>
<b>Appendices .....</b>	<b>66</b>
Project Proposal .....	66
Objectives.....	66
Background .....	66
State of the Art .....	67
Technical Approach .....	68
Technical Details .....	69
Special Resources Required.....	70
Project Plan .....	70
Reflective Journals .....	72

## Executive Summary

With the cryptocurrency market continuously growing by the day, millions of people are constantly monitoring the ever-changing market. I have designed an application that allows users to set up customisable trading strategies that will be executed through either Binance or Bitmex trading platforms. My application, TradeKing, constantly monitors the market and allows users to pre-set their buy and sell points to allow for optimum trading to occur any time of the day. Users are also provided with a link to detailed descriptions of trading terms and can practice in the market before unknowingly investing. TradeKing is able to send and receive data through the use of application program interfaces (API's). The graphical user interface (GUI) was developed using Tkinter, a python GUI package. Connecting the exchanges and Tkinter both provided their own individual challenges and additional resources were required to assist in my design. Despite these challenges and after many manual tests, TradeKing is a functional application that hold limitless potential for further development and expansion. This document's purpose is to clarify the requirements and technical details set out for the creation of TradeKing Trading application. The following sections of the document will showcase the application and its requirements.

## Introduction

### Background

The reason I have decided to undertake this project is due to the fact that it addresses a problem that I, and millions of others, face on a daily basis. As someone who has invested endless hours into crypto currency trading, creating a program that is able to send market orders to the Binance or BitMex platforms will free up a lot of time for myself to devote to more market research for which coins to invest into in the future. It would also eliminate the risk of missing out on trades that may occur at obscure hours of the day, such as in the middle of the night.

In order to meet these objectives I needed to design an application that, with the use of application program interfaces (API's) provided by the selected trading platforms, is able to successfully pull in data about the current values of the crypto currencies and be able to

send data through the API's to perform buy and sell actions on those crypto currencies at specific buy and sell points that will be predetermined in the code so that the bot follows a trading strategy. The application should also have an attractive graphical user interface to make interaction with the data easier.

## Aims

TradeKing will be an automated trading bot with the objective of helping users execute trades on cryptocurrency trading platforms, Bitmex or Binance, by following a trading strategy that has been specified to the trading bot. This would benefit users who are avid or casual traders as it would eliminate the risk of losing out on trades due to the trading platforms being inaccessible to the user at certain points in the day for numerous reasons. One of these reasons is due to the user being away from their computers to do other daily duties. Users would benefit by gaining more time in the day to spend working or creating new projects. TradeKing would eliminate the need of waiting for certain buy and sell points to be hit for their trades to be executed as it will all be done automatically through the application.

TradeKing has a login where its users will log into the application using their API keys for either the Binance or Bitmex trading platforms. Hyperlinks for both BitMex and Binance will be present in the form of a button that will direct the users to the respective webpages that have step by step processes to register and locate their API keys for these Testnet platforms.

Once logged in, TradeKing aims to have a simplistic design that will display only what is necessary to the user for making trades, consisting of 4 frames within the main frame window. The top left frame will have a watchlist component where a user can type in any cryptocurrency trading pair that is available on the respective platform and is able to get live prices for these trading pairs. The top right frame will have a button to add strategies, as well as a button for definitions that will direct the users to an online glossary that contains the definitions to all the necessary trading terminology that are either technical or break out. There will be inputs to enter the respective parameters according to the strategy as well as the time frame, amount to invest, take profit, and stop losses. There will also be buttons to swiftly turn on/off the trading strategy or delete it. The bottom left

frame will be used to log in all the events that are currently happening, whether it be for a market order made, trade closed, candle received etc.

There will also be an extra feature for crypto graphs where a user will be able to change pages and see the crypto trend prices for their desired coins displayed with customizable inputs.

## Technology

The Main language that will be implemented into this project will be python. Python is an extremely capable language and can incorporate many languages into one with the use of importing libraries to help organise data and display data in a graphical user interface (GUI) (Saabith, 2019). Anaconda will need to be installed in order for me to operate Python on my device and to have the means to import the libraries required by my program. One of the main libraries that I will use will be “Pandas” library to calculate the technical indicators when trading to perform buy and sell orders.

The GUI will be created using the python GUI package ,Tkinter. Using this technology brought up a variety of interesting and exciting challenges as it is one that I had never worked with prior to this project.

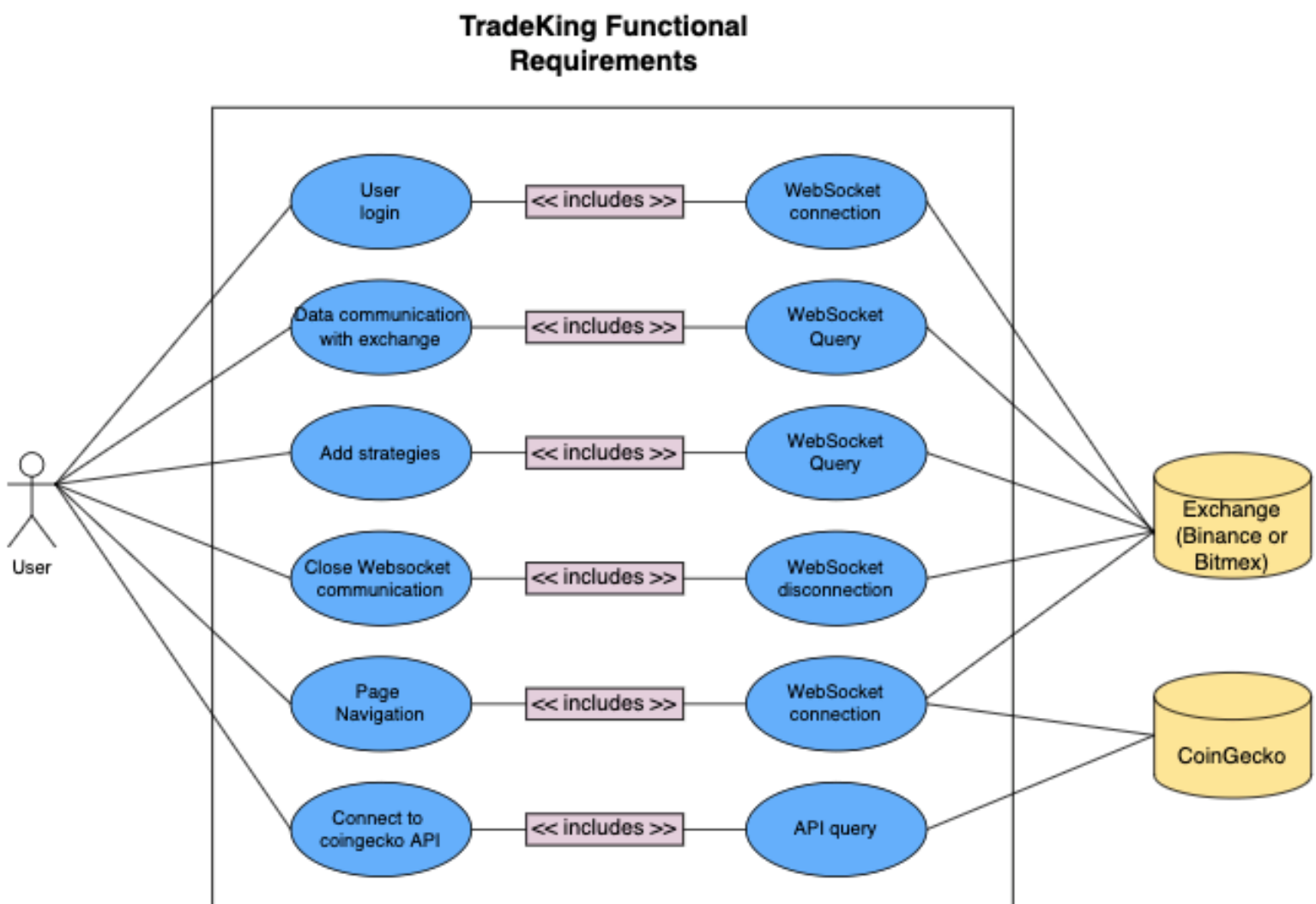
## Structure

In this report I will first cover the introduction of what I am proposing to create, the background in which motivated me to create this application, what my end goals for this application ultimately will be alongside the technologies that I plan on implementing into my application, the system requirements that would consist of all functional and non-functional requirements listed in order of importance with short, imperative sentences describing what they are. I will also provide use case diagrams to visually display how to use and interact with my application by going through step-by-step the processes to successfully create a trading strategy that will be implemented.

Moving passed the requirements section to the design and architecture section where I will elaborate on all algorithms that are proposed to be used within TradeKing application

## System

### Use Case Diagram



## Functional Requirements

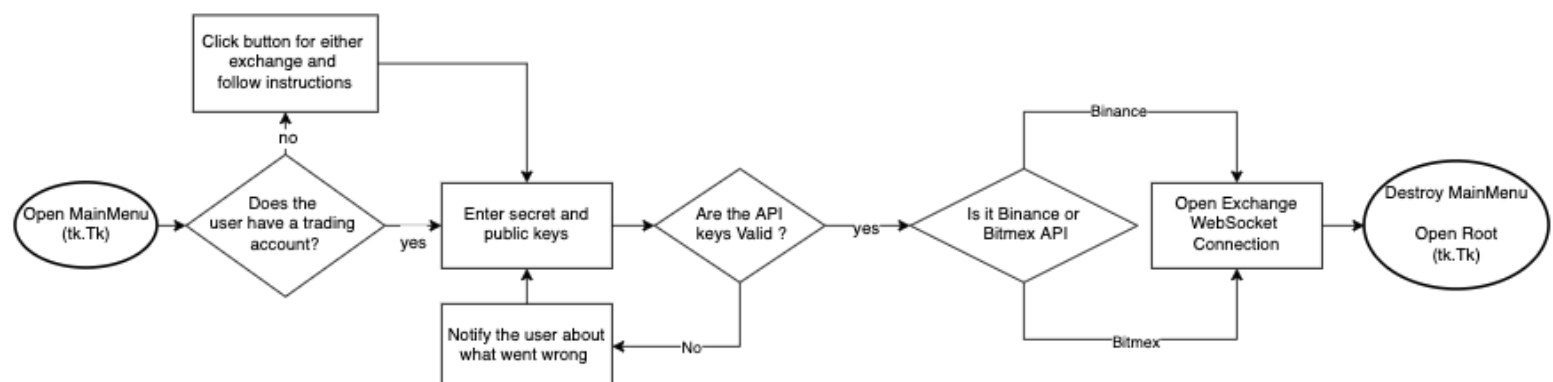
### Requirement 1 <User log in>

#### *Description & Priority:*

In order for the application to function, it is required that the user is registered with either Binance or BitMex trading applications. It is essential for users to have an account on one of these two platforms because in order for the application to work, it requires an API key from one of these two applications. Registering for one of these applications is simple, as it is a testnet platform and would only require an email and password to register, unlike alternative applications that have a very lengthy signup process. There will be a hyperlink available on the login of TradeKing that will redirect the user to the respective platforms where they can sign up freely for an account. Once a user has verified their email and has an account, they will be required to get the both the main and secret API keys that they will use as log in credentials for TradeKing. TradeKing will offer a video demonstration on how to find these keys for the user.

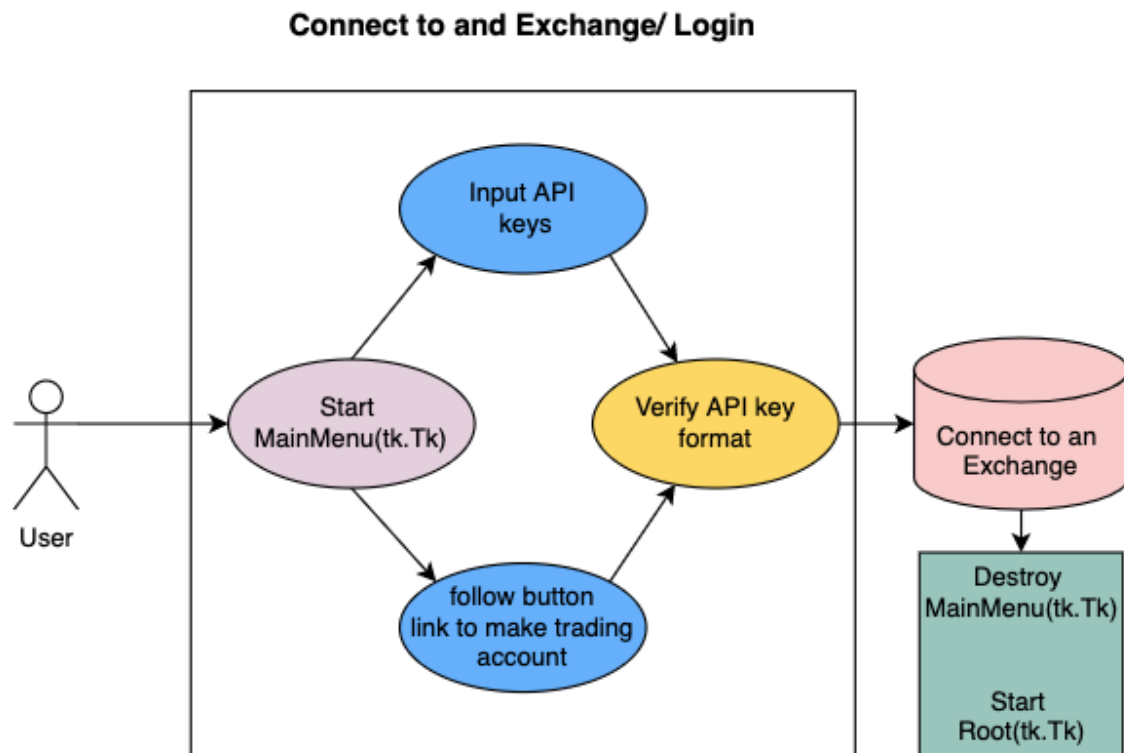
#### *Flow Diagram:*

##### Login Flow





*Use Case:*



### Flow Description

#### Precondition

The system is in initialisation mode, user is first brought to the login page to initiate the WebSocket connection to a trading exchange.

#### Activation

The user begins on the login page where they can enter their API and Secret API keys to establish a WebSocket connection. If the user has not registered with any of the exchange platforms, then there are buttons for the respective supporter platforms where they can follow clear instructions on how to register and obtain their required keys.

#### Main flow

1. Actor inputs their API keys, the system checks if the format is for either Binance or Bitmex. The Actor is redirected to the Trading window

2. The Actor clicks on a button for either Binance or Bitmex and they are redirected to a tutorial webpage that shows exactly how to register and retrieve the API keys for the respective exchange.
3. The Actor then repeats steps from 1.

### **Alternate flow**

A1 : <API format incorrect>

1. The system compares API key with the required format
2. If The API key does not conform to the required regex format, an error message is displayed, notifying the user that the key format is incorrect.
3. The actor may retry their login attempt with API keys in position 1.

### **Termination**

Once the API keys have been validated to be of the required format, then the actor is directed to the trading window.

### **Post condition**

Success Condition

1. Actor enters valid API keys.
2. Actor logs into the Root(tk.Tk) window.

Failure Condition

1. API key format is invalid
2. API secret key format is invalid
3. Correct API format but invalid API keys will result in failure to perform orders.

Requirement 2 < Data communication with exchange >

*Description & Priority:*

The data that is going to be used is regarded as private data and is accessed via the use of an API WebSocket connection. TradeKing is a data-driven application as it heavily

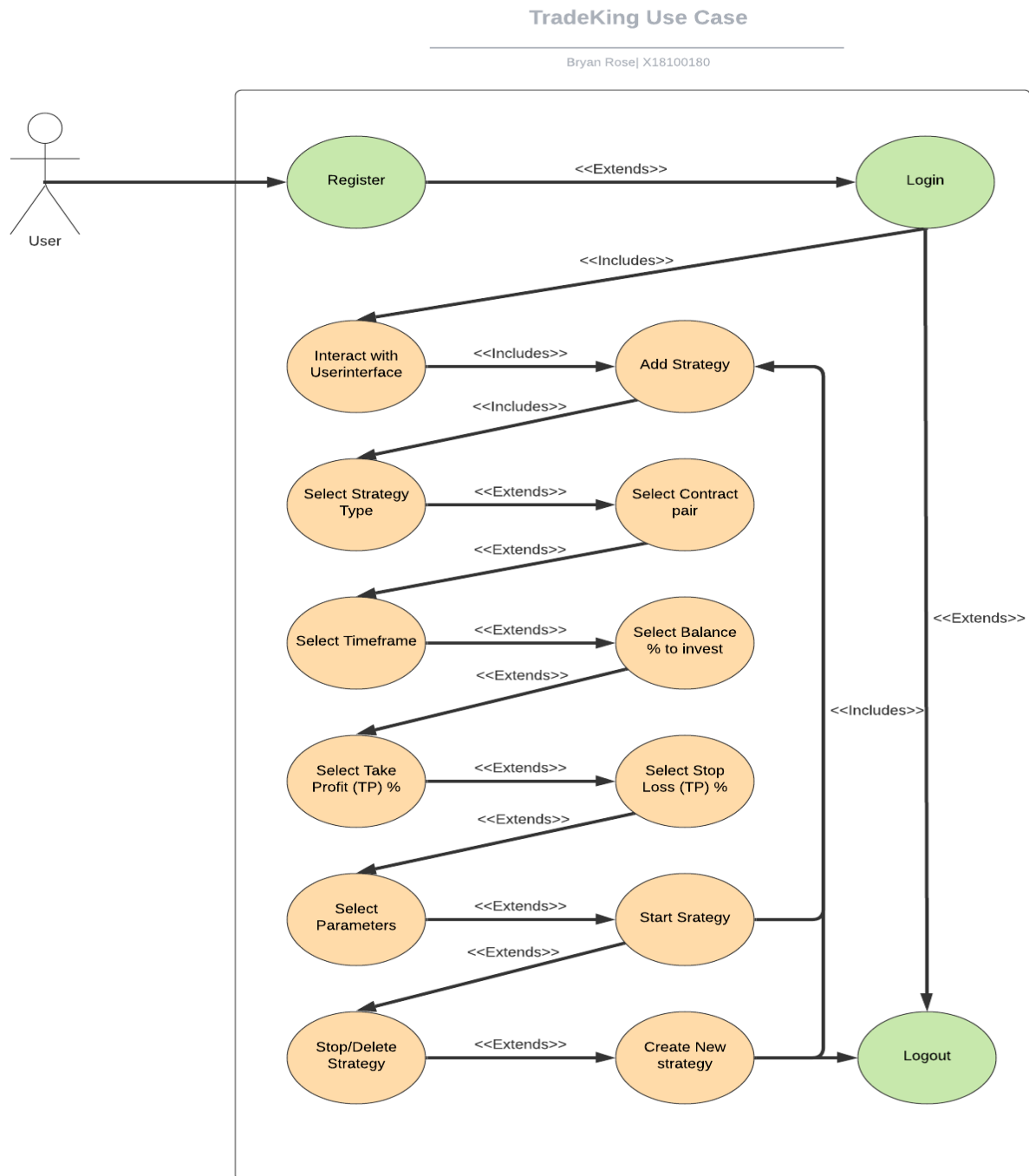
relies on the use of data from the Binance or BitMex trading platforms in order to carry out various tasks such as buy and sell orders depending on the signals being sent from the exchanges. Trade king would subscribe to data feed using a web socket communication to one of the exchange servers API, depending on what API keys were used within the log in process. The Binance and Bitmex exchanges have separate regex formats that are checked when logging in to determine which exchange to connect to.

### Requirement 3 <Add Strategies>

#### *Description & Priority:*

The main feature of TradeKing is the ability for the user to create a trading strategy that will be followed using the user's pre-set conditions that are set during the strategy creation process. The Strategy will use Websocket communication to connect with an exchange in order to buy and sell cryptocurrencies depending on what signals are being sent by the market. The trading strategy will run continuously until the user turns it off. The user is able to have multiple strategies running at a time as long as the websocket subscription limit of 200 connections is not exceeded.

Use Case:



## **Flow Description**

### **Precondition**

The system is in initialisation mode when the user clicks on the “Add strategy” button. The system will display the required inputs that are to be filled out by the user.

### **Activation**

This use case will commence upon logging into the system and pressing the “Add Strategy” button.

### **Main flow**

1. The system identifies that the add strategy button has been clicked
2. The system displays the strategy template that is required to be filled out by the user with their custom strategy inputs.
3. The system waits for the strategy type to be selected to have the correct corresponding parameter inputs.
4. User clicks on the on button to start the strategy
5. The system continuously sends and receives data from the exchange creating orders according to the users inputs.
6. User clicks the off button
7. The System stops the strategy

### **Alternate flow**

A1 : <Invalid trading volume breakout strategy>

1. The system checks if there is enough volume being traded when trading on the breakout strategy.
2. If there is not a minimum volume of 5, the system throws an error

### **Termination**

The strategy can be turned off at any moment using the off button.

### **Post condition**

Success Condition

1. The strategy starts and is logged to the logging component.
2. The trade state is displayed below showing the position of it with profit and losses.

#### Failure Condition

1. The strategy doesn't start.
2. The system fails to make orders.

#### Requirement 4 < Close stream connection when exiting application >

##### *Description & Priority:*

TradeKing uses websocket connections to send and receive information between the application and the trading platform Bitmex or Binance. These connections are opened when a user logs into the application using their API keys. Once the application closes, it is vital for security reasons that the websocket connections close along with it. It is easy to close the websocket connections and can be achieved with a single line of code. When the user is on the Root window that hosts all the trading tools, the WebSocket's are open. When the user navigates out of this page to the graphs page then the WebSocket's are closed.



```
self.binance.ws.close()
self.bitmex.ws.close()
```

The image shows a code snippet with two lines of code: `self.binance.ws.close()` and `self.bitmex.ws.close()`. To the right of the first line is a blue arrow pointing to it with the text "Close Binance WebSocket connection". To the right of the second line is a blue arrow pointing to it with the text "Close Bitmex Websocket connection".

#### Requirement 5 < Connect to CoinGecko API >

##### *Description & Priority:*

When navigating to the graphs page the Websocket connections are closed as the graphs and trading windows are both hosted separately. This is designed to maintain the security of the application as the WebSocket connections are only open when they need to be. In order for the graphs page to work, it requires an API connection to retrieve historical data on crypto currencies to build the graphs. Coingecko is a website that has a public API which allows for the retrieval of historical coin data. In

order for the graphs page of the application to function, there needs to be a connection to the coingecko API.

#### Requirement 6 < Synchronised system time>

It is extremely important that a user synchronises their systems clock to be that of the universal global time and that it is not a manually set time. Due to trades being made in real time there is a time stamp sent with order requests and time stamps received by the application on orders and when the time is unsynchronized it will throw an error to the user.

#### Requirement 7 < Page navigation >

Tkinter is a python GUI framework that operates by having all the features of the application load into a Tkinter canvas where the canvas acts similar to html on a webpage. Frames are made and GUI attributes can be packed into these frames. The Tkinter windows run on an infinite loop. The navigation for the application consists of breaking a loop for one window, while simultaneously starting a loop for another window. This was designed to keep the application light-weight as it wouldn't be required to run all the features and pack everything into a singular window.

### **Non-Functional Requirements**

#### User Trading knowledge

In order for users to successfully use TradeKing to its fullest potential, the user is recommended to have prior trading knowledge regarding the terminology used within the trading sector, as much of this terminology is used throughout the application. A new user, however, will have the ability to read up about each terminology and its definition via a small pop-up window next to each action. When clicked, it provides the user with a description of what the action represents, along with recommendations

on the safest use of the feature it is describing. This ensures that users are not blindly risking their finances and that beginners understand the fundamentals of trading.

Main trading terminology and knowledge that is recommended:

**Strategy type:**

*Technical* – a technical trading strategy would rely upon technical indicators to generate trading signals. This is the belief that all the information regarding a trade is contained within the price and the trends movement (Corbet, 2019).

*Breakout* - this trading strategy involves looking for levels or areas that a trade is unable to move beyond and waits for the trade to move beyond those levels as it could result in continuous movement in that direction. Once the price moves beyond those levels it is referred to as a breakout (Yi, 2022).

**Contract:**

A contract would refer to the cryptocurrency pairing e.g. BTCUSDT – bitcoin traded in united states dollar tether, and which exchange server the API is operating through for example Binance (binance.com).



**Timeframe:**

The timeframe refers to the time intervals between the current candle stick and the previous candlestick where the application receives trade signals differently according to what timeframe it is trading on e.g. a 1 min timeframe would receive more frequent signals appose to the 1 hr. The available time frames to run the strategies on TradeKing are:

- 1 min
- 5 min
- 15 min
- 30 min
- 1 hr



- 4 hr

**Balance %:**

The balance percentage refers to what percentage of a user's overall available balance the user wants to invest into a certain currency. So if a user has ,for example, a total of \$1000 of investable balance and wants to invest 20% of into e.g. bitcoin, then the user will input '20%' into the required field and Trade king application will put a buy order of \$200 into bitcoin. A user will only be able to put in a maximum of 100% of their balance as if they were to go above 100%, they would be using money that they do not possess making it impossible to place orders.

**TP %:**

A take profit (TP) is commonly known as a limit order which will guarantee that a position is closed at a pre-defined price point or greater (Guides, 2018). If the position of the cryptocurrency value moves in the correct direction of take profit (TP) level, the position is then closed for a profit. This method is more popular with short-term traders that would be monitoring daily/hourly price moves and the most effective way to execute trades. This will also take the human element out of managing open positions.

**SL %:**

Stoploss is used to protect investments where a stoploss order effectively is triggered in the market as soon as the price of an asset drops below the stock price identified by the trader (Vezeris, 2018). For example, a user has invested €100 into a specific cryptocurrency where the value was at €50 a share. The trader will set a stoploss order for €40. If the value then drops over the next couple of hours and drops below €40, this will then trigger the stoploss order and the position will be closed at €39.95, resulting in a loss of €10.05 per share. This method is used to minimize losses on trades due to market crashes.

**Parameters:**

- MACD Fast Length:

The moving average convergence divergence (MACD) fast length is generally a 12 day exponential moving average (EMA) ), although it is the trader's choice to set what the fast lengths parameters will be and it is used to create indicators that will be used in a short-term trading trend and will be used with the slow length (Cohen, 2021).

- MACD Slow Length

The moving average convergence divergence (MACD) slow length is generally a 26 day exponential moving average (EMA), although it is the traders choice to set what the slow lengths parameters will be and it is used to create indicators that will be used in a short term trading trend and will be used with the fast length (Cohen, 2021)

- MACD Signal Length

The MACD is calculated when the 26-day period EMA is deducted from the 12-day period EMA and the results from this calculation is called a 'MACD line'. A 9-day EMA is recommended to be used for the signal length and will be plotted over the MACD line which can then act as a trigger function to carry out buy and sell orders (Kanami, 2021)

MACD Fast Length	
MACD Slow Length	
MACD Signal Length	

#### Performance/Response time requirement

Response time is an extremely important aspect of TradeKing as trades are to be made in the real time. The response from when a user starts the strategy to when the system recognises the action could result in a user missing out on a trade when trading in the 1m timeframe.

### Availability requirement

Availability is import when offering a good service, however it is not of key importance. Users can access TradeKing 24hrs daily from their desktop. As Binance and Bitmex platforms allow for 24hr trading, users are able to activate their strategies at any point in the day. Any unexpected errors that may arise should be dealt with in a timely manner to ensure availability is met to the users. Closing the application will close the WebSocket connections and hence, the trades, meaning that there are no open trades.

### Maintainability requirement

Developing TradeKing has been a new learning experience using coding functionality and systems I have not created before. For my own understanding, it has been an important habit to clearly label functionalities with comments and briefly comment what the system is doing and how. As a result, I find it easier to recollect how my code runs which enables me to solve unexpected situations quickly and efficiently. I have also designed my code using coding classes that handle all the interactions and functionality for features. It is important that I use effective coding design practices to prevent unnecessary dependencies within the project that can affect other areas of the project when modifications are made to the platform.

### Usability Requirements

Usability requirements involve how straightforward the system is to use and in order to achieve maximum usability, the application must be designed to be simple in terms of its user interface. It must only display information necessary to add a strategy and also must be easy to follow for users that have never used the application before when it comes to creating a strategy. There could potentially be a lot of users with little to no trading knowledge that want to get into automated trading, and thus my application would require definitions and recommendations on sections that may confuse new traders with unknown trading terminology e.g. stop loss, take profit, etc. This will tie

into the learnability of the application. A lot of trading comes down to graphical aids to help users decide their next move with a trade, therefore TradeKing should provide a graphical aid for the movement of the selected crypto currencies and display this aid in a neat and orderly fashion.

TradeKing must also have a high degree of error tolerance so that the user is unable to start a trading strategy without filling in all the necessary fields for that specific strategy to begin. This will be done by adding error messages stating to the user what they have missed and by giving the recommended input variables for those specific missed fields.

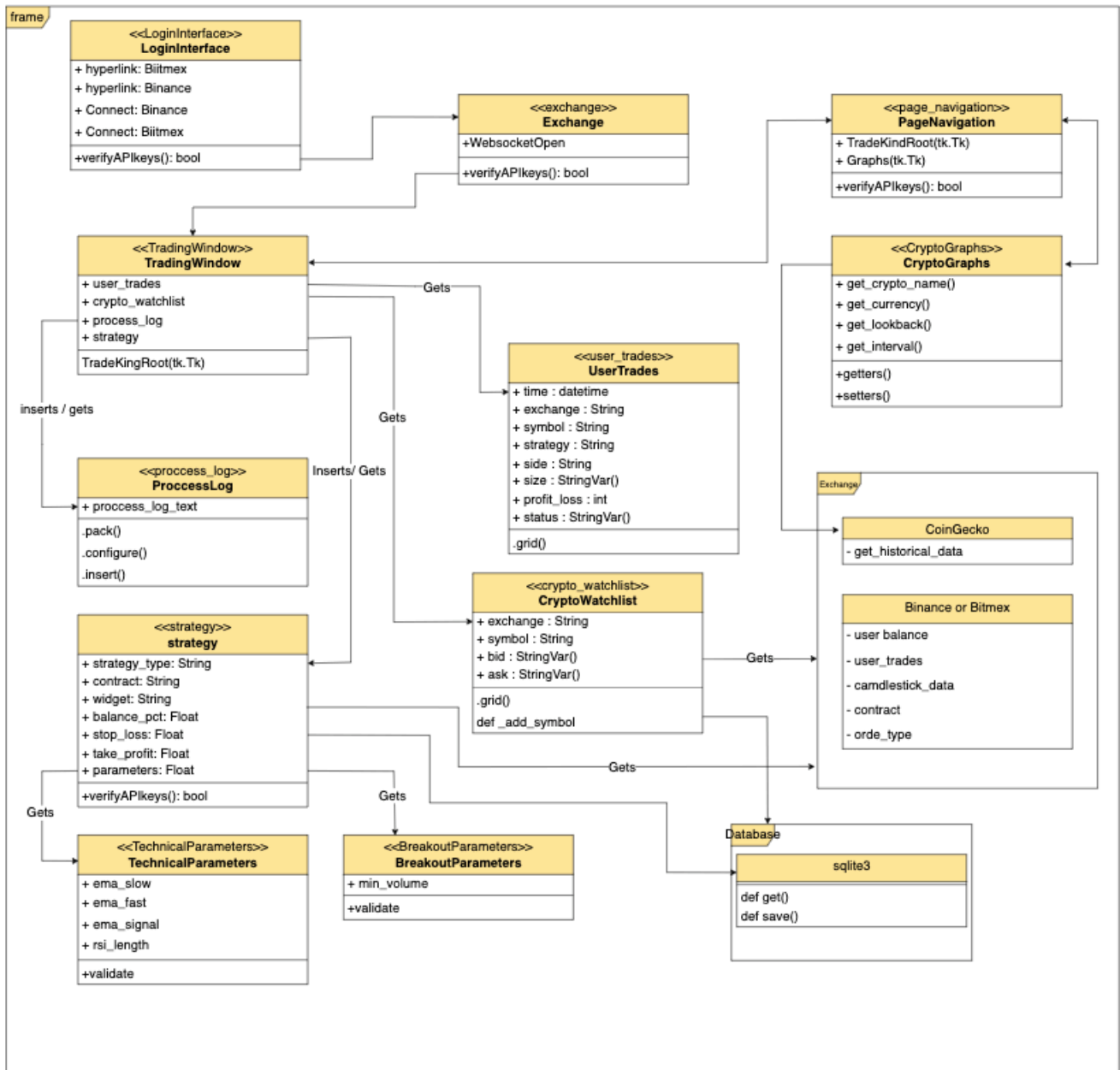
### Save Workspace requirement

TradeKing offers an extra feature for users to save their current workspace environment with their already-made trading strategies and their cryptocurrency pairing coins that had been added to the watch list to a sqlite3 database, so that the next time the user logs into the application using their API keys, all the premade strategies are already loaded. This is so that the user doesn't have to waste time creating strategies every time they enter the application. Another reason the safe workspace is useful for users is the fact that when a user enters into the graphs page, essentially the Tkinter mainloop for the trading bot is destroyed and the graphs mainloop is created, so when the user navigates back to the trading bot having saved their workspace, they would not be required to set up all their strategies and add the coins back to the watch list.

## Design & Architecture

The design architecture of TradeKing is specifically for users to run from their desktops, as it has been developed using the Tkinter framework that is solely for python programs. This limits the accessibility of the application but enhances the security. There is no hosting platforms that may be on web based applications however there is a direct connection from TradeKing to the exchange servers via API

calls. This would reduce the risk of the website being hacked and as it is a stand-alone application that has a single user per machine it increases the speed in which the application can run.



## Implementation

### Connecting to An Exchange

Connecting to an exchange for both the Binance and Bitmex applications were done by strictly following the documentation that is set out for the respective exchanges. Any alterations to the code within the `binance.py` and `bitmex.py` files within the exchange's directory in `trade king` resulted in the application to fail when making trade orders with the error.

```
Error while making GET request to /fapi/v1/account:
```

```
{'code': -2015, 'msg': 'Invalid API-key, IP, or permissions for action, request ip: 37.228.285.198'}
```

```
(error code 401)
```

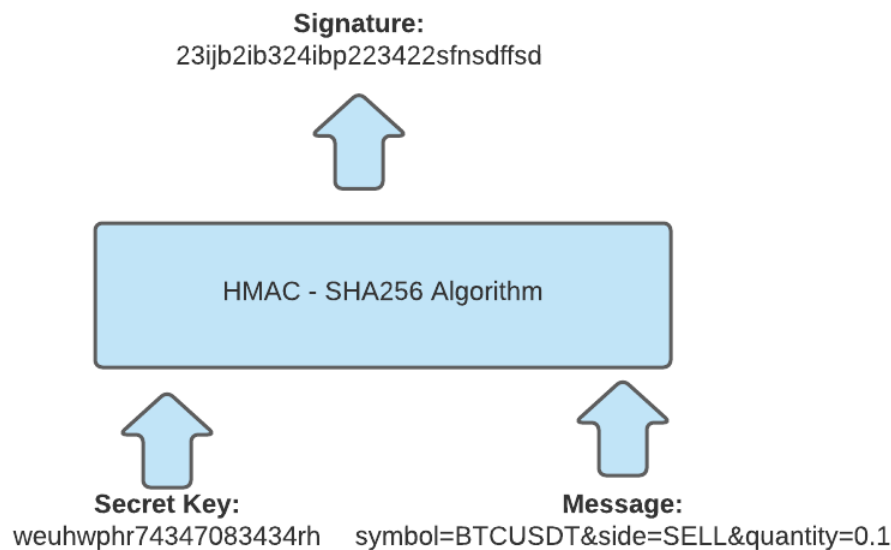
One of the algorithms that is used within the application is the Hmac-sha256 algorithm. This is a type of keyed hash algorithm that is constructed from the Sha-256 hash function and is used as a hash-based message authentication code (HMAC) (Choi, 2020). Within the HMAC process the secret key is mixed with the message data, the results are hashed with the hash function, it mixes the secret key and hash values again and then follows with the application of the hash function a second time. The output of this process is a 256 bit length hash key.

```
def generate_signature(self, data):  
    # 1. use encode method to convert the string to a byte  
    # 2. convert the library to a query string using the url encode function, then the string the a byte  
    return hmac.new(self.secret_key.encode(), urlencode(data).encode(), hashlib.sha256).hexdigest()
```

Fig. Example of the HMACSHA256 integration

A benefit of using the HMAC functionality is that it is able to determine if a message has been tampered with over an insecure channel, provided that the receiver and sender share a secret Key (Choi, 2020). The sender then computes the Hash value used for the original data and sends both the original data and the value back as a single message.

Recalculations are done by the receiver on the received hash value message and verifies that the received HMAC matches the transmitted HMAC value. If there are any changes in the data or hash value it will lead to a mismatch, due to knowledge of the 'secret key' being a requirement to change the message and produce the correct hash value. Therefore if the computed and original hash values match, then it will authenticate the message.



```
def place_order(self, symbol, side, quantity, order_type, price=None, tif=None):
    data = dict()
    data['symbol'] = symbol
    data['side'] = side
    data['quantity'] = quantity
    data['type'] = order_type

    if price is not None:
        data['price'] = price

    if tif is not None:
        data['timeInForce'] = tif

    data['timestamp'] = int(time.time() * 1000)
    data['signature'] = self.generate_signature(data)
    # End point used for the order status function
    order_status = self.make_request("POST", "/fapi/v1/order", data)

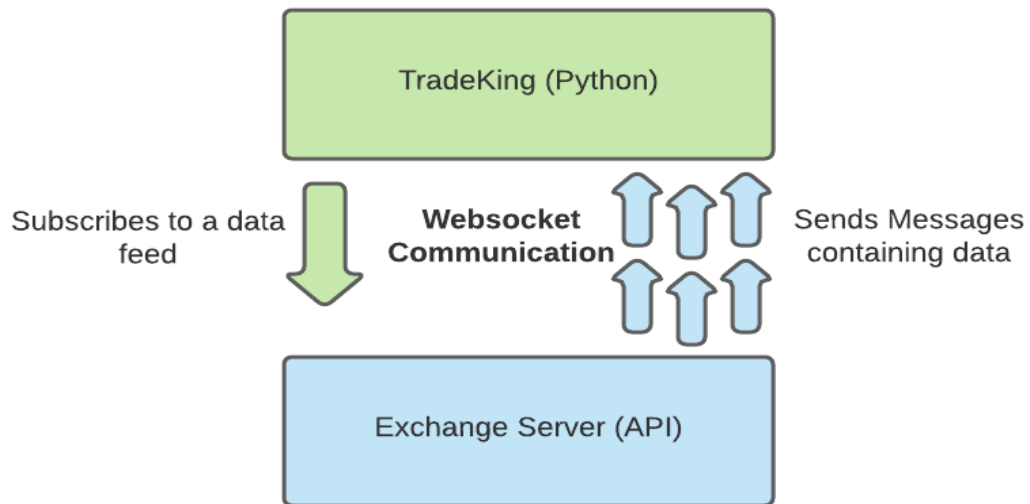
    return order_status
```

Required Data structure as described on Binance Documentation

TimeInForce is a special trading instruction to be used when placing a trade to indicate how long an order remains active before it is executed or expires

User must ensure that their computers time has been synced or there will be an error from the received time stamp from an order

In order to have a WebSocket communication between TradeKing and the exchange server, the user must first have an account on either one of these applications in order for the secret and public API keys to open the connection.



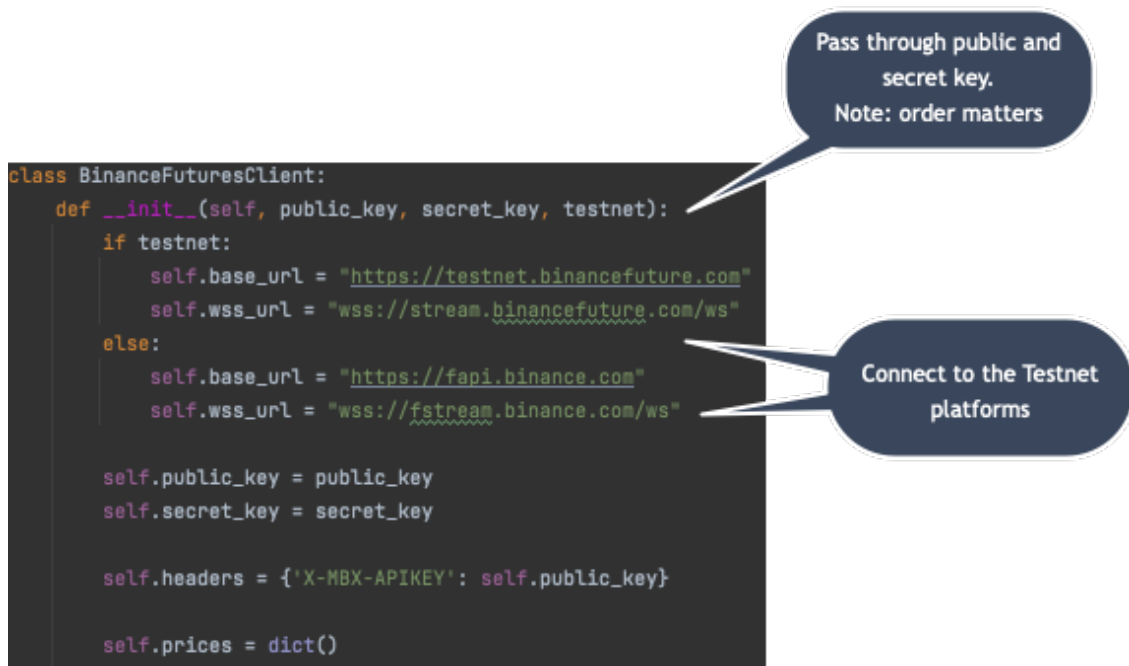
In order for the application to retrieve all the necessary data from the exchange servers, there must be a function created to connect the secret and public keys that are generated on the Binance or BitMex platforms.

```
binance = BinanceFuturesClient("b3c52a9ed9a97",  
                                "ff44bc085cbd8",  
                                True)
```

Public Key

Secret Key

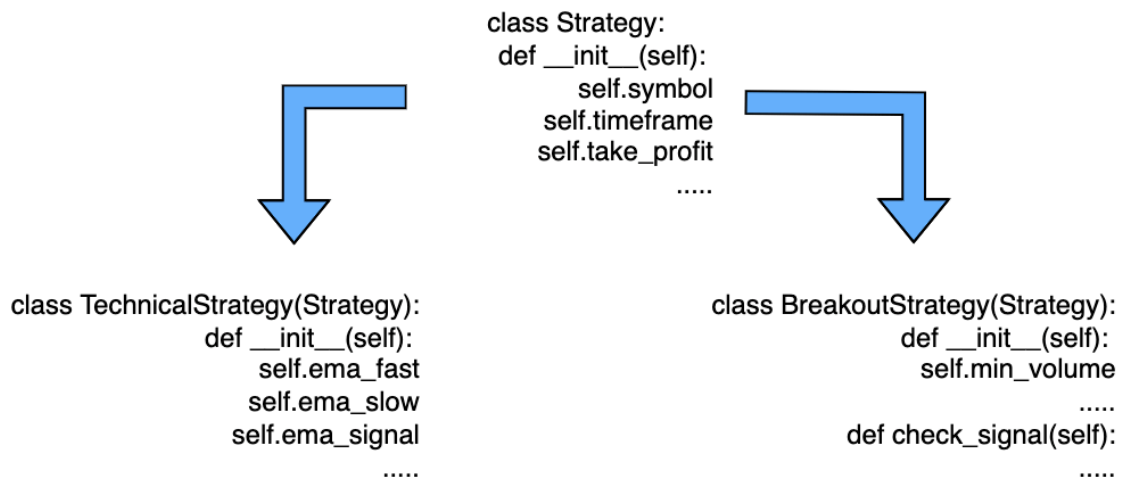




## Creating the Breakout and technical strategies

TradeKing has two strategies that are available to the user. The first one which relies on technical indicators (Technical Strategy) and another one that is based on entering a trade when a certain price level is broken by the current price (Breakout Strategy). There is a strategy object created based on a strategy class that will have attributes to make it unique, such as the contract that is being traded, the timeframe, the take profit percentage, the stop loss percentage, and the balance percentage. These attributes are common to the two strategies, but each strategy will have its particularities.

The most obvious one is that the conditions to enter a trade will be different between the strategies. To handle that there is one class, a strategy class, that will contain attributes and methods that are common to the two strategies. There are then two more strategy classes with specific methods and attributes e.g., the technical strategy class has a method to compute the technical indicators, but these two classes will inherit from the parent strategy class. Allowing for the two child-classes to benefit from the methods of the parent class.



The parent class was created and named strategy. It was initialized with the common parameters for the two strategies like the contract, this required the import of the models.py, the timeframe as a string, the exchange, the balance percentage, floating number, the take profit percentage, floating number as well and finally the stop loss percentage.

Instance variables were created for each of these attributes and both of the two strategies will make use of these attributes. For the child-classes there is a `TechnicalStrategy` that inherits from the `Strategy` class and is initialized with the same parameters with an extra one called “other\_params” that will be specific to the `Technical Strategy` such as the `ema_fast` (Estimated moving average fast).

The two strategies will need candles data in order to calculate whether or not to enter a trade. Since I would want to enter a trade immediately after a strategy has been activated, what can be done is get the historical data right after creating the `Strategy` object. Before doing this there needs to be a variable in the `Strategy` class to store the candlesticks list.

Getting the historical data when initializing the strategy:

```
# Collects historical data is just one API call
# making a query to a database containing billions of rows would freeze the interface.
new_strategy.candles = self._exchanges[exchange].get_historical_candles(contract, timeframe)
```

Could be either Bitmex or binance

Use the historical candles function

Pass the contract and timeframe object

If the list is empty so that the length of the list is equal to 0, this means that there was an error during the API request to either of the exchanges. A log will be displayed stating that there was no historical data received for the desired crypto. Once the strategy has successfully been created and the historical candles have been fetched, there needs to be an accessible place for the new\_strategy object to be stored. Since trades will be triggered by new price data coming in through the websocket channels, it makes sense to store these strategies as a variable in each connector. When the strategy is created to trade on a contract, e.g. on Binance futures, then it will be added to the Binance futures connectors and the same will apply to the Bitmex connectors. In each connect there is a Dictionary variable that has been initialised

```
self.strategies: typing.Dict[int, typing.Union[TechnicalStrategy, BreakoutStrategy]] = dict()
```

Dictionary Type

the body\_index as an integer

Union to indicate two possible values (Breakout or Technical)

In order to turn the strategy off, as the user will need the ability to turn it off once it has begun so that it does not continuously buy and sell crypto's in an infinite loop, the key just needs to be deleted from the dictionary.

```
del self._exchanges[exchange].strategies[b_index]
```

Select the client      Select the variable      Select the Key

Now that the strategy objects have some historical data, the problem is the historical data is not updated and if there is a request for historical data made every time that the application wants to check a signal it would result in the application being too heavy. The solution to this is using the web socket feed to keep building the candlesticks using live data. Candlesticks on a chart are based on “**trade data**” so that every time a trade is made it can change the high or low of a candle.

*Implementing the Bitmex:*

```
self.subscribe_channel("trade")
```

```
if data['table'] == "trade":
    for d in data['data']:
        symbol = d['symbol']
        ts = int(dateutil.parser.isoparse(d['timestamp']).timestamp() * 1000)
        for key, strat in self.strategies.items():
            if strat.contract.symbol == symbol:
                res = strat.parse_trades(float(d['price']), float(d['size']), ts)
                strat.check_trade(res)
```

Pass the timestamp to have unix timestamp in milliseconds

Time stamp in ISO 8601 format

Loop through data Key

Symbol represented by the symbol key

*Implementing the Binance:*

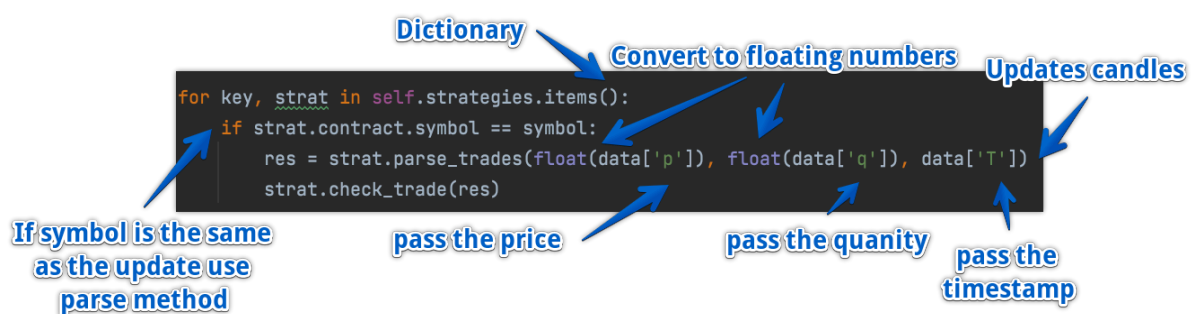
From the binance side the application needs to subscribe to the aggTrade channel. On finance, you can stream a maximum of 200 channels with a single connection. So within the def \_on\_message()



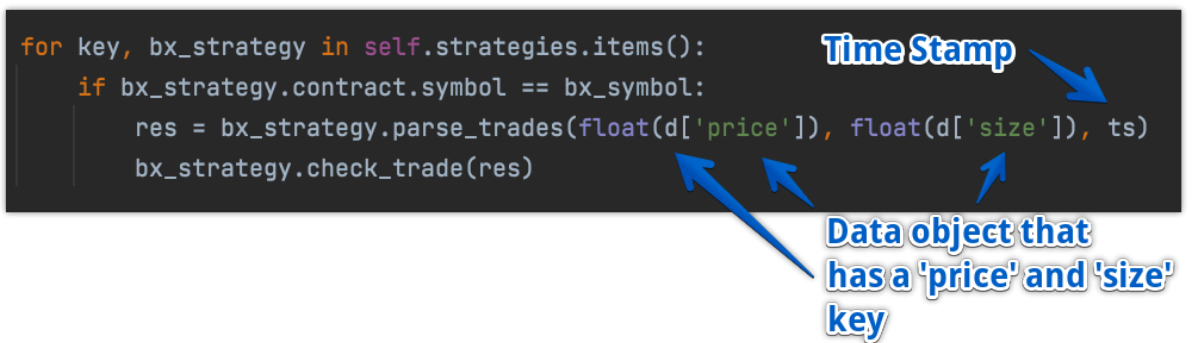
Now that the application has paused the trade coming from the websocket, what should be done next is decide whether this trade will have an influence on the current candle such that if it's a new high or low, or is the first trade of a new candle so that will be the candles opening price. In order to know this method in the strategy, a class needs to be created that will update the candles lists using the new trade data. Inside the strategy class of the strategy module there is a method created to pause the information of the trade to update or to not update the current candle.

In the Binance and Bitmex connectors the application needs to loop through the strategys every time new trade data is received and is done within the `on_message()` function.

***For Binance:***



***For Bitmex:***

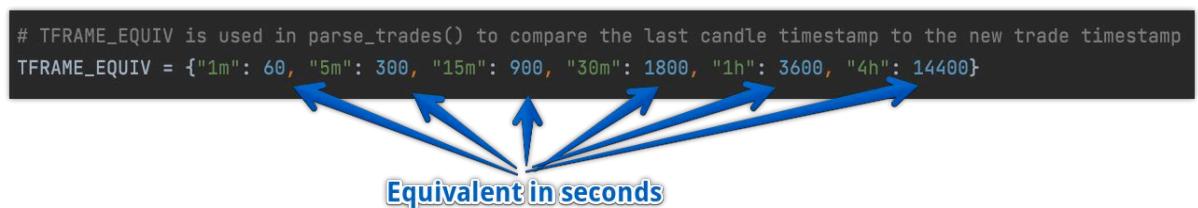


There can be three cases in the `parse_trades()` method:

1. Update the same current candle
2. The trade is the first trade of a new candle
3. The trade is the first trade of a new candle but there are missing candles between the last one recorded and the new candle.

The third case can happen with contracts that don't contain a lot of volume.

Sometimes there are no trades made over a few minutes for a specific crypto coin. In order to know which of these three cases must be used, the application must compare the timestamp of the new trade with the timestamp of the most recent candle that has been recorded and for this the application needs to know the length of one candle in milliseconds, depending on the strategy timeframe. A global dictionary variable is created which is the timeframe equivalent in seconds.



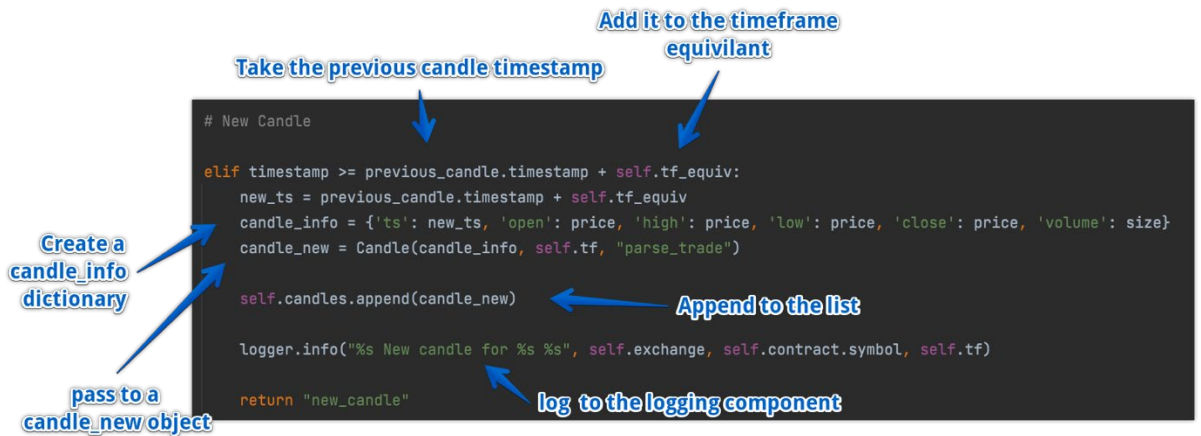
When initializing the strategy there is a variable created for the timestamp equivalent in milliseconds.



The way this works is for example, assuming the timestamps are in seconds. The last candle timestamp that has been recorded is 100 and the timeframe equivalent is 10 seconds. If the application receives an update through the web socket and its timestamp is 107 seconds, it shows that it belongs to the same candle because its timestamp, 107, is less than  $100 + 10$ . In terms of code it can be written in such a way that:



Calculating the new candle timestamp is relatively easy as the application just takes the last candle timestamp and adds the timeframe equivalent in milliseconds



The case where there are missing candles is similar to the new candles method because there still needs to be a calculation to find a new candle but with extra parameters to calculate how many missing candles there are.



Now that there is successfully candlestick data that is updating, thanks to the web socket data, the next step will be to create a method that will check the signal in order



to indicate whether the application should enter a long or short trade or not do anything.

### Long Trades:

- Buy the crypto
- Hold the crypto and wait to sell at a higher price
- Profit from the sale of the asset

### Short Trades:

- Borrow a crypto and sell it
- Wait for a lower price
- Buy the crypto back, make a profit and give it back to the lender (Binance or Bitmex)

This method has the same name within both strategies, however the logic will be different so it is created in the child-classes: *TechnicalStrategy* and *BreakoutStrategy*, and was defined as `check_signal()`. This definition will return an integer, as for a long trade signal the application will return 1, for a short trade signal -1 and if there is no signal it returns 0.

### *BreakoutStrategy:*

The breakout strategy is relatively simpler than the technical strategy as it only needs one candle and there is no need to compute any indicators.

```
def check_signal(self) -> int:
    # 1 = long signal
    # -1 = short signal
    # 0 = No signal
    if self.candles[-1].close_price > self.candles[-2].high_price and self.candles[-1].volume > self.min_volume:
        return 1
    elif self.candles[-1].close_price < self.candles[-2].low_price and self.candles[-1].volume > self.min_volume:
        return -1
    else:
        return 0
```

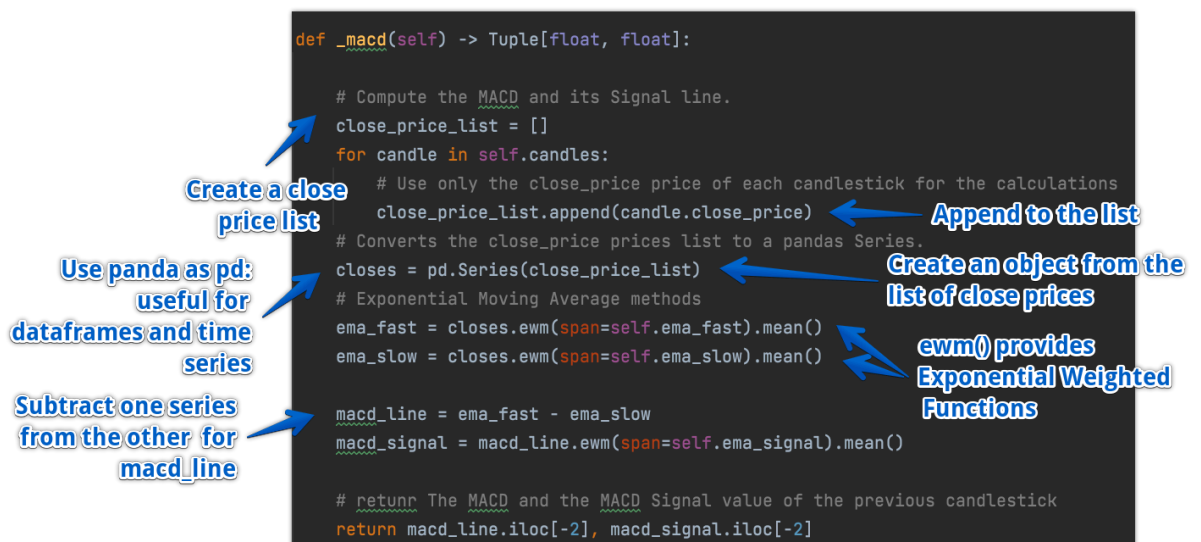
Useful to implement logic based on candle stick patterns

### TechnicalStrategy:

The Technical Strategy is far more complex as the candle sticks are used to calculate indicators. There are two more indicators from the breakout strategy, thus there shall be an implementation of two more methods, the `macd()` and the `rsi()`, relative strength index. There are four steps to calculate the MACD:

1. Fast EMA (Exponential Moving Average) calculation
2. Slow EMA calculation
3. Fast EMA – Slow EMA
4. EMA on the result of #3

When calculating the Slow EMA, it is considered slow because it uses more candles than the Fast EMA thus the slope does not change as quickly. After each step the required product is not a single figure, but rather a list of exponential moving averages, with each EMA corresponding to each candlestick that has been recorded. So the EMA of the current candle and as well as the one before and so on. The application already has the candles, what is now missing is a function to compute the exponential moving averages based on the close price of each candle.

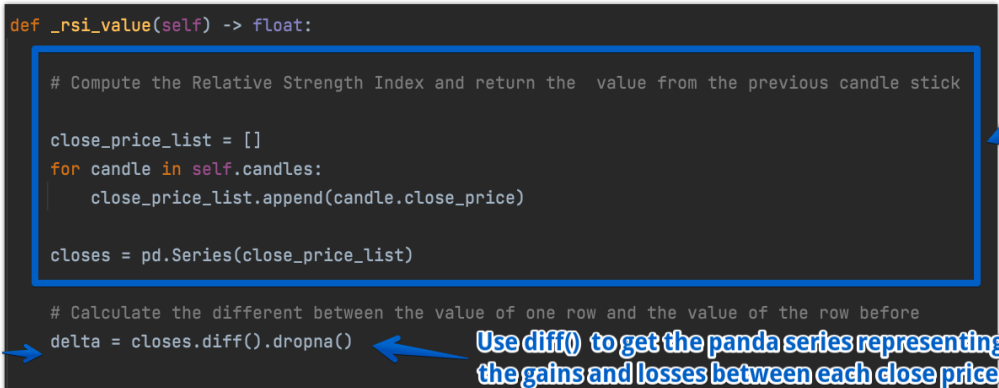


```
def _macd(self) -> Tuple[float, float]:  
  
    # Compute the MACD and its Signal line.  
    close_price_list = []  
    for candle in self.candles:  
        # Use only the close_price price of each candlestick for the calculations  
        close_price_list.append(candle.close_price)  
    # Converts the close_price prices list to a pandas Series.  
    closes = pd.Series(close_price_list)  
    # Exponential Moving Average methods  
    ema_fast = closes.ewm(span=self.ema_fast).mean()  
    ema_slow = closes.ewm(span=self.ema_slow).mean()  
  
    macd_line = ema_fast - ema_slow  
    macd_signal = macd_line.ewm(span=self.ema_signal).mean()  
  
    # returnr The MACD and the MACD Signal value of the previous candlestick  
    return macd_line.iloc[-2], macd_signal.iloc[-2]
```

Annotations:

- Create a close price list (points to `close_price_list = []`)
- Append to the list (points to `close_price_list.append(candle.close_price)`)
- Use panda as pd: useful for dataframes and time series (points to `closes = pd.Series(close_price_list)`)
- Create an object from the list of close prices (points to `closes = pd.Series(close_price_list)`)
- ewm() provides Exponential Weighted Functions (points to `closes.ewm(span=self.ema_fast).mean()` and `closes.ewm(span=self.ema_slow).mean()`)
- Subtract one series from the other for macd\_line (points to `macd_line = ema_fast - ema_slow`)

The MACD indicator is based on the close price. Instead of calculating it on every new trade, the application only calculates it when there is a new candle, so the application is only interested in the MACD of the last finished candle. Finally, the MACD\_signal variable was created, which represents the moving average of the MACD line. The signal depends on whether the MACD line is above or below the MACD signal line. There will be a long signal when the MACD line goes above the signal line, and so the application is to return the elements from this method using a Tuple where a Tuple[x, y] is a cross product of X and Y. The MACD line and signal of the previous candle is returned using -2, as -1 is a representation of the new candle.



```
def _rsi_value(self) -> float:
    # Compute the Relative Strength Index and return the value from the previous candle stick
    close_price_list = []
    for candle in self.candles:
        close_price_list.append(candle.close_price)
    closes = pd.Series(close_price_list)

    # Calculate the different between the value of one row and the value of the row before
    delta = closes.diff().dropna()
```

Annotations:

- Series based on close price**: Points to the `closes = pd.Series(close_price_list)` line.
- Use diff() to get the panda series representing the variations, the gains and losses between each close price**: Points to the `delta = closes.diff().dropna()` line.
- Still using the the close prices and the pandas series object**: Points to the `close_price_list` and `closes` variables.

After the `diff()` method, the `dropna()` method is applied because the variations can only be applied once there are at least two values and for the first line of the series, there won't be any values. The next step is to separate the upside variations from the downside variations. Two copies of the Delta pandas series are created, one for up and one for down. For the up all the rows that are under zero are filtered, these are the losses. This is to populate only the gains. The opposite is then done for the downside to obtain all the losses.



```
up, down = delta.copy(), delta.copy()
up[up < 0] = 0
down[down > 0] = 0 # Keep only the negative change, others are set to 0
```

Annotations:

- Create two copies**: Points to the `up, down = delta.copy(), delta.copy()` line.
- Only positive number**: Points to the `up[up < 0] = 0` line.
- Only negative numbers**: Points to the `down[down > 0] = 0` line.
- Initialise both to 0**: Points to the `up[up < 0] = 0` line.

Next, an exponential moving average is used on the two panda series in order to calculate the average gains as well as the average losses. The relative strength index (RSI) is returned using the `iloc[]` method to select the row before the most recent one.

```
gain_avg = up.ewm(com=(self._rsi_length - 1), min_periods=self._rsi_length).mean()
loss_avg = down.abs().ewm(com=(self._rsi_length - 1), min_periods=self._rsi_length).mean()

relative_strength = gain_avg / loss_avg

rsi = 100 - 100 / (1 + relative_strength)
rsi = rsi.round(2)

return rsi.iloc[-2]
```

**Calculations for RSI that correspond to TradingView charts**

**Like MACD return the candle that is before the current one**

## Executing Trades

Executing a trade is defined within the strategy parent class, as both the technical and breakout strategies will inherit from this definition.

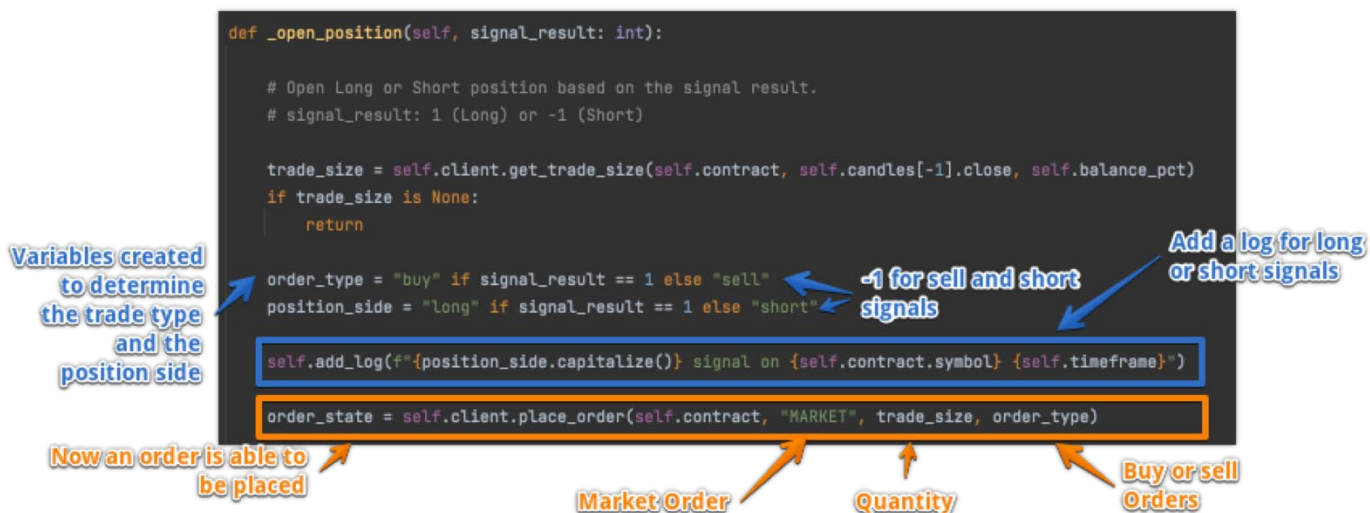
When the `on_position()` method is being called it gets the trade size and calls the `get_balance()` method, and places the order, resulting in two REST API calls which can cause some delay, resulting in the warning message to appear.

```
timestamp_diff = int(time.time() * 1000) - timestamp # multiply by 1000 to get milliseconds
if timestamp_diff >= 2000:
    logger.warning("%s %s: %s milliseconds of difference between the current time and the trade time",
                  self.exchange, self.contract.symbol, timestamp_diff)

previous_candle = self.candles[-1]
```

If the `order_state` is not none, that means that the request was successful and that the order is placed. The `ongoing_position` variable is set to true and is extremely important this step is carried out, otherwise the program will keep attempting to place an order. When placing a market order there are two possibilities. The first case is that the order is immediately executed, so that the `order_state = "filled"`. On both Binance

and Bitmex exchanges, it returns the word “filled” to indicate whether the order was filled or not, but some different exchanges may return another word. For example, the word could be “executed” and for future work if another exchange was to be implemented this would need to be taken into consideration. Otherwise the order needs a few seconds to be executed, especially in the circumstance where the order size is big for the market. In this case where the trade is not executed immediately at the price, it calls the `get_trade_order_status()` every 2 seconds until the order status is “filled” and the application can get the execution price or the “fill” price. In order to do this, the time object of the threading library was utilised which allows the ability to call a function ,with a delay, without blocking the `open_position()` definition.



In both cases of technical and breakout, the trade is recorded to a list that is created as a trade model. It is easier to access the information about a trade within the model rather than having a dictionary.

```

if order_state is not None:
    self.add_log(f"{order_type.capitalize()} order placed on {self.exchange} | Status: {order_state.status}")

    self.ongoing_position = True

    average_fill_price = None

    if order_state.status == "filled":
        average_fill_price = order_state.average_price
    else:
        tmr = Timer(2.0, lambda: self._check_trade_order_status(order_state.order_id))
        tmr.start()

    new_trade = Trade({"time": int(time.time()) * 1000, "entry_price": average_fill_price,
                      "contract": self.contract, "strategy": self.strategy_name, "side": position_side,
                      "status": "open", "pnl": 0, "quantity": order_state.executed_qty,
                      "entry_id": order_state.order_id})
    self.trades.append(new_trade)

```

Average execution price

Timer as a Thread

Trade Model

start the timer

append to a list of Trade objects

Trade Model

```

class Trade:
    def __init__(self, trade_info): #Trade_info is a dictionary
        self.time: int = trade_info['time']
        self.contract: ExchangeContract = trade_info['contract']
        self.strategy: str = trade_info['strategy']
        self.side: str = trade_info['side']
        self.entry_price: float = trade_info['entry_price']
        self.status: str = trade_info['status']
        self.pnl: float = trade_info['pnl']
        self.quantity = trade_info['quantity']
        self.entry_id = trade_info['entry_id']

```

Finally, what is left to be created is the `check_trade_order_status()`, where the `order_id` is passed and the `get_order_status()` is called if the request is successful. Then status of the order is logged and if the `order_status` attribute is filled, the program loops through already placed trades where the `trade.entry_id` is equal to the `order_id`.

```

def _check_trade_order_status(self, order_id):

    # Called after an order has been placed, continues until it is filled.

    trade_order_status = self.client.get_order_status(self.contract, order_id)

    if trade_order_status is not None:

        logger.info("%s order status: %s", self.exchange, trade_order_status.status)

        if trade_order_status.status == "filled":
            for trade in self.trades: # Loop through the trades - it may not be the only one placed
                if trade.entry_id == order_id:
                    trade.entry_price = trade_order_status.average_price
                    trade.quantity = trade_order_status.executed_qty
                    break
            return

    tar = Timer(2.0,
                lambda: self._check_trade_order_status(order_id)) # Check every 2 seconds until order status is filled
    tar.start()

```

Infinitive loop until order is filled

## Exiting a Trade

A method is created to check whether the take profit or stop loss has been reached. The idea is to check for simple conditions whether the take profit has been triggered or is the stop loss has been triggered. By default they are set to be False. The current close price is compared to the trade entry price.

```

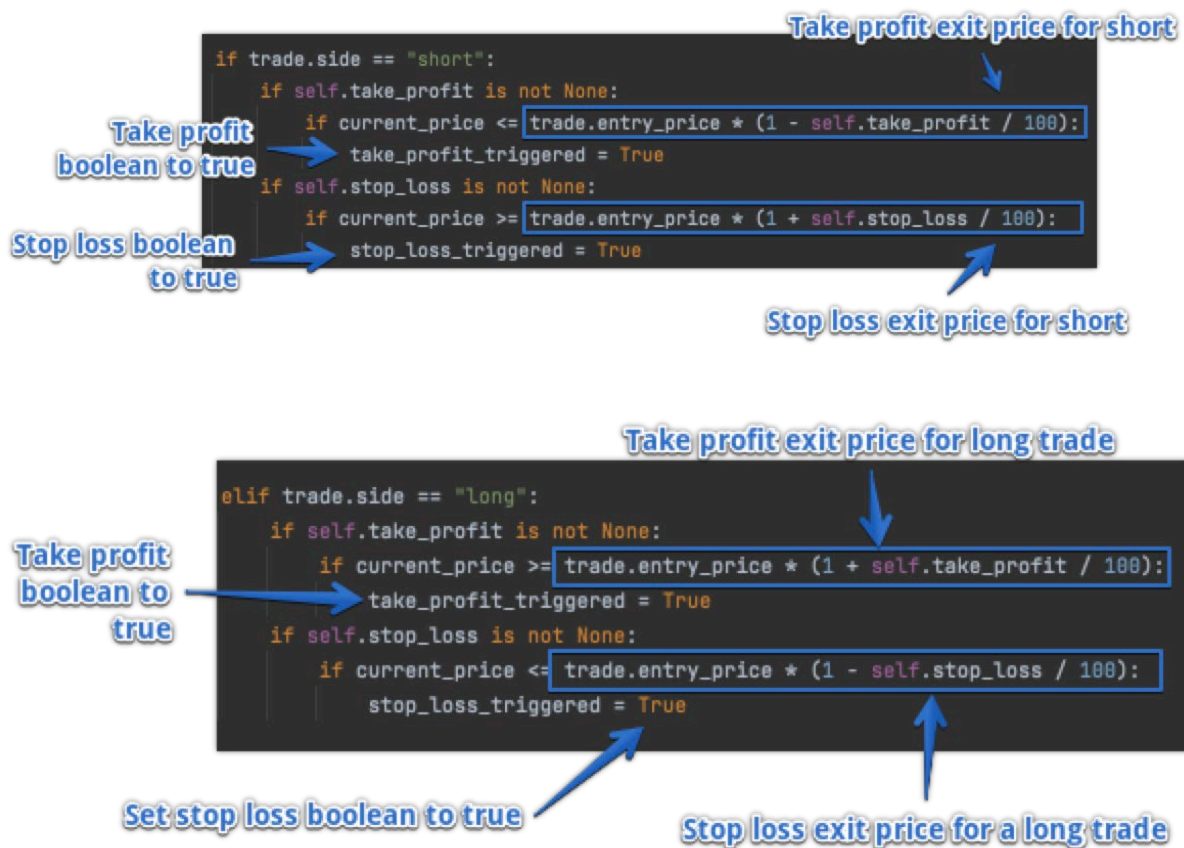
def _check_tp_sl(self, trade: Trade):

    # Based on the average entry crypto_price and calculates
    #If the defined stop loss or take profit has been acheived.
    current_price = self.candles[-1].close

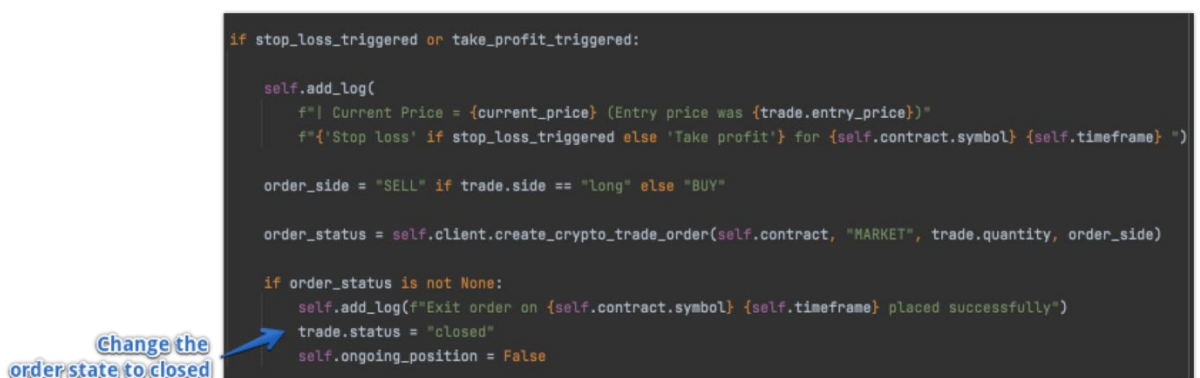
    stop_loss_triggered = False
    take_profit_triggered = False

```



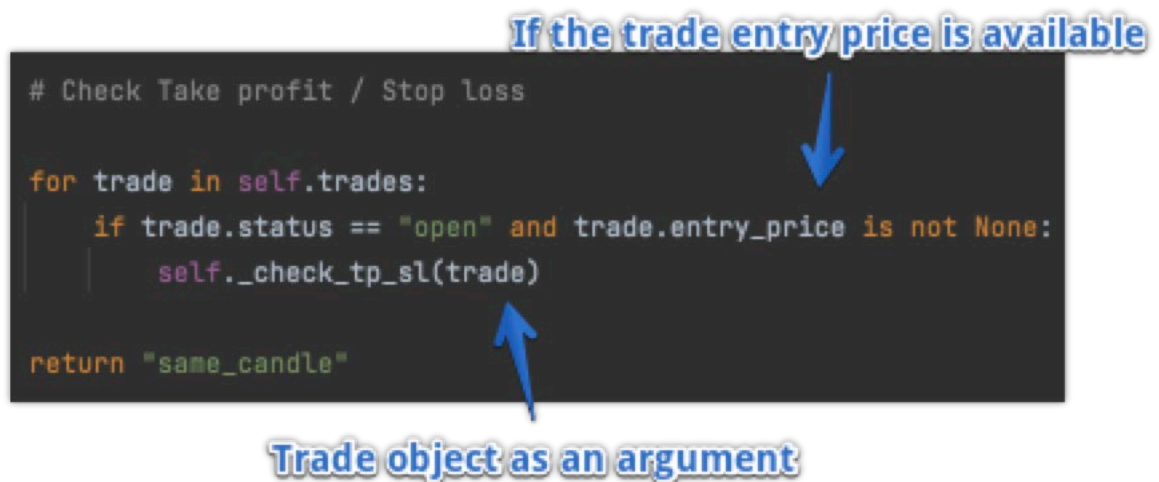


If the stop loss or the take profit is triggered then a log message is displayed to indicate to the user what is happening with the trade. If there was a long trade then the order side is going to be sell. Otherwise there will be a buy order to close the short position. The client is used to place a market order again.





This method is then called every time that there is a trade update for the same candle.



The image shows a Python code snippet on a dark background. The code is as follows:

```
# Check Take profit / Stop loss

for trade in self.trades:
    if trade.status == "open" and trade.entry_price is not None:
        self._check_tp_sl(trade)

return "same_candle"
```

There are two blue arrows pointing to specific parts of the code. The first arrow points from the text "If the trade entry price is available" to the condition `trade.entry_price is not None`. The second arrow points from the text "Trade object as an argument" to the argument `trade` in the function call `self._check_tp_sl(trade)`.

#### Creating the Crypto comparison graphs

An extra feature of the TradeKing application is the ability for a user to visualize the trends of a specific cryptocurrency with the help of matplotlib. Matplotlib is a plotting library for the python programming language and its numerical mathematics extension NumPy (Lemenkova, 2020). A user will be able to select a cryptocurrency, a look back period in terms of days and the interval, as well as the currency to be used when displaying the current value of the cryptos. Once the user has entered all these requirements the application will draw the graph, where a user has the option to take further actions on the graph, such as draw another cryptocurrency trend and compare the two, or to reset it. For this to be possible, TradeKing needs to make use of historical data and the way TradeKing works is that when entering the graphs page of the application, the websocket connections of the TradeKing bot between the exchanges are closed and a new tkinter window is opened with just the graphs page stacked on the frame. This is done to keep the application lightweight as tkinter works by pre loading everything onto a canvas and the way to navigate through is to push frames to the surface while still running everything in conjunction underneath. The solution was to break the `mainloop()` of the TradeBot and simultaneously create a new loop. The new loop is used to generate and display trading graphs for a user without

the hassle of trying to conform everything into a parent frame, while at the same time only keeping websockets open when on the TradingBot. After research into public APIs, the solution was to use a well-know crypto website coingecko. Using the URL <https://api.coingecko.com/api/v3/coins> this API endpoint shows all the available cryptos in the coingecko database with their historical data.

**Append to  
crypto\_ids  
the current  
Id of the  
loop**

```
# retrieve the available cryptos
def available_crypto():
    url = f'https://api.coingecko.com/api/v3/coins'
    response = requests.get(url)
    data = response.json()

    crypto_ids = []
    for asset in data: crypto_ids.append(asset['id'])

    return crypto_ids
```

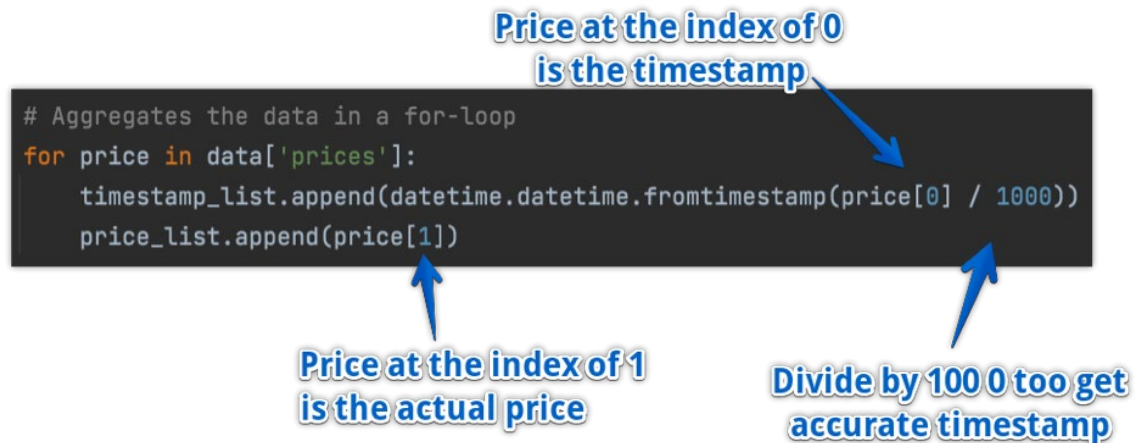
**Create an  
empty List**

Once all the available cryptos from the coin gecko website had been retrieved and stored in a cryptos\_id list, the next step is to retrieve the chart data for any currency and return it as a DataFrame. The first step is to create a definition function that was named `get_market_chart` and pass through the 4 parameters that the user will be able to customize when creating these crypto trend charts, the 4 parameters consist of:

1. `coin_id` (Used to identify which crypto the user wants to see the trends for)
2. `vs_currency` (Used to display crypto prices in a specified currency)
3. `days` (Used as a look back period for the crypto from the present day)
4. `interval` (Used for plotting frequency)

An API request is then implemented to check if the requested crypto currency is in the list of the `coin_ids`. To do this a `crypto_ids` is first created and assigned to the `available_cryptos()` function where after an if/else statement is created to check whether the given `coin_id` is within the `crypto_ids` list. If the `coin_id` is within the list, then the program proceeds to make another URL request to retrieve the `market_chart` data for that specific `coin_id` and the API call will take the form of `f'https://api.coingecko.com/api/v3/coins/{coin_id}/market_chart'`. In order for a graph

to be drawn there needs to be a list for the timestamp, which will be applied to the X-Axis and a price that will be applied to the Y-Axis created. A for loop is then used to aggregate the data into these respective lists.



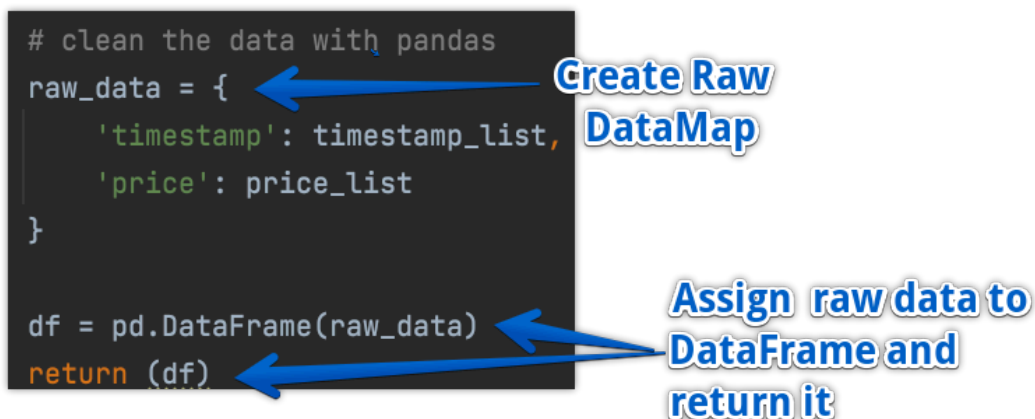
```
# Aggregates the data in a for-loop
for price in data['prices']:
    timestamp_list.append(datetime.datetime.fromtimestamp(price[0] / 1000))
    price_list.append(price[1])
```

Price at the index of 0 is the timestamp

Price at the index of 1 is the actual price

Divide by 1000 too get accurate timestamp

In order to create the required data frame the data must first be cleaned using a python package called pandas. Pandas is a software library that is written for the python programming language for data manipulation and analysis, in particular it offers data structures and operations for manipulating numerical tables and time series (Molin, 2019). Firstly, a raw data map was created with 2 inputs of timestamp and price and once this raw data map had been created, it then can be passed and processed using pandas to create a DataFrame that will be returned so the function can be used to return the data that was obtained via the API.



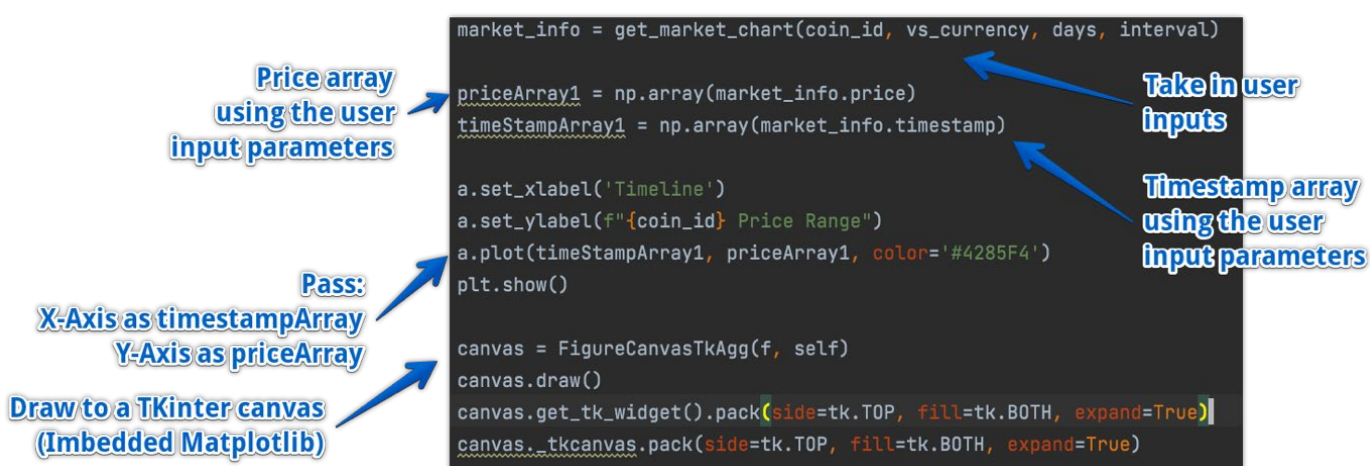
```
# clean the data with pandas
raw_data = {
    'timestamp': timestamp_list,
    'price': price_list
}

df = pd.DataFrame(raw_data)
return (df)
```

Create Raw DataMap

Assign raw data to DataFrame and return it

As TradeKing is a program that is hosted on the python GUI package, Tkinter, the Matplotlib is required to be imbedded into the Tkinter window in order for it to be a part of the TradeKing application. The use of Matplotlib was incorporated because after some extensive research into various data visualization tools, such as Plotly, it was discovered that Tkinter does not make it easy to embed this visualisation tool within their frames. The documentation for such processes are scarce which is why Plotly, for example, is not supported by Tkinter but Matplotlib is. Generally in order for a DataFrame to be plotted onto a Matplotlib canvas, the `get_market_chart()` function would be assigned to a variable and then using the Matplotlib `.plot` and `plt.show()` functions, would result in a graph appearing and displaying all the drawn datapoints. However, when translating this process to the imbedded Matplotlib canvas in a Tkinter window, the results would show an accurate range along the X and Y axis's but would fail to draw the points upon the canvas. The problem originated from the DataFrame having two dimensions for both the Price and Timestamp where the first column would be for the index of the DataFrame and the second for the actual data that is to be plotted and this caused confusion within the program. The solution to this was to transform the specific DataFrame into two separate NumPy Arrays and then pass them through as a list of datapoints when plotting on the canvas. Once this could be successfully plotted, the code was then modified to take in user arguments when retrieving the data set in order to make the graphs that are being created customisable by the user.

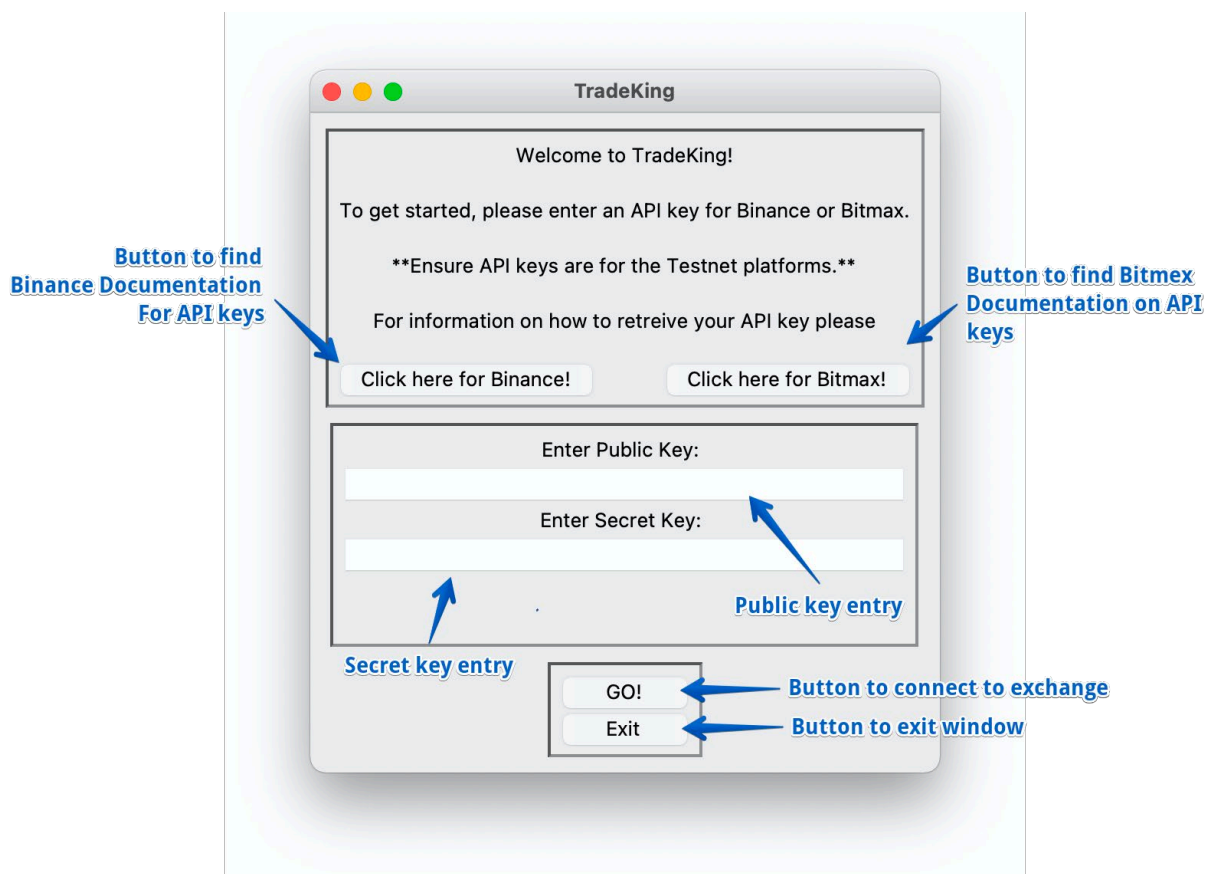


An extra feature these graphs have is to layer two different crypto currencies to contrast and compare. Once the user data was successfully drawn upon an imbedded

Matplotlib graph, the next step was to split the Y axis in half in order to have a comparison of the two cryptocurrencies. The price range between two cryptocurrencies could be so significantly different that it would be impossible to represent both upon the same axis, E.g. Bitcoin that's value is estimated to be as off today in the range of 45,000 - 50,000, compared to Shiba that has a value of 0.0005-0.0008 dollars as of the current day. These ranges are so significant that splitting the Y-Axis is mandatory.

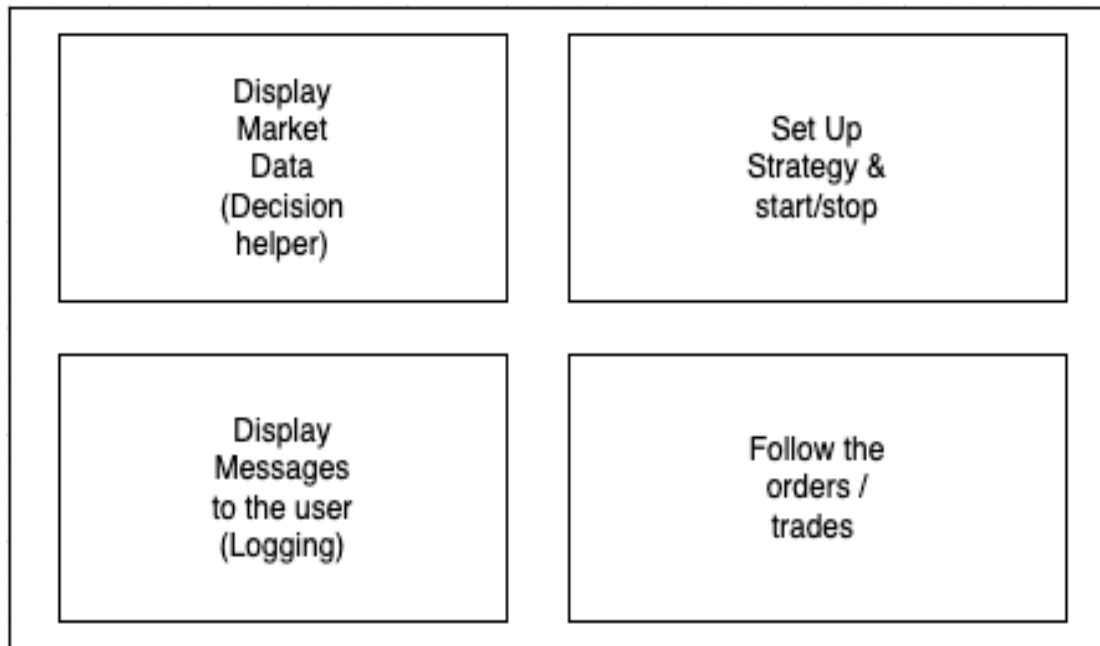
## Graphical User Interface (GUI)

When a user first opens TradeKing they are shown a simple GUI that prompts them to input their public and secret API keys. If the user does not have these then there are hyperlink buttons at the top where they can follow them to the documentation for the respective platforms.

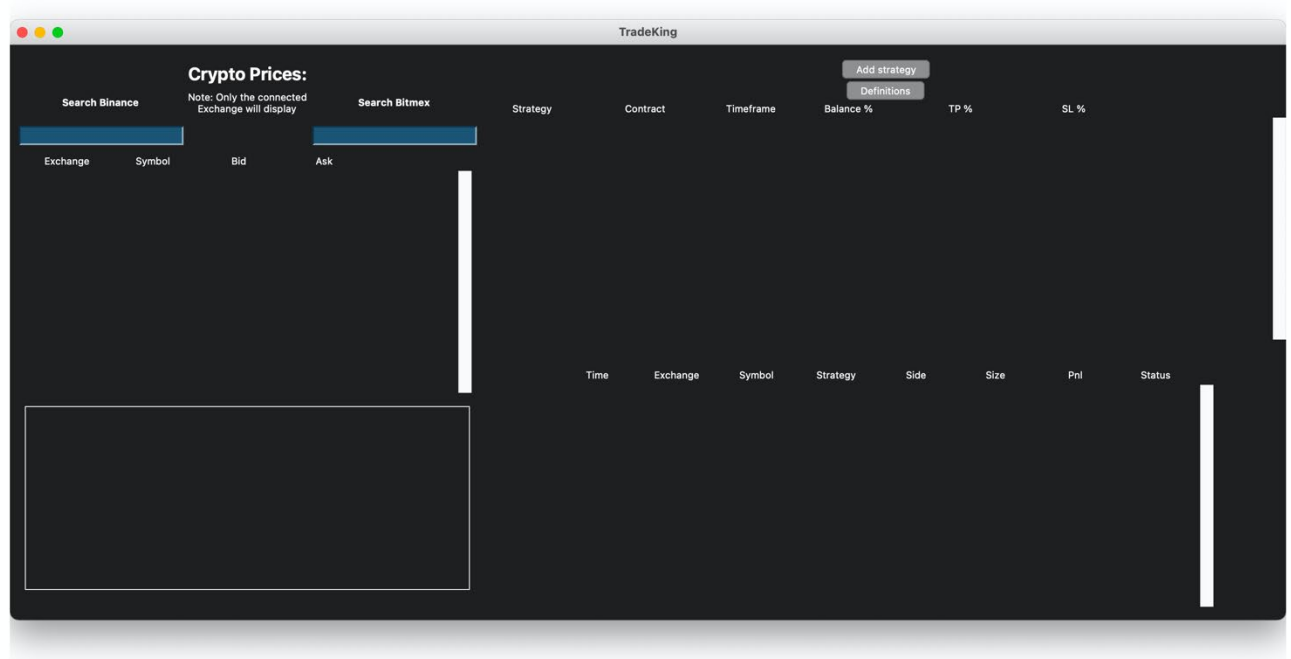


Once the user has logged in they will be redirected to the main trading window. From the trading window perspective, there are four factors that are needed by any trading application. First, to display live market data, especially in the case when a semi-automated trading tool is built where the trader prefers to keep some control over the trading bot. For example, allowing the program to open a BTCUSDT position for the user, but the user would still prefer to close the position themselves when specific market conditions are met, when there has been some news release, or any other conditions that the user has defined that are required to be checked manually. Then the program will want to inform the user about what is going on inside the application so that it is not just a “black box” that buys and sells without the user’s knowledge. This process of informing the user is referred to as logging.

Once the market data is obtained then the program should make decisions about opening or closing a position based on a predefined logic. However, this logic will require some input from the user e.g. the stop loss percentage, the percentage of balance to allocate to the strategy etc. The user will also want to be able to stop and start a specified strategy at any point in time. Once an order has been placed or an order has occurred then the user will want to follow what is happening to a position and the current status e.g. what the PnL for that trade may be, is the position opened or closed etc. This leaves 4 main components required for the trade king application, which will be coded in 4 separate python files and displayed as the following:



Upon logging in to trade king the user will see the skeleton of the application.



The user has 3 actions that can currently be performed in this stage.

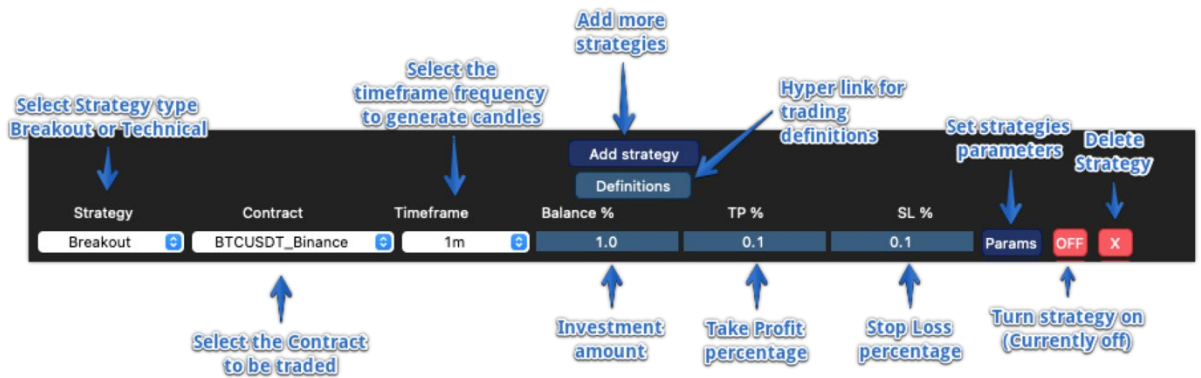
1. The user can type into the search bar for the crypto prices for the exchange that they are connected to.
2. The user can click on the definitions button to be directed to an online glossary that holds all trading terminology and definitions.
3. Finally the user can press on the “Add Strategy Button”.

Once a user has pressed the add strategy button, various inputs expand allowing for the user to toggle between the strategy type as there are two strategy types, one being technical and the other being breakout. The user then selects the contract in which they will want to trade in, this being for example ,BTCUSDT\_Binance, which will mean the user is trading in bitcoin to USTD using the Binance connections. A user then will have the options to change the timeframe for the frequency of the candlesticks as the difference between the current candlestick and the previous could have a timeframe interval of:

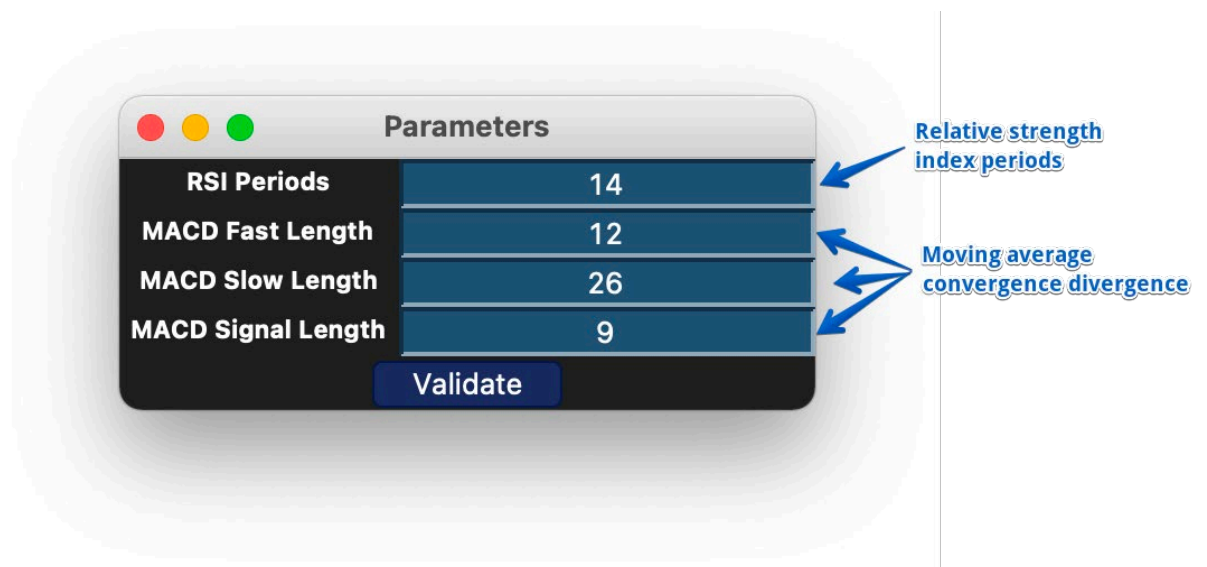
- 1 min
- 5 min
- 15 min
- 30 min
- 1 hr
- 4 hr

Once the user has selected the desired timeframe, they are then asked to input the percentage of their balance that they want to invest into their strategy. Users also need to input their take profit percentage and their stoploss percentage, along with setting parameters before they are able to turn on their trading strategy. There is also an X button used to delete the strategy.

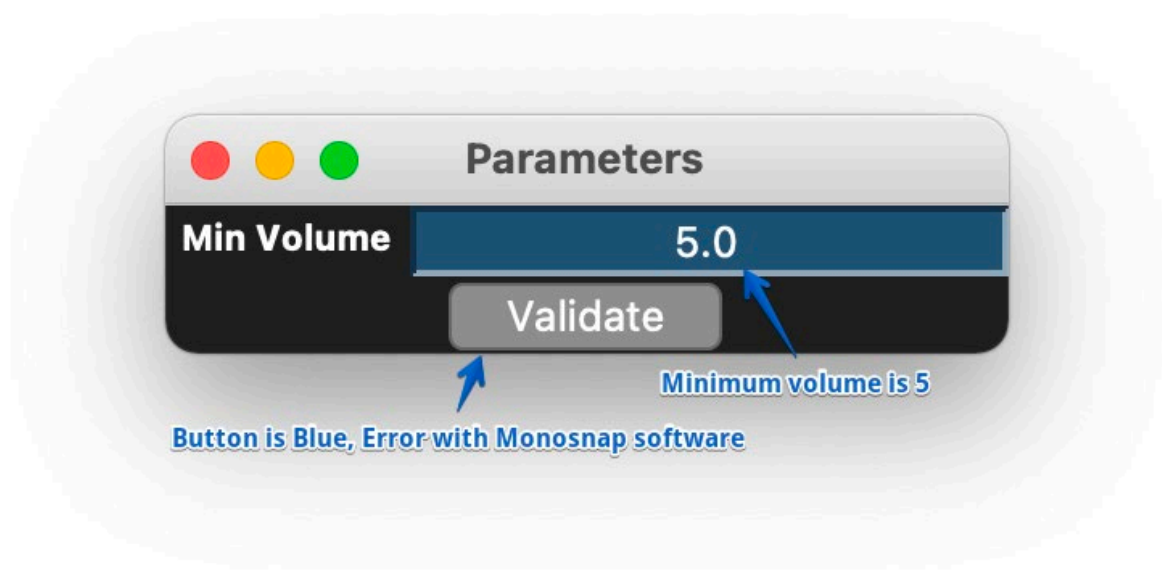




The parameters that are required by the application for the users to input differ according to what strategy they are using. The technical strategy relies on technical indicators to generate trading signals. The parameters needed are the RSI periods, MACD fast length, the slow length, and finally the signal length. Using these parameters within the technical strategy to enter trades has been explained in the implementation section of this report.

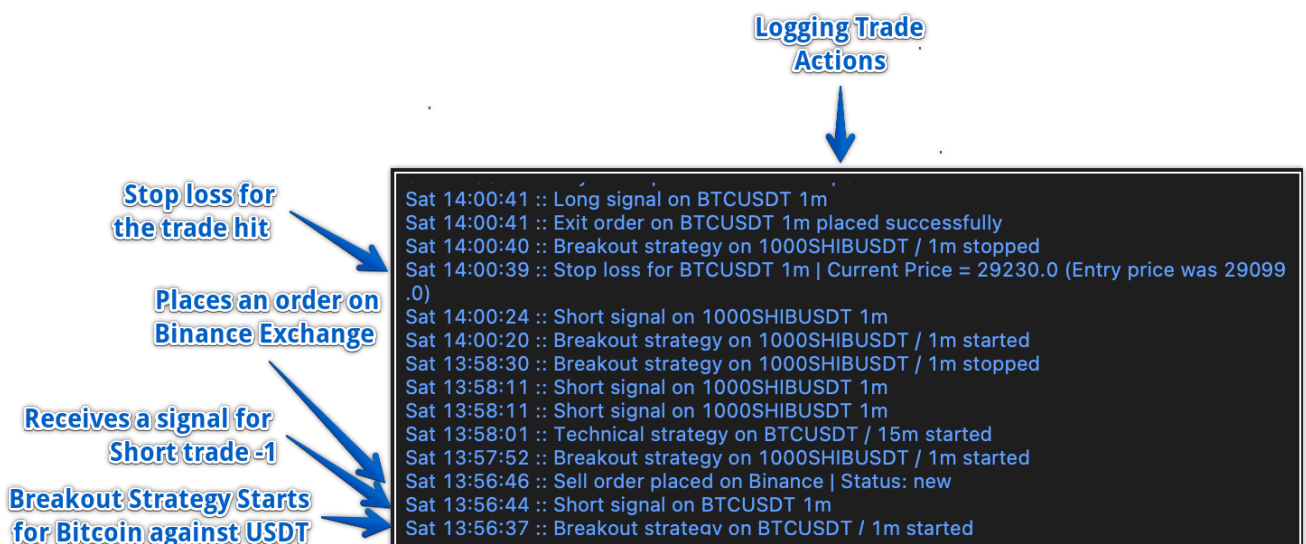


The breakout strategy relies on less input from the parameters due to the breakout strategy only waiting for breakout and trade signals to indicate price movement and entry positions. The parameters require only the minimum trading volume.



After the user has correctly set up their trading strategy they will then click the “off” button, as it indicated before being pressed that the strategy is off. Once pressed the off button will turn to green and say “on” to indicate that the strategy is running. All the input fields are thereby locked to avoid the user from editing a strategy while it is running.

At the bottom left frame of the window there is a logging component that notifies the user about the actions they are taking when creating the strategy e.g. When a strategy has started, receiving signals to enter a trade ect.



When trades have been made there is a trades watch component located at the bottom right of the window that records all the trades from all the strategies running in Realtime with a calculated profit and loss. This feature is very useful for the user as it shows all the trades separately with the status of each trade clear at the end so a user doesn't get lost with what the trading bot is currently doing.

Time	Exchange	Symbol	Strategy	Side	Size	Pnl	Status
May 14 13:56	Binance_futures	BTCUSDT	Breakout	Short	0.001	-0.13	Closed
May 14 14:00	Binance_futures	BTCUSDT	Breakout	Long	0.001	-0.10	Closed
May 14 14:00	Binance_futures	BTCUSDT	Breakout	Long	0.001	0.02	Closed
May 14 14:01	Binance_futures	BTCUSDT	Breakout	Long	0.001	-0.07	Closed
May 14 14:02	Binance_futures	1000SHIBUSDT	Breakout	Long	2446.0	-0.369346	Closed
May 14 14:03	Binance_futures	BTCUSDT	Breakout	Short	0.001	0.04	Closed
May 14 14:03	Binance_futures	1000SHIBUSDT	Breakout	Short	2469.0	-0.424668	Open

Trade Size = (current\_balance \* balance\_percentage / 100) / price

The last component of this window is the Crypto watchlist component where the user is able to see the live prices of all the available crypto currencies on the connected exchanges. There is 2 input windows allowing for the user to type a cryptocurrency pairing, however only the exchange the user had registered and logged in using the API keys will display coins. Once the user types there is an auto complete widget that is not part of the tkinter framework and had to be created. To use the autocomplete widget the user must navigate down using the arrow on their key board and then use the right arrow to autofill the coin name to the input box and destroy the autocomplete widget. Once the coin is correctly inputted the user then presses enter for the application to subscribe to the bookTicker current price and the coin is then added to the watchlist.

Using the  
autocomplete  
widget to select  
TOMOUSDT

### Crypto Prices:

Note: Only the connected Exchange will display

Search Binance		Search Bitmex	
T			
	THETAUSDT	Bid	Ask
	TLMUSDT	46.0900	46.8900
	<b>TOMOUSDT</b>	28814.20	28827.20
	TRBUSDT	1962.29	1962.73
	TRXUSDT	0.4412	0.4556
		0.004282	0.005734
Binance	YFIUSDT	9662.0	9749.0
Binance	RENUSDT	0.14610	0.15720
Binance	LTCUSDT	64.32	64.38
Binance	WOOUSDT	0.17636	0.20029
Binance	MKRUSDT	1406.50	1424.80

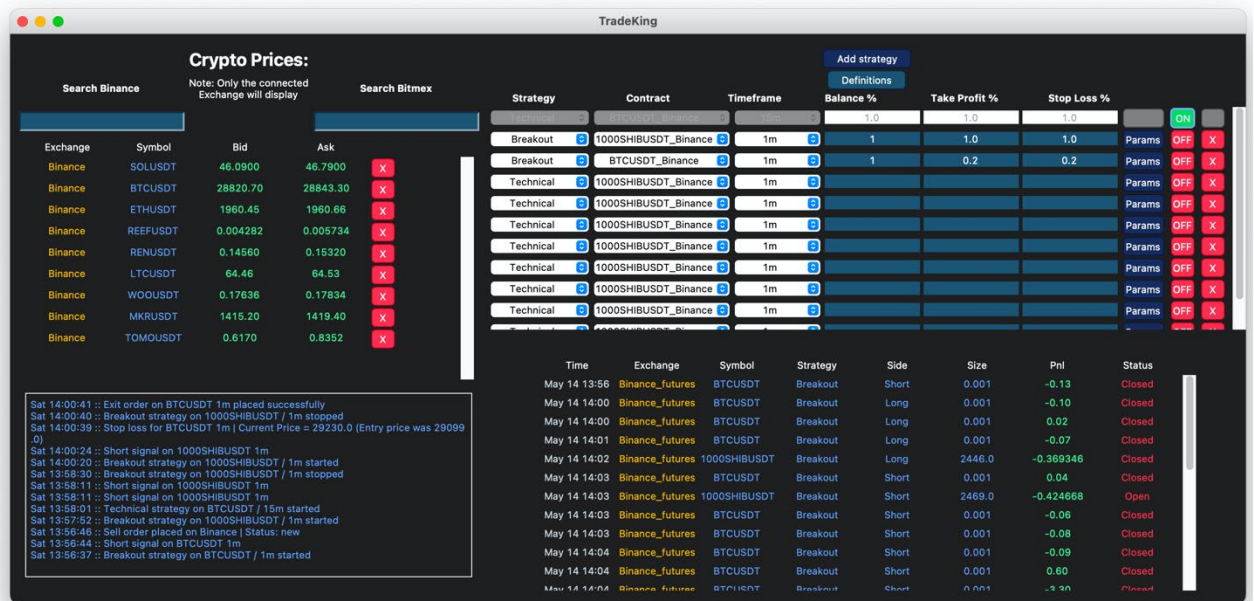
Coin added to the  
watchlist

### Crypto Prices:

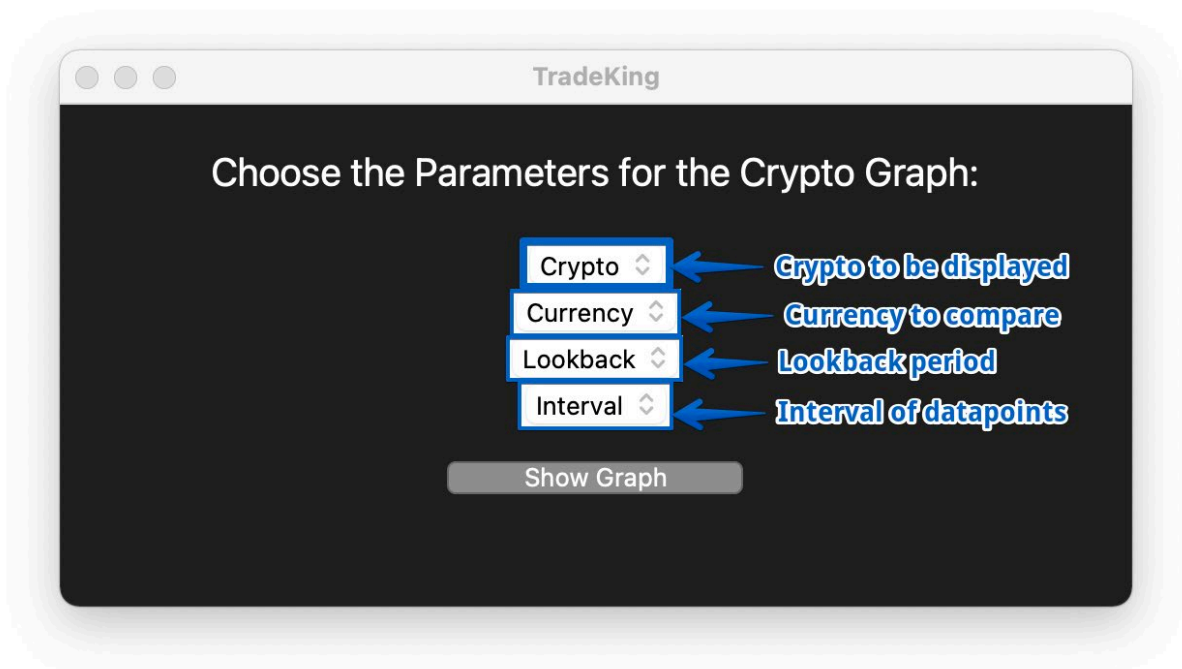
Note: Only the connected Exchange will display

Search Binance		Search Bitmex	
Exchange	Symbol	Bid	Ask
Binance	SOLUSDT	46.0900	46.7800
Binance	BTCUSDT	28835.10	28846.10
Binance	ETHUSDT	1963.99	1964.00
Binance	REEFUSDT	0.004282	0.005734
Binance	RENUSDT	0.14570	0.15300
Binance	LTCUSDT	64.55	64.62
Binance	WOOUSDT	0.17636	0.20029
Binance	MKRUSDT	1420.50	1424.20
Binance	<b>TOMOUSDT</b>	0.6170	0.8352

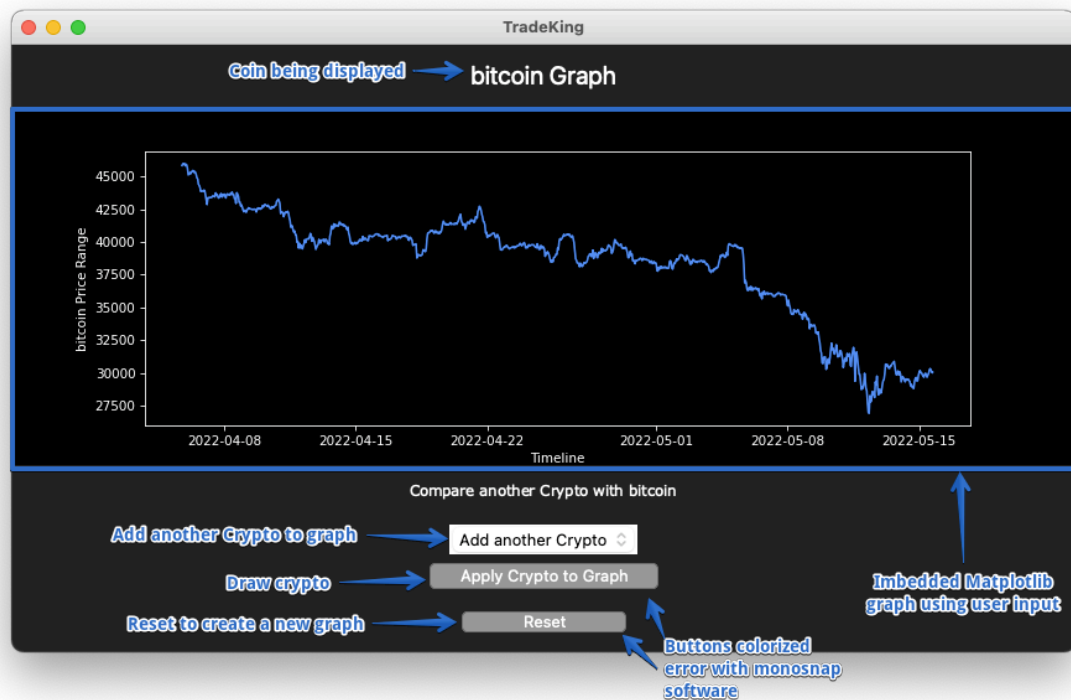
After all of these components have been created they are then packed into the split frames all available on a single window for the users convenience.



The next graphical user interface that is offered by TradeKing is the ability for a user to review crypto currencies price trends in a visual format using an imbedded matplotlib graph in a Tkinter window. The user will need to navigate to the top his window and select the graphs tab where they will then be redirected to a new Tkinter window that has user inputs required to build a graph using historical data gained from the coingeckoapi.

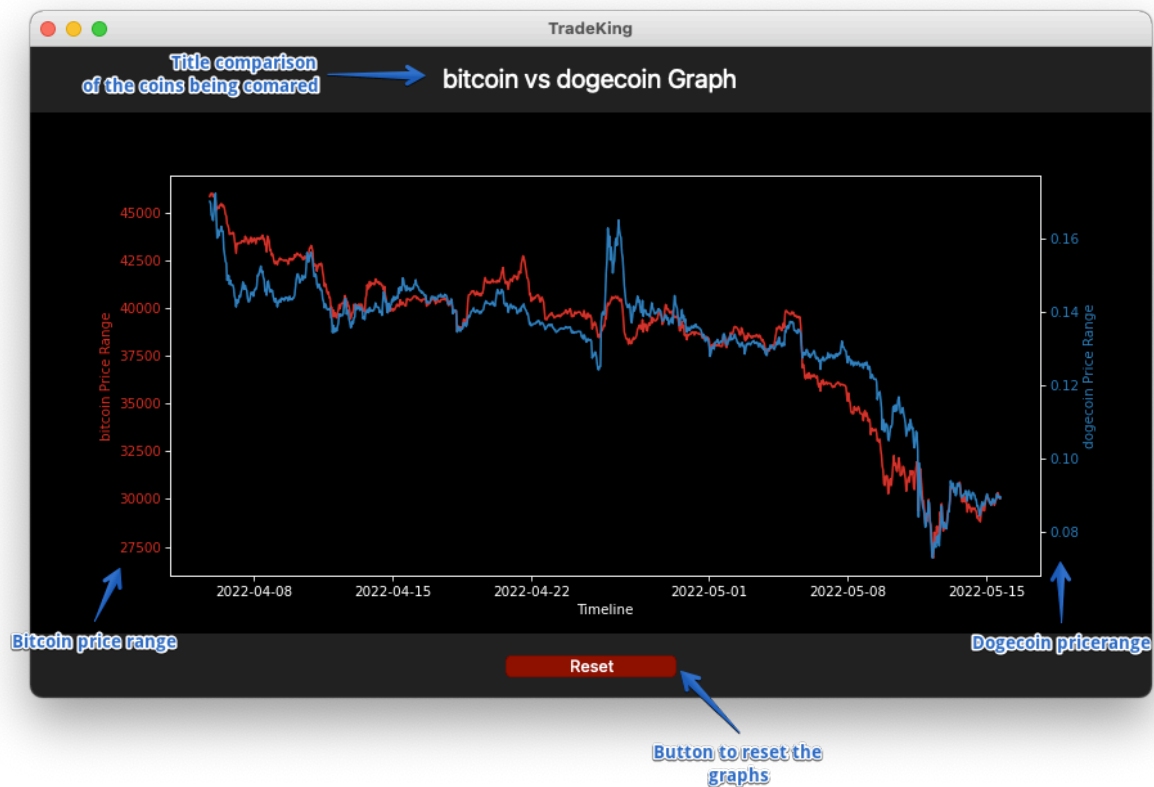


Once the user has selected the inputs that they desire they press the show graphs button and the matplotlib graphs is drawn and displayed to the user.



From here the user is able to add another graph to be compared with the displayed crypto or they may reset the graph completely if they want to have different parameters such as a different look back period.





The styling used is from the matplotlib library and has set styles.

## Testing


Primary testing of the application will fall under 2 stages.

1. Manual testing will be conducted throughout the development stage to ensure each function works as it should with unexpected inputs and actions. At the end of my project development, I sent my project to two colleagues to run the application to try and break it for any bugs they may find and provide feedback on overall quality, flow and any modifications that may need to be implemented.
2. I have used the unittest package from python. This allowed me to write unit tests to determine if functions run correctly and return the correct values. Unit tests helped me ensure functionality runs correctly and any changes or new features added to the project do not inflict on other areas of the website without human realisation.

## Unit tests

To create the Unit tests first there needs to be an installation of the python package “unittest”. All unit tests are created in the tests.py file. In this file there are 7 tests that are run .

1. Test to connect to the Binance exchange
2. Test to connect to the Bitmex exchange
3. Test the parent strategy.
4. Test the GUI title is returning the right name as true
5. Test the GUI title is not returning the wrong name true
6. Test the Geometry of the tk.Tk() window is utilised for graphs
7. Test the Geometry is not returning false values



```
class TestBinanceConnection(unittest.TestCase):
    def test_binance_connection(self):
        public_key_binance = "tyhityjdyjdyjdyjdyjdy456356y356y3565u7u657u4567u467u467uuu"
        secret_key_binance = "455345g56h56h356huk18kL689o5o567n6737g67j74j67j46873563657m7m567"
        testnet = True
        futures_client = True

        connection = BinanceClient(public_key_binance, secret_key_binance, testnet, futures_client)

        self.assertIsInstance(connection.public_key_binance, str)
        self.assertIsInstance(connection.secret_key_binance, str)
        self.assertIsInstance(connection.testnet, bool)
        self.assertIsInstance(connection.futures_client, bool)
```

The image shows a code editor with Python code for a unit test. Annotations with arrows point to specific parts of the code: "Assign to boolean" points to `testnet = True`; "BinanceClient class" points to the `BinanceClient` constructor call; "Random keys" points to the long alphanumeric strings for `public_key_binance` and `secret_key_binance`.

In the first test the `BinanceClient` class is tested by passing through the variables that it needs to connect to the Binance Exchange. This is to check that it is working as expected and is accepting the correct inputs from the user when connecting to an exchange in the correct format required.



```


class TestBitmexConnection(unittest.TestCase):
    def test_bitmex_connection(self):
        public_key_bitmex = "tyhjityjdyjdyjdyjdyjdy456356y35"
        secret_key_bitmex = "455345g56h56h356hukl8kl689o5o567n6737g67j74j67j46873563657m7m567"
        testnet = True

        connection = BitmexClient(public_key_bitmex, secret_key_bitmex, testnet)

        self.assertIsInstance(connection.public_key_bitmex, str)
        self.assertIsInstance(connection.secret_key_bitmex, str)
        self.assertIsInstance(connection.testnet, bool)

```

Next what is required to test is the connection to Bitmex, this is done with less input as for the binance exchange they offer both spot and future trades, Bitmex only offers futures which is why it only takes in 3 arguments.



```

class TestParentStrategy(unittest.TestCase):
    def test_strategy(self):
        exchange = 'binance'
        timeframe = '1h'
        client = BitmexClient
        contract = 'binance_futures'
        balance_pct = 2000.00
        take_profit = 12.00
        stop_loss = 10.00
        strat_name = BreakoutStrategy

        new_strategy = Strategy(client, contract, exchange,
                                timeframe, balance_pct, take_profit, stop_loss, strat_name)

        self.assertIsInstance(new_strategy.exchange, str)
        self.assertIsInstance(new_strategy.timeframe, str)
        self.assertIsInstance(new_strategy.contract, str)
        self.assertIsInstance(new_strategy.balance_pct, float)
        self.assertIsInstance(new_strategy.take_profit, float)
        self.assertIsInstance(new_strategy.stop_loss, float)

```

Using the asserIsInstance unittest library in the strategy class checks whether an object is an instance of the class or not.

The final 4 tests are more related to the Tkinter framework as it is a tricky framework to work with it is important that it is tested to ensure it is functioning effectively. First will test If the title is correctly loading for the window. Second will test if the window has a set geometry.

```
class TestGuiTitle(unittest.TestCase):  
  
    async def _start_app(self):  
        self.app.mainloop()  
  
    def setUp(self):  
        self.app = Graph()  
        self._start_app()  
  
    def tearDown(self):  
        self.app.destroy()  
  
    def test_title_true(self):  
        title = self.app.title()  
        expected = 'TradeKing'  
        self.assertEqual(title, expected)  
  
    def test_title_false(self):  
        title = self.app.title()  
        expected = 'NotTradeKing'  
        self.assertNotEqual(title, expected)
```

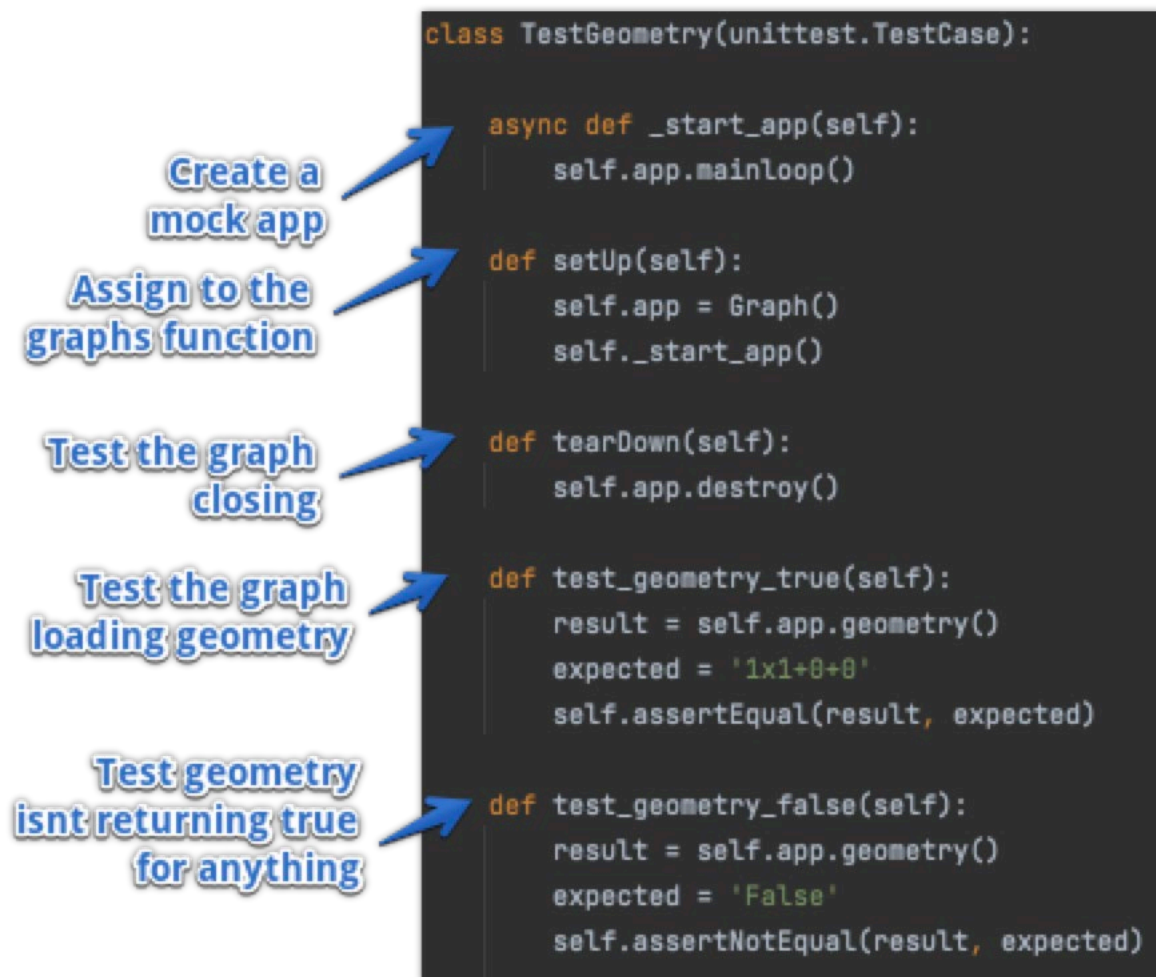
Create a mock app

Assign to the graphs function

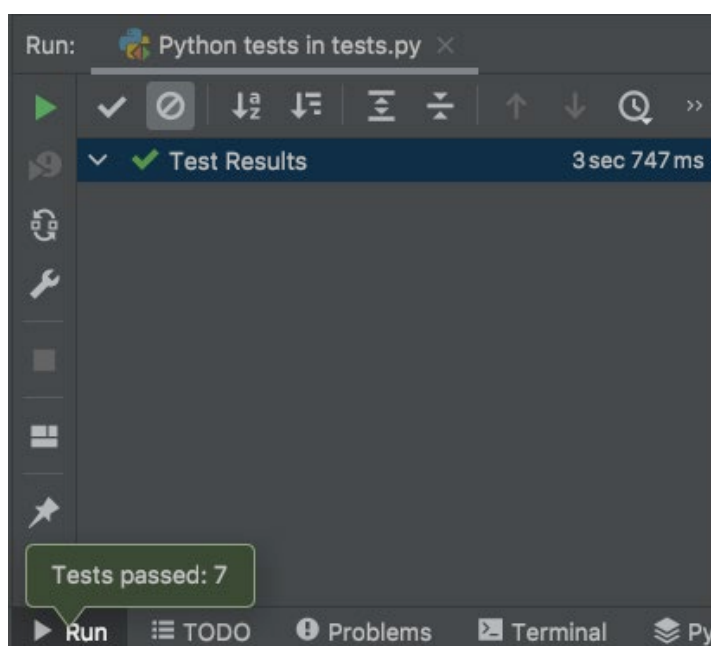
Test the closing of the application

Test is the title is correct

Test if the title isnt returning true for anything



To run these test right click on the test.py file and click “Run python test in tests” from the screenshot below we can see that all the tests have passed.



## Manual Tests

**Reviewer Name:** Dawid Marais

**Student Number:** x18133282

Bugs / Issue	Description
Graph input variables	When creating the crypto graphs on the graphs comparison page the graphs break when there is no user input for one of the parameters.
Graph reset button	When a graph has been created the reset button is hidden at the bottom and can only be seen when the frame is expanded

File Name	Line	Fix description
main.py	600-614	<p>The fix was simple for the Graph input variable bug and the graph reset button. There was no error handling for the input parameters, so that when the user pressed the “Show graph” button there was a data request sent to the coingecko API using the placeholder value of the tkinter OptionMenu, which would evidently break the program and display nothing. Checks were implemented to verify that when the user pressed the show button, that if the get function returns what is in the placeholders then a message will show to inform the user they have missed an input.</p> <pre>message = '' # Using to notify user if get_crypto_name() == "Crypto":      message = 'Crypto not set!'     print(message) elif get_currency() == "Currency":      message = 'Currency not set!'     print(message) elif get_lookback() == "Lookback":     message = 'Lookback not set!'     print(message) elif get_interval() == "Interval":     message = 'Interval not set!'     print(message)</pre>
main.py		<p>The next fix for the reset button not displaying was easy to fix as the graph’s window was separate to the log in and Trading window so its tkinter geometer was set in the open_graphs() definition All that needed to be done was to redefine the dimensions to account for the reset button.</p>

```
def open_graph():
    global graph_root

    graph_root = Graph()
    graph_root.config(background=background_color)
    graph_root.geometry('1000x550')
    graph_root.mainloop()
```


Below I have changed the window Geometry:

```
def open_graph():
    global graph_root

    graph_root = Graph()
    graph_root.config(background=background_color)
    graph_root.geometry('1000x600')
    graph_root.mainloop()
```

**Reviewer Name:** Shaurya Kumar

**Student Number:** x18138284

Bugs / Issue	Description
Time stamp incorrect	<p>When putting on a trade, the bottom left section that records the action is displaying an incorrect time. It is displaying a time that is one hour behind.</p> <p>When testing the issue:</p> <div> <div>Thu 12:37:22 :: Breakout strategy on 1000SHIBUSDT / 1m started</div> <div>  </div> <div>Current time 1:37pm</div> </div>

File Name	Line	Fix description
Interface/Logging_Component.py	25	<p>This issue was discovered after doing some searching into the code base. I didn't know whether the problem was originating from the timestamp for the trade, but in the trades component the trades timestamp was displaying correctly. I discovered the issue within the logging component where I logged in the time as utcnow() instead of having it as just now() to display the time as according to the time on the users computer.</p> <pre>self.text_log.insert("1.0", datetime.utcnow().strftime("%a %H:%M:%S :: ") + message + "\n")</pre> <p>Below I have applied the necessary changes:</p> <pre>self.text_log.insert("1.0", datetime.now().strftime("%a %H:%M:%S :: ") + message + "\n")</pre>

## Conclusions

This project, although frustrating at times, greatly improved my overall competency and use of good coding practices. I was able to design an application with a that I can use in my daily life and that can assist others in trading as well. Working with Tkinter was a new experience for me and has now broadened my problem-solving capabilities as well as my computing knowledge.

The biggest challenges that I had faced when creating TradeKing was the connection of the exchanges. There is excellent documentation available for connecting to these platforms, however this documentation left little to no room for alterations when implementing the steps. This resulted in hours of debugging for both exchanges. For each process that is carried out on TradeKing, there was a multitude of manual tests that had to be run as integration of one feature. For example, placing an order can be directly affected by another feature such as gathering a trade size or receiving historical candlestick data. There is a fine balance from which TradeKing can run correctly and it is all down to the requirements from each exchange.

TradeKing is developed using Tkinter GUI, which brought about the same amount of issues as the exchange connections. Tkinter is not a very developed framework and thus is missing a lot of functionality that is already built in e.g. web applications, simple functionality such as a drop down window or a scroll bar took a lot of time to hard code into the application and didn't add to the overall main functionality of the application as it was just an "extra feature". I chose Tkinter as I had never used this framework before and the challenge motivated me. I managed to work with this framework after hours of online learning tutorials (Salvatierra, 2018). Working with the graphs also brought up issues as Tkinter does not allow for visualisation tools, such as Plotly, to be imbedded in their window. Tkinter allows matplotlib which doesn't offer as many features as Plotly and the best resource I could find was dated from 2014 (*How to get GUI's with Tkinter, 2014*).

## Further Development or Research

TradeKing has a lot of potential for future development, Currently it is developed for a macOS specific user as a lot of the tkinter design variables are used specifically for a macOS system. However the underlying functionality works the same for these specific buttons it is only the styling that is used in the TradeKing window that is macOS specific so users working on windows or Linux computers wouldn't be able to see the text or colours of these buttons and in result wouldn't know what the functionality behind them is.

Further development could be done in the user trades frame of the main trading window. Currently what is displayed to the user is the details of their current trades:

- Time stamp
- Exchange the trade is being made on
- Traded currency symbol
- Strategy type
- Side
- Size
- Profit and Losses
- Status

More can be added to this such as the users current balance, their totals profit or losses for a specific currency and even the ability to record these trades into an excel sheet. During the user sign in process there could be more attractive features implemented and also have a user profile that holds the users API keys in a database so that when a user returns to the application they won't need to use their API keys as a form of logging into the application, but instead have a username and password. The organisation of the Trading window could be redeveloped with more time invested into learning tkinter framework or



using a completely different framework altogether as Tkinter was a very tough GUI to develop with and caused an extreme amount of errors.

## References

### Udemy courses that helped build the Application:

Rivera, F., 2018. [online] Available at: <<https://www.udemy.com/course/build-a-crypto-bot-100-functional-algorithmic-trading-freqtrade/>> [Accessed 15 May 2022].

Hagmann, A., 2020. [online] Available at: <<https://www.udemy.com/course/cryptocurrency-algorithmic-trading-with-python-and-binance/>> [Accessed 15 May 2022].

H Carmier, V., 2019. [online] Available at: <<https://www.udemy.com/course/cryptocurrency-trading-bot-with-user-interface-in-python/>> [Accessed 15 May 2022].

Salvatierra, J., 2018. [online] Available at: <<https://www.udemy.com/course/desktop-gui-python-tkinter/>> [Accessed 15 May 2022].

### Websites

Cohen, G., 2021. Optimizing algorithmic strategies for trading bitcoin. *Computational Economics*, 57(2), pp.639-654.

Corbet, S., Eraslan, V., Lucey, B. and Sensoy, A., 2019. The effectiveness of technical trading rules in cryptocurrency markets. *Finance Research Letters*, 31, pp.32-37.

Guides, T.S., 2018. Tether Trading Strategy–Bottom Rotation Trading.

Kanani, P., Karasariya, J., Zadafiya, N. and Nayak, A., 2021, December. A Ratiocinative Concept of Algorithmic Trading using MACD Indicator. In *2021 5th International Conference on Electronics, Communication and Aerospace Technology (ICECA)* (pp. 369-376). IEEE.

Lemenkova, P., 2020. Python Libraries Matplotlib, Seaborn and Pandas for Visualization Geo-spatial Datasets Generated by QGIS. *Analele stiintifice ale Universitatii "Alexandru Ioan Cuza" din Iasi-seria Geografie*, 64(1), pp.13-32.

Saabith, A.S., Fareez, M.M.M. and Vinothraj, T., 2019. Python current trend applications-an overview. *International Journal of Advance Engineering and Research Development*, 6(10).

Vezeris, D., Kyrgos, T. and Schinas, C., 2018. Take profit and stop loss trading strategies comparison in combination with an MACD trading system. *Journal of Risk and Financial Management*, 11(3), p.56.

Yi, E. and Lee, W.B., 2022. A Study on Stock Trading Method based on Volatility Breakout Strategy using a Deep Neural Network. *The Journal of the Korea Contents Association*, 22(3), pp.81-93.

(2014) *How to get GUI's with Tkinter (Intermediate)*. Available at:

<https://www.youtube.com/playlist?list=PLQVvvaa0QuDclKx-QpC9wntnURXVJqLyk>

(Accessed March 8, 2022)

(2021) *Accessor and Mutator Methods in Python*. Available at:

<https://www.geeksforgeeks.org/accessor-and-mutator-methods-in-python/> (Accessed February 11,

2022)

*Bitmex*. Available at: <https://www.bitmex.com> (Accessed: 10 November, 2021)

*Binance*. Available at: <https://binance-docs.github.io/apidocs/futures/en/#change-log>

(Accessed: 28 November, 2021)

*Firsttrade Investment Glossary*. Available at: [https://www.firsttrade.com/content/en-](https://www.firsttrade.com/content/en-us/education/glossary?&letter=s)

[us/education/glossary?&letter=s](https://www.firsttrade.com/content/en-us/education/glossary?&letter=s) (Accessed December 2, 2021)

*How to Get an API Key On Bitmex Exchange*. Available at: [https://aivia.io/blog/en/how-to-get-an-api-](https://aivia.io/blog/en/how-to-get-an-api-key-on-bitmex-exchange/)

[key-on-bitmex-exchange/](https://aivia.io/blog/en/how-to-get-an-api-key-on-bitmex-exchange/) (Accessed February 12, 2022)

*Matplotlib Style Sheets Reference*. Available at:

[https://matplotlib.org/3.5.0/gallery/style\\_sheets/style\\_sheets\\_reference.html](https://matplotlib.org/3.5.0/gallery/style_sheets/style_sheets_reference.html) (Accessed March 1,

2022)

<https://regex101.com/>

## Appendices

### Project Proposal

#### Objectives

TradeKing will be an automated trading bot with the goal of helping users to execute trades on cryptocurrency trading platforms following a trading strategy that has been specified to the trading bot. This would help users who are avid or casual traders, as it would eliminate the risk of losing out on trades due to the inaccessibility of the trading platforms at certain points in the day. This can be due to many reasons, such as being in areas of low internet coverage, where the trades would be required to be made. In turn, this would also leave users more time in the day to spend working or creating new projects as they wouldn't have to constantly be following the price trends of the very volatile crypto currency market throughout the day to wait for certain buy and sell points to be hit for their trades to be executed, as it will all be done automatically throughout the TradeKing bot.

#### Background

The reason I have decided to undertake this project is due to the reason that it addresses a problem that I personally face, being someone who has invested a lot of time into crypto currency trading. By creating a program that will be able to send market orders to the Binance or BitMex platforms, I will free up a lot of time for myself to invest into more market research for which coins to invest into in the future. I also wouldn't have to risk missing out of trades that may occur at obscure times of the day.

In order to meet the objectives listed in section 1.0, I will need to have an application that, with the use of API's provided by the selected trading platforms, is able to successfully pull in data about the current values of the crypto currencies and send that data through the API's to perform buy and sell actions on those crypto currencies at specific buy and sell points. These points will be predetermined in the code so that the bot follows a trading strategy. The Application should also have an attractive graphical user interface to make interaction with the data easier.

### State of the Art

There are Trading bots out there built for their own specific reasons and have their own benefits; one of these being Coinbase. When it comes to an application that provides a wide range of trading strategies for its users, Coinbase allows users to customise their trading with more than 150 trading templates that can be automatically executed when the market conditions meet the predefined parameters with accumulation and long-term holding strategies and stop-loss settings. This application, however, is a paid application with a free option that comes with a select few of templates that can be used for a short period of time before payments to the application will be made. This application makes use of the paid package, as it is not clear which strategies the free version contains compared to that of the paid version, as the application wouldn't allow users on the free plan to have access to the best strategies and would provide them with limited options.

Another option for trading that offers 12 unique trading bots for no additional fee is Pionex and is to be considered the best for high-volume traders. Pionex supports manual trading using crypto – crypto conversions but its primary product will be that of its trading bot selection. Although there are many positives about using the Pionex application, to automate trading there are also many very obvious downfalls. Pionex uses a maker-taker fee schedule, which means that you'll pay a fee when you place trades that "make" liquidity on the market and that "take" liquidity away from the market, causing there to be higher fees on every trade as

compared to other paid applications, which in the long run will accumulate to be a steep fee.

Another option is less automated trading using the crypto currency exchange programs that the API's are connected to, these being Binance and Bitmex. These trading platforms are well-designed and offer a wide range of tools to help crypto currency traders. They have tools such as setting moving average markers to indicate the general price trend of specific currencies to see how the future forecast of these currencies may look. They also offer other services such as Binance earn, which allows users to lock the currencies that they have for a certain period of time and receive a certain amount of interest depending on the time period that the currencies are being locked for so that the Binance platform may use the locked currencies as collateral for other users. In this period of time, the users are not able to sell their locked or "staked" coins, but if desired can opt to redeem their coins and after a period of one day. After redeeming they would be able to trade with their coins again, but would lose any coins accumulated through interest over the period of the day the coins were locked. Binance and BitMex offer the service to set buy and sell points to sell the crypto currencies when they reach a specified value.

### Technical Approach

The approach I will take will be firstly in the lines of code where I spend the time attempting to have my application successfully communicating with the crypto trading platforms through the use of APIs, so that if I can efficiently collect the live data from these applications, I could then start working on sending data to the platforms in order to carry out specified tasks such as buy and sell operations. Once I am able to perform these, I can then work on the way the data is being presented to the users in forms of graphs with an appealing graphical user interface that displays information in an organised and easy to understand format.

In order to identify the requirements, I will need to do extensive research into the fundamentals of using APIs from the Binance and BitMex platforms, as well as using the libraries that are developed to help handle and organise the data that is

being seeded into my application. I have already identified one of the very useful libraries called Panadas, which is a python-orientated library that is used to handle data received from APIs. I will need to plan ahead what I may encounter in terms of requirements for the end user that will be interacting with my application and build a solid foundation around how those needs will be addressed. Being an avid trader myself, I have the knowledge on what my application should be able to deliver to the end user and how the navigation through the application should ultimately be, with the users login being the first interaction the user has with the application in order for the application to connect the user with their live data from the trusted trading platforms.

I will set myself milestones throughout the production of this application to ensure that I am not leaving the application's development to be done last minute as I must plan for there to be problems that would need to be solved, as there always is when coding an application. These milestones would consist of me having an application by Christmas time that will be able to seed in the information of different crypto currencies as a live data stream, as with such markets the prices are very volatile and are consistently changing every second. This will be my first task that will be completed in the next month. The application should also allow for the user to send requests to the trading platforms with the use of the APIs to perform different actions, such as buy and sell coins, and this will be my goal for the month of December. If I have completed these tasks within the specified time frame, then I will move onto creating log in pages for users to log into either their Binance or BitMex account, depending which platform that they use.

## Technical Details

The Main language that will be implemented into this project will be python. Python is an extremely capable language and can incorporate many languages into one with the use of importing libraries to help organise and display data in a GUI. Anaconda will need to be installed in order for me to operate Python on my machine and to have the means to import the libraries required by my program.

One of the main libraries that I will use will be “Pandas” library in order to calculate the technical indicators when trading to perform buy and sell orders.

My application will not only need to perform trades, but should also record the trades for later referencing so that the users that are using the application can follow trades that have been taking place by the trading bot if the trading bot has been left to perform strategies over a long period of time. This will be logged into the trades component of the application that will be displayed in column form.

### Special Resources Required

To create this application, I have created a profile on both the BitMex and Binance testnet platforms for me to obtain the available APIs that will be used in my application to receive the live data feeds of the cryptocurrency’s values. At the end of the project, I will have a login available for the end user to input their login details for their respected platform, that being Binance or BitMex, for my application to get the API keys for that specific user to successfully carry out buy and sell orders for those users when the trading bot is following a specific strategy. Users would thereby be required to have a Binance or BitMex account created prior to using my application, as there are various security checks that are needed to be passed when creating a Binance profile. These include validation checks by ID submission and the use of the camera to confirm identity. These processes would not be available through my application and users would not be able to sign up for these trading platforms via my application.

### Project Plan

The plan to create my project has been broken down into phases which will be implemented over the course of creating the application over the next few months. The first phase of the development of my application, TradeKing, will be the research and planning of the development section in which I will take time to investigate different aspects of my software project in order to have the most effective methods, algorithms and trading strategies implemented into my application.

I then had to decide which crypto trading platforms my application should link to and communicate with when placing buy and sell orders. That is when I had decided to use Binance. They are one of the world's leading crypto trading platforms with a significant increase of users from 2017 (1.5 million) – 2020 (21.5m), as many people have started to seriously invest into the crypto market over the course of coronavirus pandemic. The second application, BitMex, has more than 1.3 million accounts, but the content available for working with this crypto trading platform across a multitude of learning sites is vast, which in the long term will be helpful as I will have material to review if a problem arises during the creation of my application.

This decision-making process took a week for me to undertake with all the different research variables. The next few months up until December, I will be working on using the Pandas Library and the API keys from both Binance and BitMex trading platforms in order to have the currencies' values and market data streaming into my application, with the use of WebSocket API's, and to have messages logging into my application to communicate to the users, and myself during development, about the status of the various processes that my application is trying to do.

Once this process is successfully implemented into my application, I will then work on the API connectors within the Binance Futures model, making use of the REST API's public endpoints (Coding and Testing) and the private endpoints where my application will be able to perform operations on a user's Binance account using authentication requests. The next process for me to have would be to try and organise my data to have it more efficient and readable using various Data Models and Variable typing. From there on I can work on Error handling methods. The time period I set myself for these tasks will be 2 weeks, where I should have the majority of this section finished by the middle of November.

Once I have figured out the streaming and the organisation of my data through the use of the Binance API and WebSocket API, I can then begin work on the user interface that my application will have, using the TKinter package for creating user interfaces with python programs. For this section, I will aim to have the user interface operational two weeks after my proposed finish date for connecting the



API and WebSocket's and would fall to the end of November. I will leave a week of leeway time in case there is any unforeseen events that could cause a disruption in following my proposed schedule and thus the final deadline will be just after the first week of December.

From there on, I will start working on creating the strategies that will be used when trading by using classes to organize my strategy module and start coding the first strategy, which is going to be the break out strategy that is based upon the candlestick patterns.

## Reflective Journals

### Month: 1

#### What?

This Month I needed to successfully pull in the live data streams of the response content and the currencies compared to the USD using the API keys for the trading platform into my application.

I have been coding this application using the python language, which has packages that make seeding in the live feed possible. Before I could start importing anything into my project, I first had to obtain the API keys that would be used to draw in the live data from my chosen Crypto trading platforms. I decided to go with BitMex and Binance. Both of these applications have a separate "Testnet" platform that allows for users to get API keys that won't be linked to specific profiles in order to allow for any users on their Testnet platforms to practice with investments and implement trading strategies to get better understandings on how their investments would play out without having to risk their money from the start. Once I had obtained the Testnet API for both trading platforms, I then moved onto creating my application and I started by creating the main.py file where I imported tkinter and logging packages. These packages are used for drawing in the live stream data and logging the data into the console in pycharm. The APIs are used in the binance\_futures.py and Bitmex\_futures.py files that I had created and the logging and requests packages are used in order to seed the data.

## **So What?**

Consider what that meant for your project progress. What were your successes? What challenges still remain?

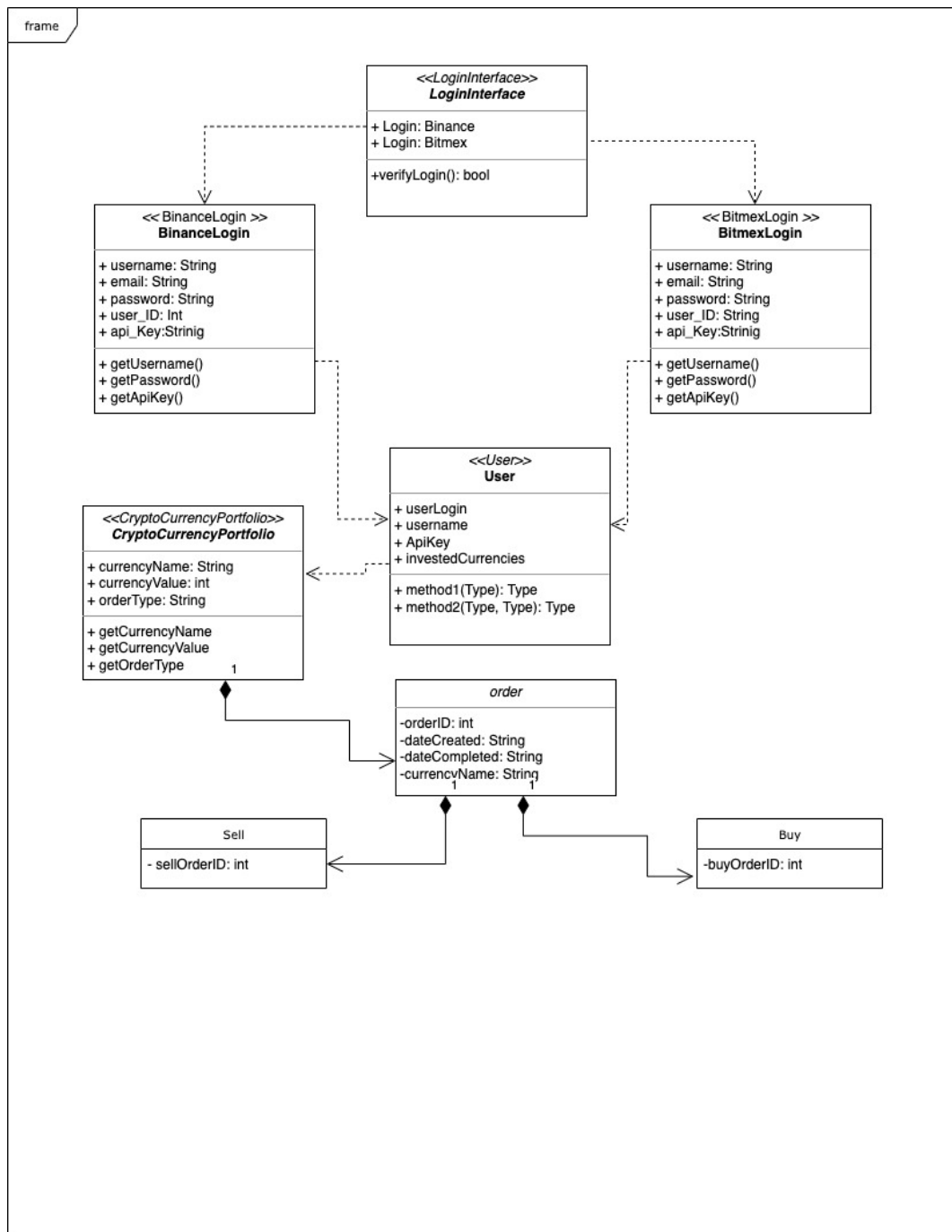
This has been a big step for my project, as I am able to successfully communicate with the Binance and Bitmex testnet platform using their API and successfully receive information using code. Next I will need to pull in the live prices using these API's and display them into a graphical interface to make reading the information much easier before I start coding the bot, which will be used to send order and sell requests to the trading platforms

## **Now What?**

In order to address what is still to be done, I will need to do some more extensive research on how to work with the entry point that I have created and become more familiar with working with the public API, as well as how to create an attractive interface in order to display the API data.

## **Month: 2**

This month I have successfully gotten the API to communicate currency values between my application and Binance and Bitmex applications. This involves the real time data of the currencies' values, which would mean that I now have a solid foundation to build my application on and to develop the various strategies. I have also created an initial basic UML diagram for my application to show the general flow of my application, subjected to change as the program is created.



So What?

The next step for me will be to display these live prices that my application is pulling into an easy-to-read graphical user interface and from there I can start to develop the strategies.

## **Month: 3**

### **What?**

I have been doing research into the best ways to explain trading terminology to users that don't have prior trading knowledge. These will be available to the user as a hover-able feature located next to each required input displayed as a question mark icon. If a user is not familiar with what they are inputting, although it would be highly recommended that a user has knowledge about the terminology before putting in a trade, TradeKing will be linked to a demo account platform and the money used will not be real money.

**\*\* UPDATE \*\***

TradeKing has a link in the strategies section of the application that directs users to a glossary web page that holds definitions for ALL trading terminology, which serves a better purpose as it eliminates the risk of a user being confused on a term that may not be referenced in the application's definitions.

### **So What?**

This is a good step and feature for my application to have, as it will help diversify the users that are able to use my application. It will not only give the user tools to make a trade, but the knowledge and understanding for those new traders on what exactly is needed and why it would be needed instead of trading blindly, which is not recommended by any standard

### **Now What?**

Now I am going to work on the UI of my application, as I made a basic one for the mid-point demo of the project, and then will be implementing the definition features into the project while at the same time, looking into how my application will be able

to webscrape the test net platforms for the API keys once the account has been made on.

## **Month: 4**

### **What?**

During this month I discovered that the testnet applications will not allow for my application to web scrape as there is no way to get around the 2 factor authentication that is present on both platforms. This requires me to rethink the way the user will enter my application.

I have developed the GUI of the Trades application that will need to be developed as the application progresses, there is 4 frames hosted into the Tkinter window and they are going to be used to hold the 4 required sections of my application

- An area that the user can type a crypto currency and be shown the live bid and ask price of that currency
- An area where the strategy setup is hosted where a user is able to set up what is required in order for a trading strategy to be created after clicking on a button
- An area where the user can see the activity of what they are doing in the application such as when a strategy is started.
- An area where a user can see the strategies that they have made hosted be recorded with the status of whether the trade is active or not.

### **So What?**

Now that I know about the limitations of the log in and the inability to obtain these API keys automatically through the application there needs to be a rethink of the design architecture of the application. For the GUI this is a good step in the development of my application as it opens doors for what I am able to do next in regards to starting to fill out these areas of the window with the information that I need to display to the user.

### **Now What?**

I will need to look into alternative methods of logging into the application. This could require some research into creating a user profile that holds user keys or a log in using the keys.

Now that the skeleton of the application has been created what next to do is to create the two strategies that are going to be used: 1. The break out strategy 2. The technical strategy.

## **Month: 5**

### **What?**

During this month I have discovered an alternative way of logging into my application that does not require a user to set up a profile. Users would be required to set up a profile on one of the exchanges so to avoid the user having to go through a very lengthy process of creating multiple profiles that will just put them off using TradeKing . There will be a GUI pop up that asks the users for an input of 2 variables

1. The API public Key
2. The API secret Key

If the user does not have any open account there will be a video available for them to watch that goes through the processes of how the user may signup – find – and obtain the required keys. . I have also got the application to successfully subscribe to crypto currencies where a user can type in what crypto currency they want to see the live bid and ask prices of and the application will be able to show them the live prices via a websocket connections.

Furthermore I have coded the breakout and technical strategies that will be used by the user when trading. I used classes to organise the strategy module which has the Strategy class as the Parent class with two child classes, Technical and Breakout, that will inherit from the Parent strategy. Then I moved to working with candle stick data as it is essential for my application to receive this data from the exchanges to receive signals on whether or not to enter a trade. The Technical strategy was the hardest to write as it relies on technical indicators in order to get signals to enter a trade or not.

## **So What?**

Now that I have an alternative way to log in that means that the user is able to gain access to their exchange data without having to hard code in their API keys into the source code. For the case of coding the strategies it is now time for me to enter trades using these strategies as I have a successful link between the TradeKing application and the exchanges shown by the live crypto prices.

## **Now What?**

Now I need to try and create a simple log in GUI using Tkinter that asks the user to input their API keys and to just have the system print into the console what the API keys are so that I can validate that the system can assign those keys to a variable. I was also noted about adding functionality to my application and thus I will look into creating some visualisation of the crypto prices in the form of graphs that the user will be able to customize to their own preference.

## **Month: 6**

### **What?**

During this month I have created a GUI that has two input fields for the users API keys, Once the user has put in their keys the system prints the keys in the console. This is to show that the keys are being obtained by the application and can then be used to initialise a connection with the binance and bitmex exchanges. I was then left with a problem of trying to navigate through my application as with Tkinter all components of the application are loaded when the mainloop is initiated and buttons are used to bring certain frames to the surface and hide the others as a form of navigation. This left me with a problem of trying to create a single tkinter frame that loads everything that I need on the window.

Moreover I had looked into using graphs in my application where I needed to have a API connection separate to that of my main trading application as it was not being run

at the same time as when I was testing. Plotly was a consideration for the visualisation but unfortunately there is no way to imbed this tool in a tkinter window, however Tkinter does allow for Matplotlib to be imbedded. I was also able to get the application to make trades using both Breakout strategies and Technical strategies.

### **So What?**

There was a lot achieved this month as I have found a graphing framework that will be used in my application and using an external API separate to the connection with either the binance or bitmex exchanges. I have also solved the issue of logging in using the API keys what next to do is to create efficient navigation between the pages