Investu

A real-time stock market investment simulator for students

J-- H-----

Table of Contents

Analysis

Background and Problem Identification

Summary of the Identified Problem

Mr. Butterworth, is the head of Economics at King Edwards School, who has had a lot of experience over the years in preparing students for the Student Investor Challenge. As an Economics and Computing student Mr. Butterworth approached me in search of a solution regarding the Student Investor Challenge that the school participates in. Mr. Butterworth feels that those who are not currently competing in the Student Investor Challenge generally didn't do so as they felt they didn't have enough experience to do so, and even those who did participate feel they could be in a much stronger position had they had some prior experience before starting the competition.

Background and Explanation

The A-Level Economics course does not go into a lot of detail regarding stocks and investment, despite the fact that trading stocks and shares is a significant part of Economics in the real world and relies heavily on many of the underlying principles of Economic theory. The first exposure most students get to investment is in the Student Investor challenge, an investment challenge participated in by around 40,000 students from across the UK, who compete in teams of 4, investing £100,000 of virtual money on stocks such as those in the FTSE100[i]. The teams aim to try and make the most money possible in the time allowed, with the top 500 teams going through after 2 months to compete in further rounds. Of those 500 teams, the winning team receives a paid trip to New York, and money for their school.

With such a large prize available and the prestige that comes with winning the Student Investor Challenge, one would imagine there would exist a program to practice buying and selling stocks before the Student Investor Challenge starts. As an Economics student and a Computing student, I was interested in whether this software existed. However, after much research there appears to be only a few reliable programs, and those that do exist only support stand-alone accounts, with no built-in way to collaborate with a team. Furthermore, during practice, it would be useful for teachers to be able to track student teams who are planning to enter the challenge, and observe them as they prepare for the start date. Through research I've found that this feature also doesn't appear to exist.

It therefore appears that there is a need for a multi-user, integrated and intuitive simulator, to:

1) Allow students participating in the Student Investor Challenge to practice investing in the stock market with the ability for teachers to guide and aid users in their trading decisions.
2) Allow even users not participating in the Student Investor Challenge to become familiar with the concept of trading on the stock market, and to practice with the guidance and advice of their teacher.

How the Stock Market Functions

A share is a piece of a company that someone can buy. When a company needs to raise extra funds, it issues shares through an initial public offering (IPO), in which shares are issued at a price determined by the estimated value of the company and the quantity of stocks being issued. These are then sold to traders and investors. After these initial shares are sold, the company keeps the funds generated, and the investors keep the shares they have bought in the company. The investors can then trade these shares, with other traders, however the company now has no part in this deal and receives no money from any of these trades.

Investors and traders by shares because they have a value that changes. If an investor buys a share, and the value of that share increases, they can then sell that share for profit. The price of shares is determined by the market, which is a huge collection of investors and traders who buy and sell stocks. It should be noted that when an investor decides to buy shares for a certain price, there must be someone else in the market willing to sell their shares for that same price. If there exists a higher demand for shares in a company than there exists supply, then the price of the company's shares will be driven down, and vice versa. The fluctuations in supply and demand for shares are simply referred to as 'market forces'. The reasons behind why these market forces push or pull in a certain direction are, in their most basic form, the feelings and attitudes of investors towards a certain company. These feelings and attitudes are largely dictated by the earnings of a company. If a company's revenue begins to decrease, an investor would perceive shares in that company to be of less value, and so be more reluctant to purchase them, leading to a fall in demand and an increase in supply, and hence a drop in the price of the shares.

In reality, however, it can be very difficult pinpoint the future prices of stocks and shares, and there doesn't exist a perfect model that can explain with complete certainty the reasons behind fluctuations, or predict them in the future. [ii]Shares are volatile, meaning their price can change rapidly, with seemingly very little reason. It is this quality that makes trading shares a potentially lucrative pursuit.

The stock market can be divided into sections that are valued using indexes. An index is the measurement of value of a selection of stocks, calculated from the prices of the stocks that comprise it. There are many such indexes, the most well-known of which being the FTSE 100, S&P Global 100 and the MSCI World. The FTSE 100 is a list of 100 companies on the London Stock Exchange that hold the highest market capitalization. The FTSE 100 is used as a gauge of growth in the UK, and is one of the indexes that can be traded on in the Student Investor Challenge, and is arguably the most relevant index to track for investors in the UK.

Terminology

The following are definitions of subject-specific terms and phrases that appear in the write-up of this project [iii]

Security	A tradable financial asset – refers to any form of financial instrument including stocks and shares
Stock	The general term used to describe ownership certificates of a part of any company
Share	Usually used to refer to shares in a specific company
Index	A hypothetical portfolio of securities tracking a particular market
FTSE 100	An index tracking the top 100 UK based companies with the highest market value
Market Value	The price which a security would fetch in a market.
Shareholders	An owner of shares in a particular company – an individual who has an interest in the success of the company
Execution	The completion of a buy or sell for a security
Quote	The most recent price at which a security was sold Ask quotes are the most recent prices and quantities at which shares can be bought or sold
Volume	The quantity of securities in a given market traded in a given period of time
Initial Public Offering	The initial value at which a company sells its shares, usually by companies looking to quickly raise revenue
Volatility	The statistical measure of the dispersion of returns in a given security, calculated using the standard deviation. Generally, the higher the volatility, the riskier the investment

Investor	A entity who puts money into securities or commodities with the expectation of receiving profit
Market Capitalization	The value of a company traded on the stock market, calculated by multiplying the current share value by the total number of shares
Intra-day Price	The movements of a share price during the day
Outstanding Shares	Refer to the number of shares owned by investors
Day trading	Short term trading of securities in which the buying and selling takes place within the same day
Hedging	The practice of taking a position in one market to offset the risk adopted by in an opposing market
Portfolio	A range of investments made by a person
Spread	The difference between the bid price and the ask price of a security

Interview with client

The following is an interview with the client, Mr. Butterwort, the head of Economics at King Edwards School. The interview has been edited for brevity.

"What would you say is the issue currently with the Student Investor Challenge?"

"I feel as though a lot of students go into the challenge with very little knowledge of trading, because it's not a topic covered in our syllabus. Of course, the economic principles that underpin the stock market are covered, but you can't substitute real life experience with theories from a textbook. On one hand, that's the whole point of the challenge – to expose the students to the world of trading, but on the other, with it becoming so competitive, those without some background knowledge and experience are struggling to place well. The students need a way to practice before the start date of the challenge, so that when it starts, they have their strategy and knowledge already in place to do the best they can."

"Why do you think this is an important problem to solve?"

"The challenge is becoming increasingly more competitive, and over the years has become well known because teams who do well have a good chance of winning pretty significant prizes, and recognition for their achievements from Universities and employers and so forth. When a student includes it on their personal statement, that they've placed highly, it creates a good impression. It's therefore quite important for us to step up our game when preparing our students, so that they have a head start over teams from other schools, and can do as well as possible. Although having said that, I'm not sure how long we'd keep that advantages as a program that solved this problem would be widely used by other schools who do the challenge."

"If we were to create a simulator to model the trading experience, how would that help solve this problem?"

"A simulation similar from the one they use in the real challenge would allow the students to practice before the real simulator is available to the teams, which is obviously a great head start as almost all students entering the challenge

"Could you specify what features you would the simulation to have?"

"Well the program would need to have the ability for students to login into their account and buy and sell stocks, like the Student Investor Challenge. They should have a balance, maybe that I can see and edit. They would have to be able to see all of the stocks they currently have in their portfolio, and sell them if they decide it's a good time to sell. It would be useful to be able to see the historic prices of a stock, so the students can see where the price has moved in the past 24 hours, for example. To simulate the real thing, teams should definitely be able to work together, and perhaps share their balance together so that all of their trades contribute to one team result. It would help if I could have an overview on a teachers account, of all the teams in the school. Because a lot of students don't have much experience with trading, if the program could suggest stocks to invest in, and suggest stocks to stay away from, it might help the students to start to get a feel of which stocks to trade and why."

Interview with users

Ben is a student participating this year in the Student Investor Challenge. His team was 4th out of 10,000 teams last year, narrowly missing out on the prize money. The following is an interview with him which aimed to discover other areas that could be improved upon, and further features, other than those already suggested by Mr. Butterworth, that Ben and his team feel would have been useful to have access to before the challenge started. The interview has been summarized for brevity.

"Ben, the basic premise of the program would be a stock trading program that your team could log into at any time, ever before the competition has started, to practice trading. At its core, what do you think this program should be able to do?"

"I think what's most important is that the program has instant access to the prices of stocks in the FTSE 100, as that's the index we trade on most. Also, it would be good if the price data is displayed in real time and is accurate. Without that, it's very difficult to trade as information displayed on websites and on the program, wouldn't match up, which would make our research confusing and would mean we'd probably make decisions based on misleading information"

"Visually, how do you imagine this program looking?"

"I find the actual program for the challenge is quite difficult to use. Its cluttered and not clear, with information spread over many tabs. To make progress faster, we'd like a program that clear and concise, with information displayed in an easy to understand way. Perhaps it would have stock information on one side and a list of stocks currently in our portfolio on the other. There would also need to be space for other information like notes and graphs.

"What other features do you think would add to quality-of-life when using the program?"

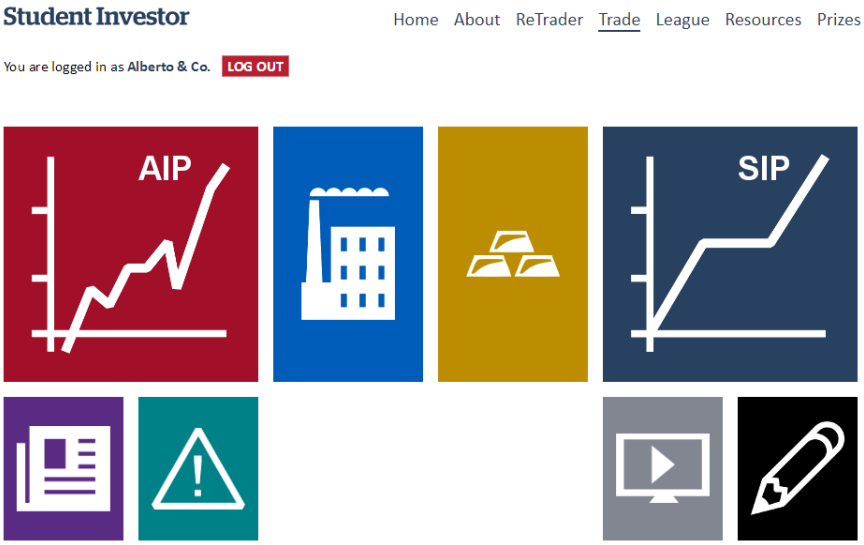
"If data from the last hour, day and week could be visualized in graphs, it would make the process of deciding which stocks to invest in a lot clearer and more accurate. Also, the program for the actual challenge wastes a lot of our time as it doesn't display the commission charge for each trade. If that could be displayed before each trade it would save us having to work it out, which would allow more time for more important things like researching stocks. With all of us logging in and trading at different times of the day it can sometimes be difficult to know who's bought what and why. If we could see who made which trade and their reason why in a notes section it would clear up a lot of confusion. None of us are that great at computers so I think if it could be as simple to use as possible it would be good."

"It would be good if this program was useful not only for the Student Investor Challenge but also for general use by students, after the challenge is over. Can you think of any features that could be added that would be beneficial in that respect?"

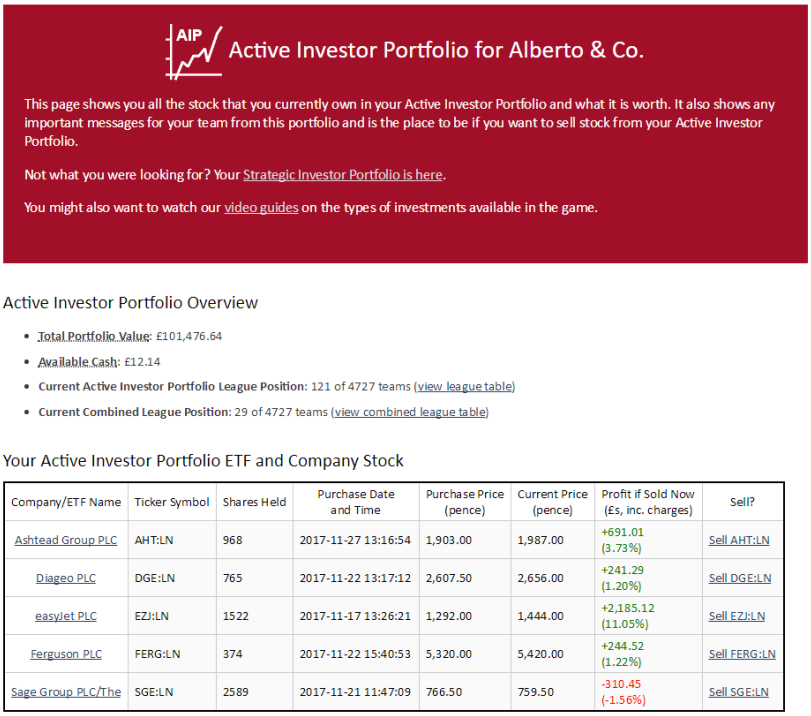
"Yes – there are some features that the challenge doesn't have but would be very useful if we used the simulator outside of the challenge, like later in the year. – the Stop Loss / Take profit feature is one of them. That basically would allow us to set a price that, if reached by a stock, would automatically trigger it to be sold, to allow users to cash out or stop losing money, even if they aren't online at the time. Being able to see the volatility of a stock is another idea. Depending on your strategy, stock volatility is a great indicator of whether to invest or stay away from a trade."

Description of the current system

The system currently in place is the Student Investor program. The interface the user is faced with when they log into the program. It features 8 buttons that allow the user to navigate. The function of each button is not overly clear for new users.



Selecting the first box opens the Investor Portfolio for the team, which displays the interface to the right. This has the list of currently open positions and the details for those positions



FTSE 100 Companies

Company Name (sort)	Ticker Symbol (sort)	Industry Sector (sort)	Industry Group (sort)	Price (pence) (sort)	Change (percent) (sort)
3i Group PLC	III:LN	Financial	Private Equity	867.50	-7.50 (-0.86%)
Admiral Group PLC	ADM:LN	Financial	Insurance	1870.00	-23.00 (-1.22%)
Anglo American PLC	AAL:LN	Basic Materials	Mining	1327.00	-25.50 (-1.89%)
Antofagasta PLC	ANTO:LN	Basic Materials	Mining	887.00	-0.50 (-0.06%)
Ashtead Group PLC	AHT:LN	Consumer, Non-cyclical	Commercial Services	1987.00	+16.00 (+0.81%)
Associated British Foods PLC	ABF:LN	Consumer, Non-cyclical	Food	2889.00	-31.00 (-1.06%)
AstraZeneca PLC	AZN:LN	Consumer, Non-cyclical	Pharmaceuticals	4711.50	-12.00 (-0.25%)
Aviva PLC	AV/:LN	Financial	Insurance	503.50	+2.50 (+0.50%)
Babcock International Group PLC	BAB:LN	Consumer, Non-cyclical	Commercial Services	663.50	-14.50 (-2.14%)
BAE Systems PLC	BA/:LN	Industrial	Aerospace/Defense	554.50	-6.50 (-1.16%)
Barclays PLC	BARC:LN	Financial	Banks	191.90	+0.90 (+0.47%)
Barratt Developments PLC	BDEV:LN	Consumer, Cyclical	Home Builders	608.50	-2.50 (-0.41%)
BHP Billiton PLC	BLT:LN	Basic Materials	Mining	1326.00	-11.00 (-0.82%)
BP PLC	BP/:LN	Energy	Oil&Gas	492.00	-2.25 (-0.46%)
British American Tobacco PLC	BATS:LN	Consumer, Non-cyclical	Agriculture	4994.00	-34.00 (-0.68%)
British Land Co PLC/The	BLND:LN	Financial	REITS	645.50	+8.50 (+1.33%)
BT Group PLC	BT/A:LN	Communications	Telecommunications	262.65	+5.45 (+2.12%)
Bunzl PLC	BNZL:LN	Consumer, Cyclical	Distribution/Wholesale	2087.00	+4.00 (+0.19%)
Burberry Group PLC	BRBY:LN	Consumer, Cyclical	Apparel	1724.00	-15.00 (-0.86%)
Carnival PLC	CCL:LN	Consumer, Cyclical	Leisure Time	4872.00	+4.00 (+0.08%)
Centrica PLC	CNA:LN	Utilities	Gas	147.00	+0.00 (+0.00%)
Coca-Cola HBC AG	CCH:LN	Consumer, Non-cyclical	Beverages	2348.00	+16.00 (+0.69%)
Compass Group PLC	CPG:LN	Consumer, Cyclical	Food Service	1501.00	-4.00 (-0.27%)
ConvaTec Group PLC	CTEC:LN	Consumer, Non-cyclical	Healthcare-Products	208.90	+1.40 (+0.67%)
CRH PLC	CRH:LN	Industrial	Building Materials	2612.00	+33.00 (+1.28%)
Croda International PLC	CRDA:LN	Basic Materials	Chemicals	4272.00	+6.00 (+0.14%)

Selecting the gold button on the main page takes the user to a list of FTSE 100 stocks and displays relevant information such as latest price.

Feature List

After analyzing the interviews with the client and user, and looking at the Student Investor Challenge program (screenshots provided by the user, Ben), the following list was compiled. It consists of features that are intended to be implemented into the simulation. After each feature, ‘client’, ‘user’ or ‘SIC’ is written, indication whether it was the client, user, or Student Investor Challenge that inspired that feature. ‘Inferred’ means that although not explicitly stated, this feature will be necessary for the other features to be added.

- Ability to create and login to accounts (client)
- Ability to join and trade on a team account (client)
- Ability for users to be designated as admins (client)
- Ability for account progress on team and personal accounts to be saved between sessions (inferred from client and user)
- Ability for admins to view teams list (inferred from client)
- Ability for admins to view team details and progress (inferred from client)
- Ability to view real-time information for all FTSE 100 stocks (client and user)
- Ability to select an amount of stocks and buy that amount using an up to date virtual balance, at real current price (client/user)
- Display for all stocks currently held in portfolio (client/SIC)
- Ability to sell stocks in portfolio at real current price (client)
- Graphs to display current day price trends of all stocks (user)
- Graph to show all time price changes of all stocks (inferred from user)
- Ability to create price alerts and be notified when stock reaches current price (inferred from user)
- Interface allowing users to see all current alerts on their account (inferred from user)
- Interface allowing user to see entire trade history (SIC)
- Interface allowing user to see all stocks in the FTSE 100 in a single screen, with details such as price (SIC)
-
- Notes section displayed in trade history and portfolio with reasons for trade decision (user)

Feedback #1 - Client

The following is a brief email discussion with the client regarding the feature list, in order to check that the project is progressing in the right direction before continuing with the analysis and design phase. The replies have been edited for brevity.

"Mr. Butterworth, attached is the General Plan for the project, with the feature list included. Do you have any comments regarding the project so far?"

"Structurally that outline looks very good, and I'm sure if you could get all of those features in the simulation will do everything we need it to do. I would add, however, that some extra information displayed would perhaps make the trading experience a bit more enjoyable for the students. By that I mean things like world events and recent market news, to liven it up a little. On a slightly larger scale, I think features such as currency trading, or trading on commodities and other indices could increase the scope of the program and would be a really positive addition. Students have become quite interested in crypto currency recently and so I think if we could possibly add those trading options in it would really increase the pull of the program from not just Student Investor participants, but to a much wider base of economically minded students.

"I'll take that into consideration. Now that we've discussed potential additions, is there anything there that you don't think needs to be in the program?"

"No not at all – I think it's looking good. I look forward to seeing the program develop! Good luck."

Feature List Updates

Following this discussion, the following have been added to the feature list:

- News section displaying market news (client)
- News section displaying world news and events (client)

Objectives

General Objectives

In general terms, the goal of the project is to create a program that will allow users to trade stocks on the stock market, with a virtual balance, to aid their understanding and develop their trading strategy and ultimately increase success rates in the Student Investor Challenge.

Specific Objectives

The following is a list of objectives that the program needs to meet in order to fulfill the criteria set by the client and the users. These have all been derived from the interviews with the client and user, and from the feature list.

Objective	Specific?	Measurable?	Attainable?	Relevant?
Create an intuitive interface with easy-to-use controls				
Ability to select stocks and view the price info within a 30 second delay of real price				
Visualization of price changes of current stock, and ability to display historic price data				
Ability for users to buy and sell stocks using a virtual balance that is kept up to date.				
Ability for users to trade on a private account or on a team account				
Ability for teacher to create teams and observe their progress				
Implementation of all secondary features stated in the feature list, such as a news feed, trade history, alerts system etc.				

Potential Solutions

There are multiple different ways which the client could solve their problem. The following is a list of

Current Software

The current software used by the students is the actual software that is used in the challenge. This does have the advantage of being already in place, and of course is the real program being used in the challenge, *but is only accessible during the period of the competition* and is made unavailable after the challenge ends. It is therefore not possible to practice on it before hand or after the challenge ends. The features of the actual program used are extremely basic and so students waste a lot of time of manually completing repetitive tasks that could easily be automated, which would avoid students wasting valuable practice time and allow for faster progression. For this reason, the current software does not meet the requirements of the client and is therefore not a feasible solution to the problem.

Online Software

Although there are strong candidates for online software that could perhaps be used, there exists a trend among them that makes them unattractive solutions to the problem; over-complicated interfaces and the inability to work as a team. Although the current software is too basic, online solutions tend to be at the opposite end of the spectrum, with hundreds of superfluous features that students don't need. Online solutions also only provide stand-alone accounts that cannot link to other accounts to form a team. This combination of factors means using online software would not provide a realistic enough practice environment for students. Online solutions are therefore not adequate to solve the problem.

Bespoke Software

Bespoke software is a solution that provides a completely original piece of software that would fulfill the specific requirements of the client, and would be made by professional programmers. Although this software will play an important role for the students, it would be difficult to justify the financial cost of designing it. Furthermore, by contracting programmers from outside the school one runs the risk of facing communication barriers, which would lead to a costly solution that does not completely fit the design description. The department does not have the budget for an investment of this size and so for now, it is not a feasible solution to outsource this project to a software design company.

Visual Basic Solution

A VB solution would still be a bespoke piece of software; however, it would be created in-house and so would be effectively free. With a solution created internally within the school, it allows for much easier communication between the client, end-users and the developer, with the added bonus of being completely free. This means the objectives can be met much more easily and without instructions and requirements being lost in transmission between the developer and the client, and no financial cost to the school A solution programmed in Visual Basic would be able to fulfil all the requirements needed by the client as VB is, despite being basic, able to handle all of the data needed and has the functionality to easily create the required program.

We can see that the other possible solutions are inadequate to solve the problem set by the client, and therefore can conclude that a solution programmed in Visual Basic would be the best option to create software that meets all the requirements set by the client.

Advantages of Visual Basic

- Visual Basic is an intuitive and easy to understand language and the IDE has been optimized for rapid application development.
- Visual Basic has a built-in section for GUI development, making it easy to design and implement an effective and intuitive user interface.
- Visual Basic has built in features that allow for visualization of data in graph.
- Visual Basic has built in features that allow for processing of XML data and data from Access Databases.

Limitations of Visual Basic

- Visual Basic is an object orientated programming language that does not have a mechanism for handling hanging objects, and as such is prone to memory leakage.
- Visual Basic operates sluggishly in comparison to other programming languages, when dealing with network traffic and high volumes of data.[iv]

General Limitations for the Project

- Due to time constraints, it will not be possible to create feeds for every single index and type of security – commodities like gold and oil, and other indexes such as NASDAQ and the FTSE 250 will not be able to be traded on. For this reason, the focus will be on the FTSE 100 index.
- Google Finance provides data that is usually between 1-15 seconds old, however due to latency on the school network and processing time required by the program, data may be up to 30 seconds old.

Data sources and destinations

This project will need data from multiple sources.

- The first source is User1 (the students/users) who will input information such as their username and password, stock symbols and quantities, which will be processed by the program.

- The fourth source of information is provided by User2 (the admin/teacher), which is a different account type that allows for observation of student accounts. The User2 account type can view create teams, account details and portfolios and adjust balances.

- The third source is the Internet, from which the program will extract stock information based on the information input by the user.

- The fourth data source will be a database, from which portfolios and user information such as balance and user ID can be extracted, based on login information provided by the user.

Data Input by User1

Input	Process	Output	Destination
Username	Compares username with accounts in database then checks to see if there exists a match	Username exists / Username does not exist.	Ability to progress to password stage
Password	Assuming username exists in database, compares the password against the password in the database, and checks for a match	If there is a match, login is successful and the main program opens, else login is unsuccessful	Ability to open the main program if both the username and the password match
Select Stock	The selected stock is queried in database function to receive an output string of information regarding that stock	A string of information about the selected stock is output, which can then be split into individual details of the stock, e.g. price	The price of the stock is displayed in the StockPrice box. The price of the stock is also mapped onto a graph. The name of the company whose stock is being queried is displayed in the StockName box
Buy Quantity	Perform a multiplication of BuyQuantity and StockPrice	Total price of purchase	Output on the 'Buy' screen to show the user the amount of money they will spend on a purchase

Data Input by User2

Input	Process	Output	Destination
Team Name	Queried against the database to check if they already exist	The name, team code and balance are inserted into the database and therefore a new team is created	The admin has created a new team that is now stored in the database and can therefore be joined by users
Team Code			
Starting Balance	Validated to check if within the bounds of accepted balances		
Selected Team	Selected Team name is queried in the database	All the information relating to that team is returned	A display box on the admin page

Data Retrieved from Database

Input	Process	Output	Destination
Query for user info	The table relevant to the query is searched for the data relating to the query	Login details are returned to the client	Login form
Query for 24-hour graph data		The last 24 hours of data relating to the queried stock are returned to the client.	Graph displayed on the main form
Query for all-time graph data		All data relating to the queried stock are returned to the client	Graph displayed on the main form
Query for trade history		All trade history relating to the queried account or team is returned to the client	Trade history section on the main form
Query for open positions		The portfolio relating to the queried account or team is returned to the client	Portfolio section on the main form
Query for alerts		All alerts relating to the queried account or team are returned to the client	Alerts section on the main form

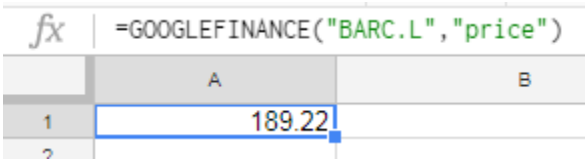
Data Retrieved from the Internet

Data Source

The data needed for the program will be collected from the Stock Market. Google Finance is one medium through which this data can be retrieved. Inside Google Sheets, the GOOGLEFINANCE function can be called to retrieve information about a stock. The syntax for the function is as follows;

```
GOOGLEFINANCE("Symbol","Attribute")
```

For example, to retrieve the current price of Barclays shares we can call the price attribute, using the symbol for Barclays, which is BARC. This is called in the function area for the cell, as shown in the example below

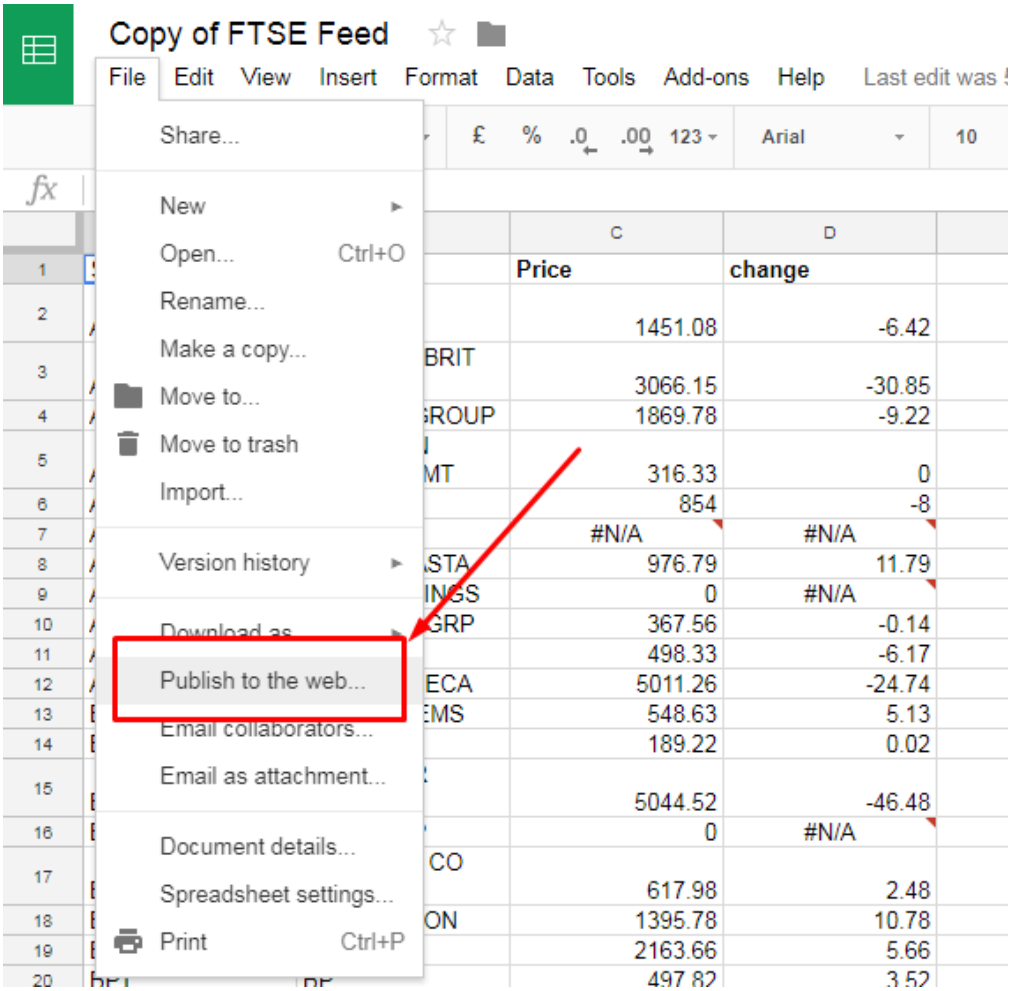


By inputting all the stock symbols for the companies listed in the FTSE 100, and using the attributes “name”, “price” and “change”, we can create a spreadsheet that acts as a live feed for all prices in the FTSE 100, as well as displaying the full name of the company and the intra-day change in price.

	A	B	C	D
1	**Symbol**	**Name**	**Price**	**change**
2	AAL.L	ANGLO AMERICAN	1451.08	-6.42
3	ABF.L	ASSOCIAT BRIT FOODS	3066.15	-30.85
4	ADM.L	ADMIRAL GROUP	1869.78	-9.22
5	ADN.L	ABERDEEN ASSET MGMT	316.33	0
6	AGK.L	AGGREKO	854	-8
7	ANTO.L	ANTOFAGASTA	976.79	11.79
8	ASHM.L	ASHMORE GRP	367.56	-0.14
9	AV.L	AVIVA	498.33	-6.17
10	AZN.L	ASTRAZENECA	5011.26	-24.74
11	BA.L	BAE SYSTEMS	548.63	5.13
12	BARC.L	BARCLAYS	189.22	0.02
13	BATS.L	BRIT AMER TOBACCO	5044.52	-46.48
14	BLND.L	BRIT LAND CO REIT	617.98	2.48
15	BLT.L	BHP BILLITON	1395.78	10.78
16	BNZL.L	BUNZL	2163.66	5.66
17	BP.L	BP	497.82	3.52
18	BRBY.L	BURBERRY GROUP	1751.65	-9.35
19	BT-A.L	BT GROUP	249.98	4.48
20	CCL.L	CARNIVAL	5010.08	0.08
21	CNA.L	CENTRICA	163.38	3.18
22	CPG.L	COMPASS GROUP	1528.56	-9.44
23	CPI.L	CAPITA	477.17	7.27
24	CRDA.L	CRODA INTL PLC	4275.19	-44.81
25	CRH.L	CRH PLC	2645.75	-27.25
26	DGE.L	DIAGEO	2581.5	-19.5
27	EMG.L	MAN GROUP	195.68	-0.82
28	EVR.L	EVRAZ	294.41	11.21
29	EXPN.L	EXPERIAN	1552.9	2.9
30	FRES.L	FRESNILLO	1347.55	54.55
31	GFS.L	G4S	260.86	0.86
32	GKN.L	GKN	304.77	1.77
33	GLEN.L	GLENCORE INTL	363.5	1.4
34	GSK.L	GLAXOSMITHKLINE	1300.58	-2.42
35				

This spreadsheet will be the main source of the data for the real-time section of the program, which displays a live feed of the prices of the stocks.

Visual Basic supports XML, so to retrieve the data, we must can pull it from the document by converting the document into an XML-based RSS feed. For this to work, we need to publish the document;



Once the document is published, it can act as an RSS feed. Using a query link, we can query the document for information;

https://spreadsheets.google.com/feeds/list/1LI2Co50o8hTSf_fg1ULoK1dpKae1uJWhDWiLRlv9yaE/1/public/basic?sq=symbol=BARC.L

This link has the symbol “BARC.L” appended to the end, and so will return the information for Barclays stock prices, in XML format:

Inside this XML is the information we’ll need in the program. This XML is structured and so it will be easy to sort through and extract the relevant information using Visual Basic.

Data Destination

The data retrieved from the internet will end up in two locations:

1) Database – The stock data will be recorded into the database roughly every minute to keep a record of the history of each stocks price, so that graphs can be plotted.
2) Simulation – The stock data for a certain stock will need to go directly into the program when a user selects that stock. It could be argued that this is unnecessary as the up-to-date data is already being sent to the database and so the latest entry for the selected stock could just be selected and displayed directly from the database. However, by fetching it directly from the internet it will avoid making an unnecessary connection to the database and avoids complications that may arise from database connection errors, out of date database data and server crashes.

Data volumes

Volumes of Data Input by User

Variable Names	Data Type	Data Size	Description of Data
Username	String	Up to 32 bytes	The username will be a string with a limit of 16 characters
Password	String	Up to 32 bytes	The password will be a string with a limit of 16 characters
BuyQuantity	Integer	Up to 12 bytes	An integer up to 100,000
SelectedStock	String	Up to 12 bytes	A string determining the stock to query – stock symbols are between 4 and 6 characters
BuyComment	String	Up to 512 bytes	A short comment up to 256 characters

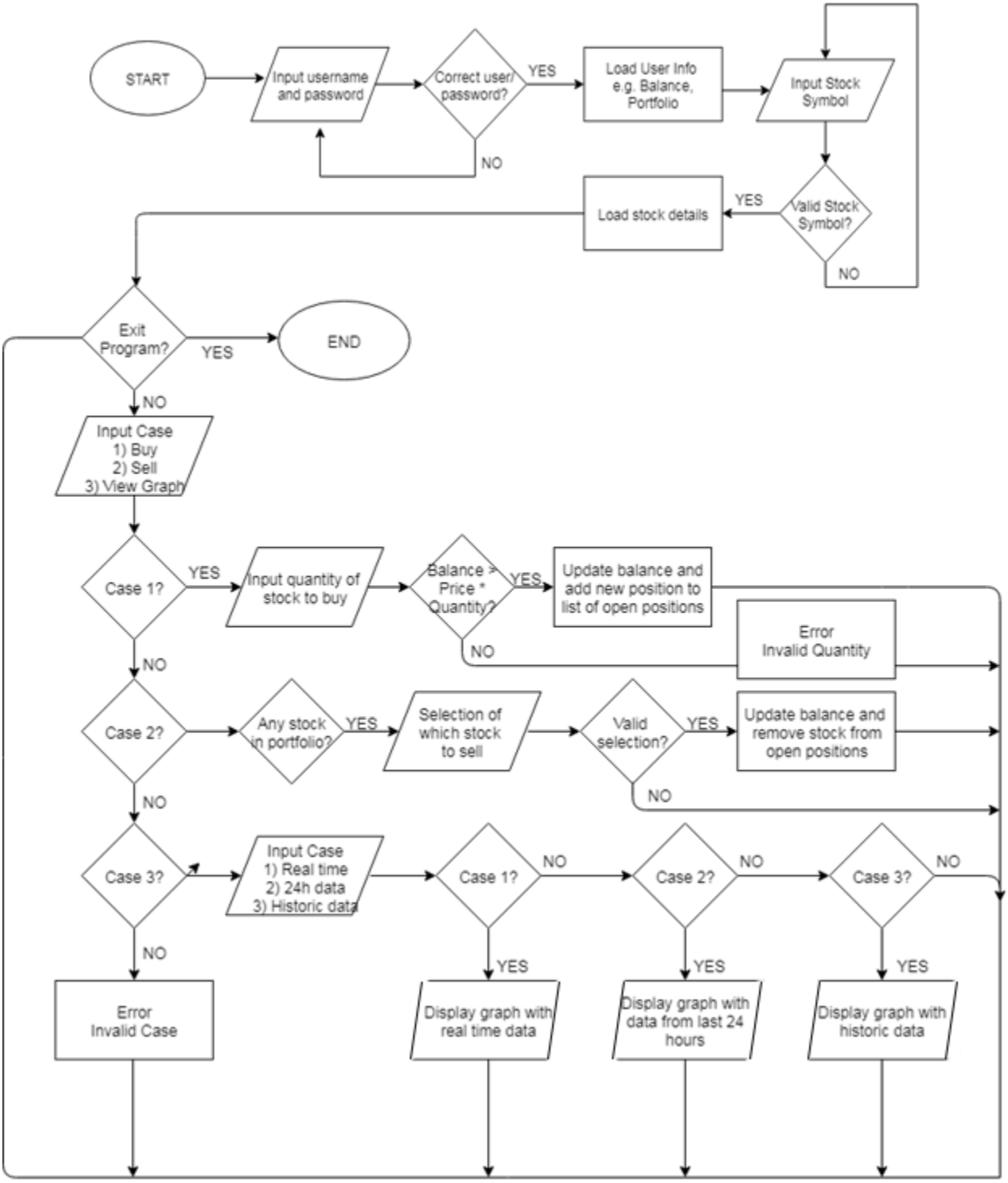
Volumes of Data Retrieved by Program

Data	Data Type	Data Size	Description of Data
Stock XML of a specific stock	String	1954 bytes	The raw XML data retrieved from the Google Sheets file via an RSS feed
Extracted string from the XML	String	Between 32 and 64 bytes	From the raw data, the string of relevant information can be extracted. The size of this data varies depending on the length of the company name and the value of the stock

Last 24 hours of price data for a specific stock	Array of Integers	Between 3-6 Bytes for each integer, with 450 minutes in a trading day. Therefore between 1350 and 2700 kb for each stock.	There are 450 minutes between 9:00am and 4:30pm, and the price will be recorded into a database for each minute. Therefore, when retrieving the data for the last 24 hours of trading there will be up to 2700 kb per stock

Proposed solution

Flowchart for Proposed Solution



Design

General Plan

The aim of the project is to create a program that a teacher can create a team on, then invite students to join their team. Students should then be able to create an account using this team invite, and then login to their account. Once logged on, the students should be able to buy and sell stock using their virtual balance, with the program providing information to help users to judge their trading decisions.

The program will have a database that stores all the data needed, such as account information and the latest stock data.

To fetch the stock data even when the program is not running, there will need to be a secondary program, which will need to be run 24/7 on a server so that it is able to fetch constant price updates.

Feature List

To break this down further, the following is a feature list of all the features that will be included in the program. All of these features are derived from the interviews with the client (client) and user (user), or from researching the Student Investor Challenge software (SIC).

- Ability to create and login to accounts (client)
- Ability to join and trade on a team account (client)
- Ability for users to be designated as admins (client)
- Ability for admins to view teams list (inferred from client)
- Ability for admins to view team details and progress (inferred from client)
- Ability to view real-time information for all FTSE 100 stocks (client and user)
- Ability to select an amount of stocks and buy that amount using an up to date virtual balance, at real current price (client/user)
- Display for all stocks currently held in portfolio (client/SIC)
- Ability to sell stocks in portfolio at real current price (client)
- Graphs to display current day price trends of all stocks (user)
- Graph to show all time price changes of all stocks (inferred from user)
- Ability to create price alerts and be notified when stock reaches current price (inferred from user)
- Interface allowing users to see all current alerts on their account (inferred from user)
- Interface allowing user to see entire trade history (SIC)
- Interface allowing user to see all stocks in the FTSE 100 in a single screen, with details such as price (SIC)
-
- Notes section displayed in trade history and portfolio with reasons for trade decision (user)
- News section displaying market news (client)
- News section displaying world news and events (client)

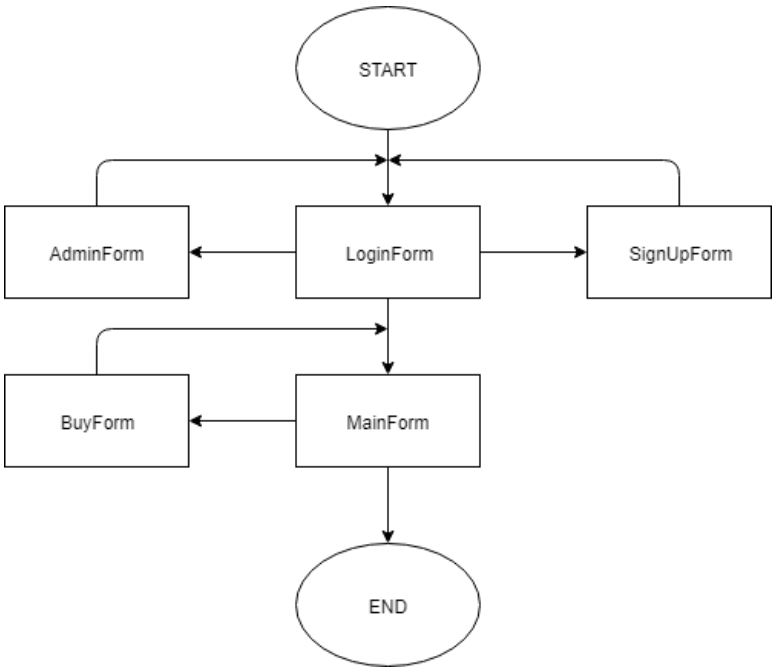
System Design

From the general plan we can see that there are 5 main forms that are needed:

- **LoginForm** – the first form shown to users. From here users can login to their account, or create a new account
- **SignUpForm** – this form should allow users to create a new account that can then be logged in to. They must be able to link their account to a team here.
- **AdminViewForm** – this form should be used by teachers, and should provide an overview of all of the teams that they have created, showing details about the progress of each team and the details of each user in each team. AdminView should also allow for creation of new teams.
- **MainForm** – the main form should be the heart of the program and will contain most of the core functionality of the program. Features should include the ability to buy and sell stocks, an overview of a students or teams portfolio, live feeds for the latest stock prices and news, the users trade history, and the ability to create alerts for stock prices.
- **BuyForm** – from the main form, the user should be able to buy stocks, once they have chosen a stock to invest in. This should open a new form called BuyForm, which should allow users to view the name and price of the stock they are investing in, and to select the quantity they wish to buy.

Program Flowchart

The flowchart shows the flow of the forms, and the sequence in which they are loaded and viewed by the user. The normal program sequence would be LoginForm -> MainForm -> End, with AdminForm, SignUpForm and BuyForm all being optional interfaces for already-registered users.



Data Flow

The following table shows the data flow within the program. Each form will have data being input into it, and will have to process that data to create an output. This table details those inputs, processes and outputs specifically.

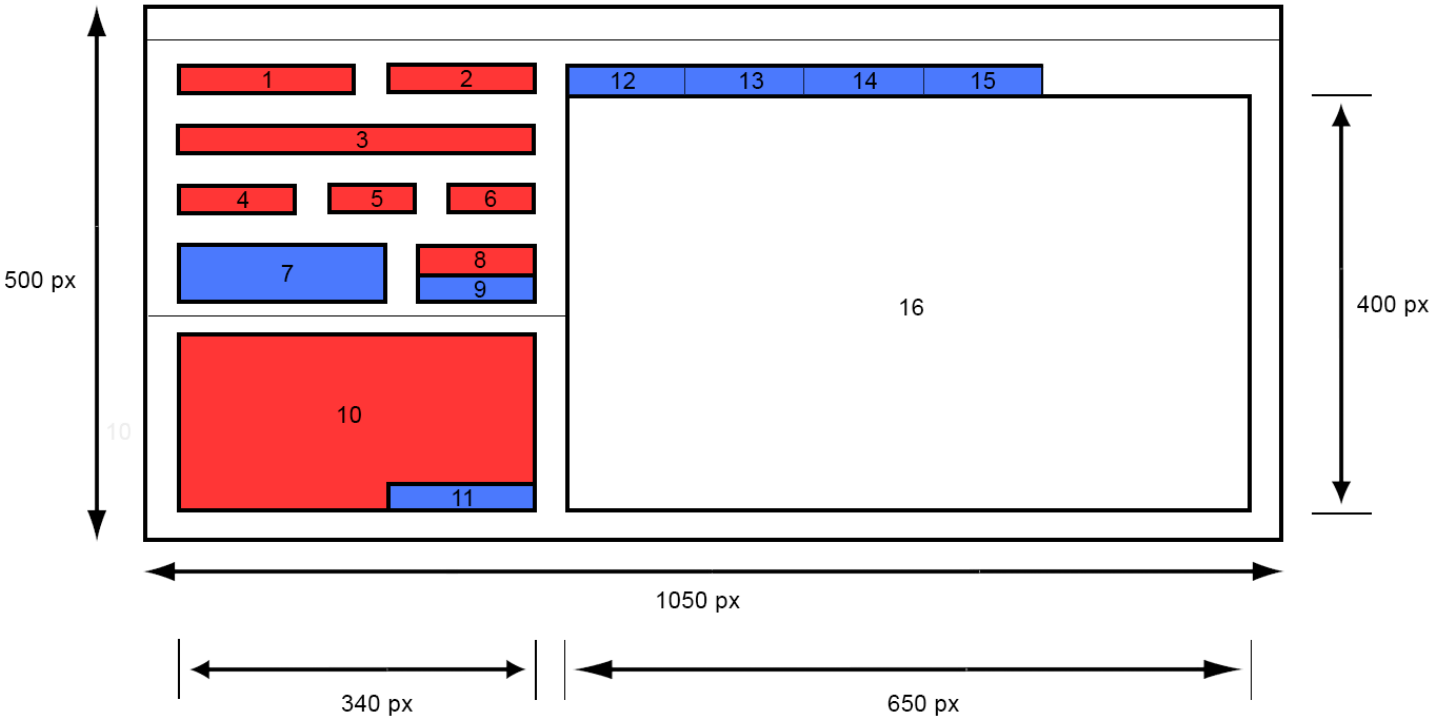
Form Name	Input	Process	Output
LoginForm	Username (str), Password (str), TeamMode (bool)	When the user clicks the login button, the program will query the database twice: Firstly, the program will scan all usernames in the database to check for a match with the input username. Secondly, if a username match is found, then a query of the password related to the stored username is performed. If this password matches the password input by the user, then a Boolean LoginSuccess will be set to true.	If LoginSuccess is true, then the user is shown the main form. The Boolean TeamMode will be set to true or false depending on the Boolean in the LoginForm. If LoginSuccess is false, then access is rejected.
SignUpForm	Username (str) Password (str) Email (str) TeamCode (str)	Firstly, the username will be queried in the database, to check if that username already exists.	If the username already exists, the sign up is rejected. If it doesn't exist, then an account is inserted as a new entry into the accounts table of the database.
AdminViewForm	TeamName (str) TeamCode (str) StartingBalance (int)	When 'Create New Team' is clicked, a validation check will be carried out on TeamName and TeamCode, to check there does not exist a team with these details already. Then a connection to the database is opened and the data is inserted into the respective columns.	If validation is successful, a new team is created in the database.
MainForm	SelectedStock (str)	When the user selects a stock, the program will fetch the current data for that stock, and	The current data will be displayed in textboxes, such as 'Price' and 'Change' and the

		also the historic data of that particular stock.	historic data will be displayed in graphs to show the price trends of the last 24 hours.
BuyForm	Quantity (int) Note (str)	In the Buy form, the user can select the quantity they want to buy, of the currently selected stock. This is multiplied by the current price.	The result of the price multiplied by quantity will be output in a textbox. When the buy is confirmed, the quantity and total price, as well as other details about the buy will be displayed in a textbox called 'OpenPositions', showing all of the current open positions of the user. The details will also be inserted into the database, so that they can be loaded when the user logs back in again.

Form Layout Designs

The visual aspect of the program is very important, as can be seen in the objectives derived from the interviews with the user and client. It is therefore important that the GUI is thought of and laid out carefully, in order to create an intuitive and easy to use interface that makes the user experience simple and enjoyable. The following are mock designs of the forms that will be discussed with the client and end users in order to discover which design is most suitable for this project. All mock-ups were created in Photoshop, employing the grid tool for spacing and alignment purposes.

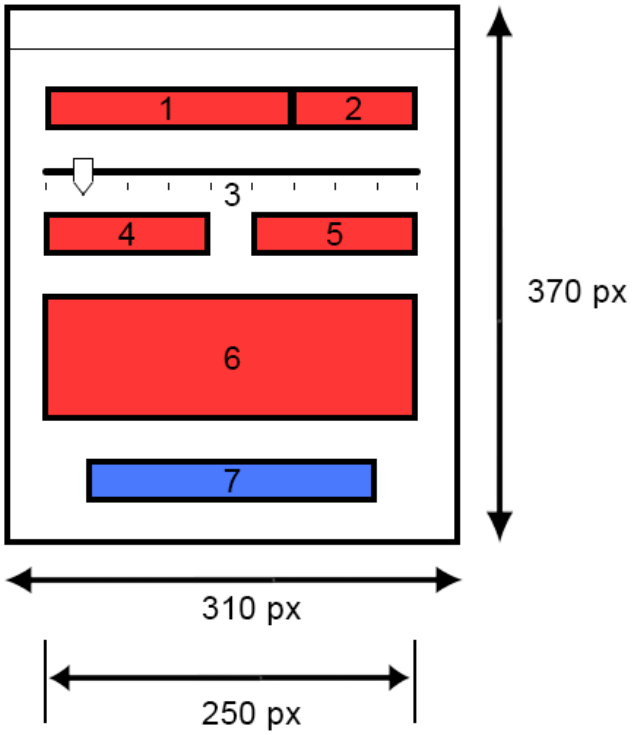
Main Form



1. Input box for stock symbol
2. Users / Teams current balance
3. Full company name that corresponds to the stock symbol selected by the user
4. Price of 1 share in the currently selected company
5. Intraday change of the currently selected share
6. Intraday volatility of the currently selected stock
7. Buy button to open BuyForm
8. Alert price input box for the currently selected stock
9. CreateAlert button to create an alert at the price set in 8 for the currently selected stock
10. OpenPositions textbox that displays all currently open positions
11. ClosePosition button that closes the currently selected position in the OpenPositions textbox
12. Tab1 – displays 24-hour price history graph in 16 for currently selected stock
13. Tab2 – displays all-time price history graph in 16 for currently selected stock
14. Tab3 – displays market news for the FTSE100 in 16
15. Tab4 – displays all alerts currently active for the user/team logged in
16. Displays the currently selected tab – when one tab is selected, all content from other tabs is hidden

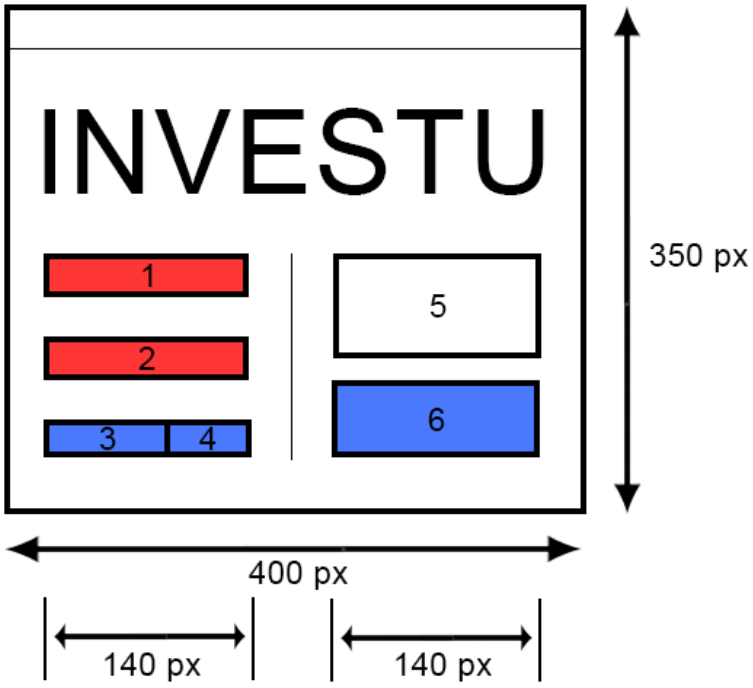
Buy Form

1. Company name
2. Current price per share of currently selected stock
3. A track bar to allow users to select the quantity they wish to buy
4. Output of the track bar – a numeric value of how many shares the user has selected to buy
5. Price of total purchase – calculated using number in 2 multiplied by number in 4
6. Notes section in which users can write comments detailing any useful information regarding the buy
7. Buy button



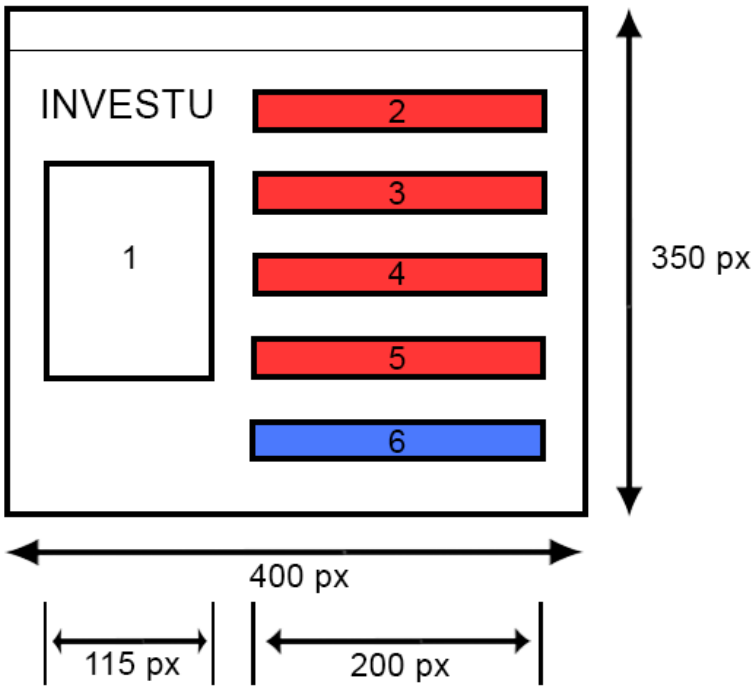
Login Form

1. Username box for existing users to enter their username into
2. Password box for existing users to enter their password into
3. Login button to login to the program and proceed to the main form
4. Cancel button to close the program
5. Informative text regarding signing up
6. Sign up button which loads the sign up form

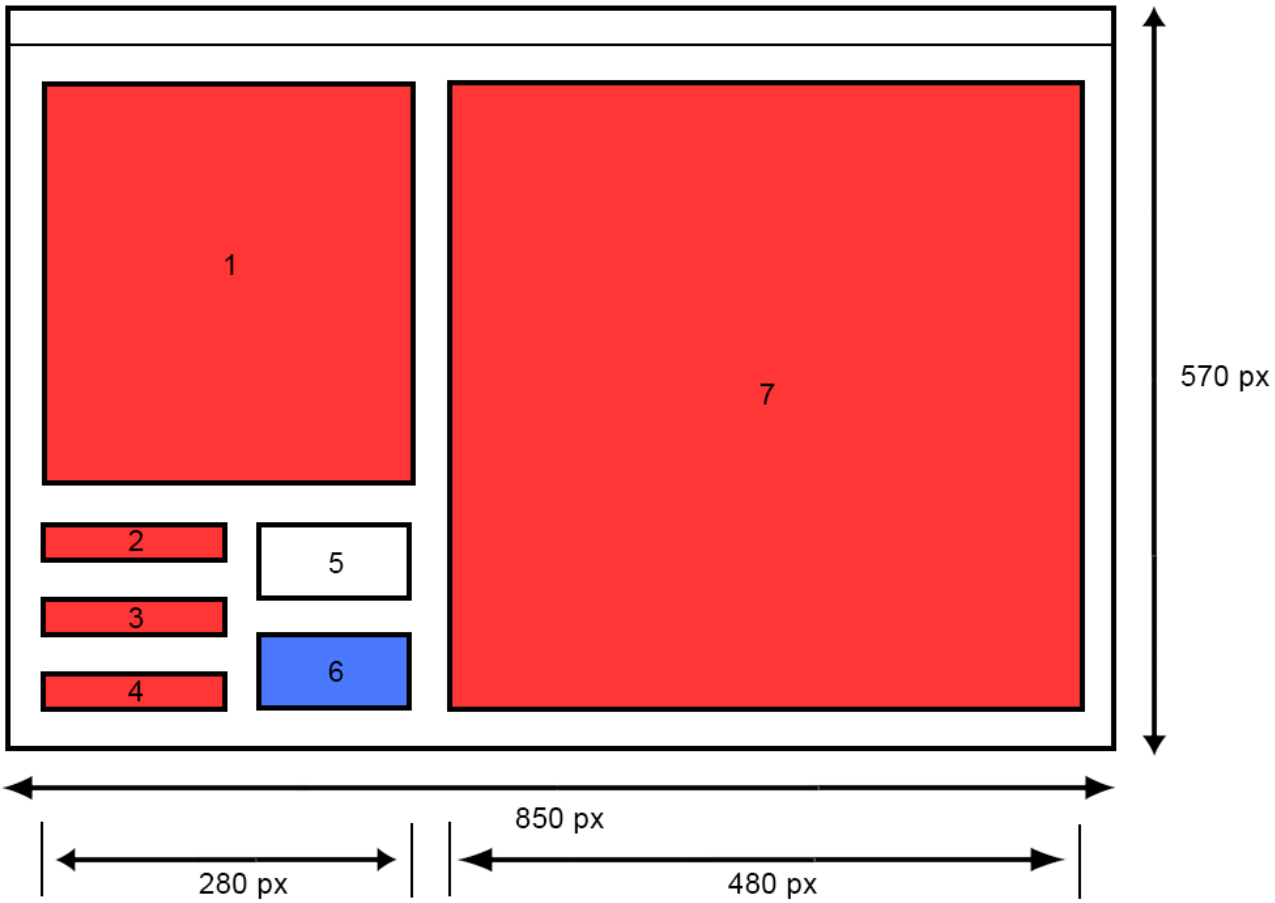


Sign-Up Form

1. Useful information regarding how to use team codes
2. Username box for new users to enter their desired username into
3. Username box for new users to enter their desired password into
4. Email box for new users to enter their email into, for verification purposes and for use when sending alerts
5. Team code box for new users to enter the team code they have been provided, in order to be assigned to a team
6. Sign up button to create an account using the details provided in boxes 2-5.



Admin View Form



1. Teams box – all teams created by this admin are displayed in a textbox here
2. Team Name for new team creation – this will be the name of the new team being created
3. Sign up code for new team creation – this will be the code used by users to join the team about to be created
4. Starting balance – this is where the balance that the team will start with is set
5. Useful information regarding team creation
6. CreateNewTeam button – this inserts the new team into the database and from then on students with the sign up code will be able to join it

Database Design

It is clear that a database is necessary for this program to run, both for the login system and for the ability to remember data between logins, and across multiple accounts. To create a database, we first need to detail each table and the data that it will hold, and then draw relationships between the tables.

Normalisation

To create an efficient database, we must use database normalisation, the goal of which is to "create a set of relational tables with minimum amount of redundant data that can be consistently and correctly modified"[v]. Normalisation comes many different forms. The form we will be applying is 3rd Normal Form. However, because each stage of normalisation depends on the previous stage, we must first start at 1st Normal Form, and move upwards

1st Normal Form

1NF dictates that all attributes in tables within the database must be atomic data. That is, there cannot be field that holds two values. Each field name must be unique and there must also be no repeated data.

2nd Normal Form

A database that is in 2NF must be in 1NF, but with an additional rule. The additional rule is that no non-prime attribute is dependent on a key within the table. That is, if an attribute is a subset of the key, then it should not be stored in a table.

3rd Normal Form

A table is in 3NF if it is first in 2NF, and then for every dependency, either the dependent variable is a super key of the table, or the thing that is being depended on is a prime attribute of the table. Another way of saying this, is that 3NF requires there be no non-key attributes that depend on other non-key attributes.

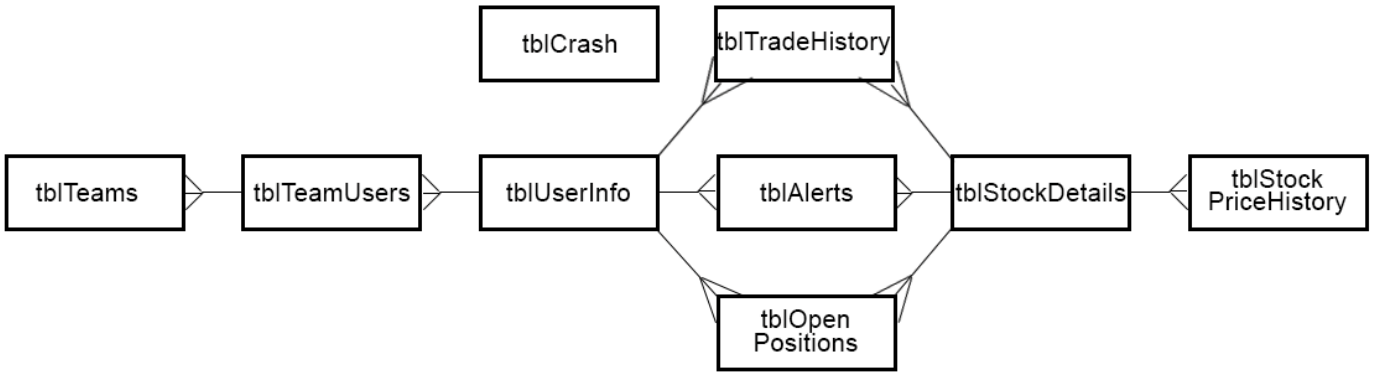
Following these rules of normalisation will lead to an efficient and compact database that can be queried quickly, allowing the program to run as smoothly and fast as possible.

Tables

From the feature list derived from researching stock market simulators and the interview with the client and user, we can extract the information we need to structure a database for the program. The tables will be as follows;

1. tblUserInfo – for storing account information
2. tblTeams – for storing information about teams
3. tblTeamUsers – a link table storing all of the users in each team
4. tblStockDetails – contains all of the information of the stocks in the FTSE 100
5. tblStockPriceHistory – contains the price history of every stock
6. tblTradeHistory – a log of all trades made on the program
7. tblAlerts – contains information of all alerts currently set in the program
8. tblOpenPosition – contains information of all open positions currently in the program
9. tblCrash – a log of all crashes that occur in the program, for debugging purposes

Entity Relationship Diagram



Tables – Breakdown

tblUserInfo						
AccountID	Username	Passwrd	Balance	Admin	TeamName	Email

tblUserInfo – This will be one of the most vital tables and will be used by the program when logging in. This table will store all of the users information, including their login details, balance and team ID. Their login details will be used to validate a login attempt, after which their balance will be loaded to allow them to continue to trade. If the user runs the program in team mode, then the team ID will be used to identify which team they are in, and then allow them to login to the account of the correct team.

tblTeams			
TeamID	TeamName	UserTeamsID	Balance

tblTeams – This will contain information about teams, including their unique Team ID, set by the admin, which will be used by users to join the team and to uniquely identify the team. There will a link table linking to tblTeamUsers which will store the ID of the team members. If there are less than 4 entries in the link table that have a specific TeamID, then the program will know that the team is not full and users will be allowed to join. The balance works the same way as a user balance, with the exception that the team balance is shared between up to 4 accounts, whereas the user balance can only be affected by that user when they're logged into single user mode.

tblTeamUsers	
TeamID	AccountID

tblTeamUsers – This table is a link table that links up to 4 accounts to a team. Every time a user joins a team, an entry will be made into this table linking the AccountID of the user to the TeamID of the team they are trying to join. Before this is done, however, the program must scan the database to find out how many users are already assigned to that team. If there are less than 4, then the user will be allowed to join the team.

tblOpenPositions					
PositionID	AccountID	StockSymbol	StockQuantity	BuyPrice	TradeDate

tblOpenPositions – This table will contain information regarding positions that are currently open on each users account. The table will contain every open position, regardless of whether it was made by a user or a team. The positions will be tied to each account through the AccountID, and to teams through the TeamName column, which, if the position was opened in team mode, will contain the team name, or if it was opened by a user on single user mode, will contain nothing.

tblTradeHistory							
TradeID	AccountID	StockSymbol	StockQty	TradePrice	BuyOrSell	TeamName	Date

tblTradeHistory – This table will be updated every time a buy or sell action takes place. It includes the AccountID of the user that made the trade, and all the relevant information related to the trade, such as quantity and price. Because both buy and sell trades are stored, there is a column called BuyOrSell that can be used to differentiate buy and sell trades at a later date. This information will all be presented to the user so they can review their trade history.

tblStockPriceHistory			
PriceID	StockSymbol	StockPrice	FetchDate

tblStockPriceHistory – In order for the 24hr graph to function, the program needs historic data to read from, which it can then plot into a graph and display to the user. This table stores this data, which will be created by an additional program that will run 24/7 on a server. The table will contain the stock symbol, current price, and the date at which the price was taken. Using this data, the program will be able to fetch all entries where the stock symbol is the symbol of the company that the user is interested in investing in, and then use the price and date of each entry to plot a graph.

tblStockDetails				
StockSymbol	StockName	MarketSector	Price	Change

tblStockDetails – One feature of the program will be list, similar to that in the Student Investor program, that shows the symbol, name, market sector, price and change of each stock, in one scrollable box. The content for that box will be sourced from this table, which will be updated continuously with the latest prices by the additional program running on a server.

tblAlerts					
AlertID	AccountID	StockSymbol	AlertPrice	TeamName	upOrDown

tblAlerts – When the player sets an alert, they may log off their account before the price gets to the alert price, and so they will have no way of knowing whether the stock they were interested in has passed said price. For this reason, alerts need to be stored in the database so that they can be processed even when the program is closed. Alert processing will take place on the additional program on the server.

SQL Queries

Now that the list of table has been created, we can begin to plan the SQL Queries that will be needed and the interactions between tables that will happen. This may be useful to refer back to during development. This list is by no means exhaustive and only contains the most important queries that will definitely be made. There could arise unexpected queries in future that will need to be added.

Form	Query Name	SQL Query Type	Related Table	Description
LoginForm	CheckUserNameExists	SELECT	tblUserInfo	When a sign in attempt is made, this query will be used to check if there is a username is the table that matches the username entered by the user.
LoginForm	LoginSuccess	SELECT	tblUserInfo	Once it is known that there exists a user with the given username, the details of that user must be pulled from the database so that the password can be checked for a match. It will also then be important to know whether the user is an admin or a student, and what team they are in.
AdminView	FetchTeams	SELECT	tblTeams	When admin view is loaded, the admin will need to know all of the teams that they had. For this, a query is needed to select the teams and all of their information from the database, so that they can be displayed for the admin to see.
AdminView	ValidateNewTeam	SELECT	tblTeams	The admin is going to have the ability to create teams. This means there has to be a validation check to make sure the team name and team code are unique. This query will select every team name and team code and then scan through each one, checking if it matches the new team name/code. If there is a match, then the new team name/code is invalid.
AdminView	CreateNewTeam	INSERT	tblTeams	If the new team information is validated, then a new team can be created. This means all of the details have to be inserted into the database, which will be done by this query.
AdminView	FetchTeamInfo - TeamInfo	SELECT	tblTeams	Part of the AdminView part of the program is for admins to be able to see the progress of their teams. From the list of teams loaded with the FetchTeams query, they should then be able to view individual details of these teams. This will be done by the FetchTeamInfo query, which selects all of the currently selected teams information from the database

				and displays it.
AdminView	FetchTeamInfo - UserInfo	SELECT	tblUserInfo	This query is also part of the FetchTeamInfo part of AdminView, in which all of the teams info will be displayed for the admin to see. A query will be needed to select all of users in the team and their information, so that the admin can monitor the team members and their details.
SignUpForm	ValidateTeamCode	SELECT	tblTeamInfo	Similar to the validation in the AdminView, this query will also be used for validation. Before the user can sign up, the program will have to check the team code is valid, by making sure it does not already exist.
SignUpForm	ValidateUsernameTeamCode	SELECT	tblUserInfo	This query will also be used for validation. Before the user can sign up, the program will have to check the username is not already taken, and that the team code selected is available
SignUpForm	CreateNewAccount	INSERT	tblUserInfo	With all of the details given by the user, a new account can be inserted into the database.
SignUpForm	CheckSpaceInTeam	SELECT	tblTeams	The teams will only have a maximum of 4 spaces available, so if a user tries to sign up to a team with 4 players already in, an error must be returned. In order to do this, the content of the team must be retrieved and scanned to check if an empty slot exists.
SignUpForm	AddNewPlayerToTeam	UPDATE	tblTeams	With the AccountID, the new account can be added to the team. This will be useful when a query is run about the team, for example, when finding out how many players are currently in a team.
MainForm	FetchAlerts FetchTradeHistory FetchBalance FetchOpenPositions	SELECT	tblAlerts tblTradeHistory tblUserInfo tblOpenPositions	When the program is loaded, all of the users/teams information needs to be loaded so that they can continue from where they left off.
MainForm	LoadDetailsGrid	SELECT	tblStockDetails	One feature of the program is a grid showing all 100 stocks and their prices. The data for this is stored in the database and regularly updated, and so to get it into the program a

				query is needed.
MainForm	Plot24hrData	SELECT	tblStockPrices	When the program is run, graphs will be plotted. The data for this will be collected when the program is not running and so must be loaded upon loading the program.
MainForm	UpdateBalance	UPDATE	tblUserInfo	When a position is closed (a stock is sold) the users balance needs to be updated, not only on the program but also in the database. This requires and update query to alter the balance in the UserInfo table.
MainForm	ClosePosition	DELETE	tblOpenPositions	When the position is closed, the database needs to be updated, so that the user no longer owns those shares.
MainForm	StoreNewTrade	INSERT	tblTradeHistory	When a position is closed, this counts as a trade being made and so it has to be stored into the trade history table, so the user has an accurate picture of the entirety of their buy/sell history.
MainForm	CreateNewAlert	INSERT	tblAlerts	Part of the main form GUI will be a section for users to create alerts. These alerts must be stored so that they can be monitored externally, and also so they can be displayed at a later date for the user to see.
BuyForm	UpdateTeamBalance	UPDATE	tblTeams	If the user is on a team account, the balance affected when they make a trade, is the team balance, and therefore there needs to be different queries depending on whether the program is running in team mode or not.
BuyForm	UpdateUserBalance	UPDATE	tblUserInfo	This query will update the user balance when a trade is made, however it will only be run if the program is not running in team mode.
BuyForm	CreateNewOpenPosition	INSERT	tblOpenPositions	When a buy is confirmed, the database needs to be updated with the new open position, so that it can be remembered and reloaded when the program is reopened.
BuyForm	CreateNewTradeHistory	INSERT	tblTradeHistory	When a buy is confirmed, the database needs to be updated with the new trade so that when trade history is loaded later, it can be added to the history.

Pseudo Code for Forms

Login Form – Pseudo Code

```
When user clicks Login button

      Connect to database

      If (SELECT Username, AccountID FROM tblUserInfo WHERE Username =
      UsernameBox.text AND Password = PasswordBox.text) returns 1 reply Then

            AccountID = SQLReply.AccountID
            UserValid = True

      If UserValid Then
            SELECT * FROM tblInfo, tblTeams, tblUserInfo
            WHERE tblTeams.TeamID = tblUserTeams.TeamID
            AND tblUserTeams.AccountID = tblUserInfo.AccountID
            AND tblUserInfo.AccountID = AccountID


            If Admin Then
                  Show AdminViewForm
            Else
                  Load AccountID, Username, TeamName, Balance, Email into
                  MainForm

                  Show MainForm
      Else
            Error
      End If

      Disconnect from database

End
```

AdminViewForm – Pseudo Code

```
Connect to database
      SELECT TeamName, TeamID FROM tblTeams
      Load TeamName, TeamID into TeamsList
      Display all items in TeamsList into TeamsListBox
Disconnect from database

When Admin selects team from TeamsListBox

      Connect to database

            SelectedTeamID = TeamsList(TeamsListBox.selectedIndex).TeamID

            SELECT All information relating to selected team FROM
            tblOpenPositions, tblTradeHistory, tblTeams, tblTeamUsers,
            tblUserInfo WHERE tblTeams.TeamID = SelectedTeamID

            Add all relevant team information to a display box

      Disconnect from Database

When admin clicks CreateNewTeam button

      Connect to database

      If (SELECT * FROM tblTeams WHERE teamName='teamnamebox.text' OR
      teamCode='teamCodeBox.text') returns 1 reply Then

            Msgbox("Invalid team code or team name")

      Else

            INSERT INTO tblTeams TeamName, TeamCode VALUES (TeamNameBox.text,
            TeamCodeBox.text)

      End If

      Disconnect from database
```

SignUpForm – Pseudo Code

```
When user clicks SignUp button

      Connect to database

      If (SELECT * FROM tblUserInfo WHERE Username='UsernameBox.text')
      returns 1 reply OR PasswordBox.text doesn't contain (at least 1 number
      and more than 8 characters) Then

            Msgbox("You've chosen an invalid username or password.")

      Else

            INSERT INTO tblUserInfo Username, Password VALUES
            ('UsernameBox.text', 'PasswordBox.text')

      End If
```

MainForm – Pseudo Code

```
When the user logs on
      Determine whether the use is in TeamMode
      Load World News
      Load Market News
      Load Trade History
      Load Alerts


When the user selects a stock symbol
      PriceBox.text = FetchStockPrice(StockSymbol)
      NameBox.text = FetchStockName(StockSymbol)
      ChangeBox.text = FetchStockChange(StockSymbol)

      Plot24hrData(StockSymbol)

Sub FetchStock(Price/Name/Change) (byval StockSymbol)
      Query Google Sheets document for XML data of (StockSymbol)
      Create node list of XML
      Find node containing (Price/Name/Change)
      Return contents of node
End sub

Sub Load(Market/World)News
      Retrieve XML data from News RSS feed
      Split into nodes
      Add content of nodes to string and format
      Display formatted string in news display

End sub

Sub LoadTradeHistory
      Connect to database

      If TeamMode = True then
            SELECT * From tblTradeHistory WHERE
tblTradeHistory.TeamName='TeamName'
      Else
            SELECT * From tblTradeHistory WHERE tblTradeHistory.TeamName='0'
AND tblTradeHistory.AccountID='AccountID'

      Display retrieved database in data grid view
End sub

Sub LoadAlerts
      Connect to database
```

```
      SELECT * From tblAlerts WHERE tblAlerts.AccountID='AccountID'
      Display retrieved data in data grid view
End sub

When user selects a position and clicks 'Sell'
      Balance = Balance + (Current price of stock * quantity owned)

      Connect to database

      INSERT INTO tblHistory (StockSymbol, Price, Quantity, Date) VALUES
('StockSymbol','Price','Quantity','DateTime.Now')

      DELETE * FROM tblOpenPositions WHERE
PositionID='OpenPositions(CurrentlySelectedOpenPosition.UniqueID)


Sub Plot24hrData
      SELECT * FROM tblStockPriceHistory WHERE
StockSymbol='CurrentlySelectedSymbol' AND FetchData >= Date.Today

      For Each Entry retrieved from database
            Add a new point to the graph with X=Price, Y=Date
      Next
End su
```

BuyForm – Pseudo Code

```
When user clicks Buy button

      Price = MainForm.Price
      Quantity = QuantityBox.text
      SelectedStock = MainForm.SelectedStock

      TotalPrice = Price * Quantity

      If Balance > TotalPrice Then
            INSERT INTO tblOpenPositions, tblTradeHistory
            (tblOpenPositions.StockSymbol, tblOpenPositions.StockQuantity,
            tblOpenPositons.StockPrice, tblTradeHistory.StockSymbol,
            tblTradeHistory.TradePrice, tblTradeHistory.BuyOrSell VALUES
            ('SelectedStock',Quantity,Price,'SelectedStock',Quantity,'Buy')
      Else
            Msgbox("Not enough money")
```

Development 1

MainForm – Investu - Development 1

The first development of the program neglects the account system and does not have all of the features laid out in the design. The purpose of this development is to get the main part of the program functioning, the ability to read stock data and display it for the user, in the form of a graph.

Global Variables – MainForm - Investu Development 1

```
Public Class MainForm


    Public Balance As Decimal = 100000000
    Public StockInfo As String = ""

    Public OpenPositions As New List(Of StockAttributes)

    Public TimerInterval As Integer = 5000

    Public symbol() As String = {"RR.L", "AV.L", "BARC.L", "GSK.L", "TSCO.L",
"GLEN.L", "HSBA.L", "ITV.L", "BA.L", "ADN.L"}
    Public s As New Series

    Dim OpenPositionIdentifier As Integer = 0
```

There are a number of global variables that will need to be removed and declared locally, however for the purposes of the first version they will be declared globally.

```
Dim Balance As Integer = 100000000
```

The very first variable declared is balance, which is a count of the users' money in pence, which they currently have available to buy shares. The balance will be affected when a buy is confirmed and when stocks held in the users' portfolio are sold. In future versions, this variable will have a value that is loaded in from the database, where the balance reflects that of the currently logged in user. For this version, the balance is fixed and will reset to 100,000,000 pence every time the program is run.

```
Dim OpenPositions As New List(Of StockAttributes)
```

OpenPositions is built from the class StockAttributes, and will hold the information of all open positions currently held by the user. These will make up the users portfolio.

```
Public Class StockAttributes
```

```
    Public stockSymbol As String
    Public stockName As String
    Public stockValue As Decimal
    Public stockQuantity As Integer

End Class
```

StockAttributes is a class containing attributes related to stocks. Each stock has a StockName, (e.g. BARCLAYS), a StockSymbol (4 letters with '.L' appended e.g. 'BARC.L'), and a value, which is measured in pence and changes regularly. StockQuantity would hold the quantity of shares that the user holds.

```
Dim TimerInterval As Integer = 5000
```

The program will operate around a timer which executes code every 5 seconds. The more frequent this timer ticks, the more up to date the stock information will be. More up-to-date the stock information is, the more accurate the simulation will be, as the share prices users are trading with will be closer to reality. Ideally, this would be set to 1000, to cause the price to update every second, however a request for information from the internet usually takes more than a second. If the timer was set to 1 second, a stack overflow error would eventually occur. For this reason, the timer is set to 5 seconds.

```
Dim Symbols() As String = {"RR.L", "AV.L", "BARC.L", "GSK.L", "TSCO.L", "GLEN.L",
"HSBA.L", "ITV.L", "BA.L"}
```

This array of strings is responsible for holding the Stock Symbols for the stocks that can be traded in the program. The FTSE 100 has 100 companies; however, this list only shows 10 symbols for the purposes of creating a working solution. Later, the symbols will be read in from a text file instead of being stored in the program, and the whole list of 100 stocks will be included.

```
Dim Series1 As New Series
```

A series is VB is a line displayed on a graph. This series will hold the X and Y co-ordinates of each point relating to the price of the data. It is called 'Series1' as in future, there may be a 'Series2/3/4/5' depending on which variable attributes of each stock need to be measured and displayed. Having multiple series allows for multiple lines to be displayed on one graph, which can help with visualization.

```
Dim StockInfo As String
```

The nature of how data is going to be retrieved means that stock information will enter the program as XML data, and will need to be processed. Once it is processed, there will be a string containing all of the information. This string will need processing further to extract each individual piece of information. This composite string, before it is processed, will be stored as StockInfo. StockInfo can then be passed to functions that specialize in splitting this string down into its component parts.

MainForm_Load – MainForm - Investu Development 1

```
Private Sub MainForm_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load

        BalanceBox.Text = "£" & Balance / 100

        For L = 0 To StockDetails.Count - 1
            SelectStockComboBox.Items.Add(StockDetails(L).stockSymbol)
        Next

        UpdateIntervalComboBox.SelectedItem = "5"

        GraphScaleComboBox.SelectedItem = "3"

        CreateChart()
    End Sub
```

When the MainForm loads, this code executes. The program itself is based on which stock is selected, and there is currently no stock selected, so there is no significant code to run yet – this happens when the user chooses the stock symbol they'd like to look into.

```
BalanceBox.Text = "£" & Balance / 100
```

This takes the balance declared earlier and divides it by 100, to exchange it from pence to pounds. The user can no see their balance in an easy to understand format.

```
For L = 0 To Symbols.count - 1
      SelectStockComboBox.Items.Add(Symbols(l))
Next
```

The program will have a drop-down box containing all of the companies in the FTSE 100. From this list, the user can select the stop they wish to view, which will bring up information like graphs and price data. This for-loop populates the drop-down box with every item in the symbols array.

```
CreateChart()
```

CreateChart() is asub-routine that prepares the graph section of the program, ready for data to be added upon selection of a stock.

CreateChart – MainForm - Investu Development 1

```
Sub CreateChart()

        Series1.Name = SelectStockComboBox.SelectedItem
        Series1.ChartType = SeriesChartType.Line
        Series1.BorderWidth = 4
        Series1.XValueType = ChartValueType.DateTime
        Series1.BorderWidth = 2

       Chart1.Series.Add(Series1)
        Chart1.Legends.Clear()
```

This sub-routine configures the display of the graph. The chart is set to a line type with the X-axis set to a DateTime value type. 'Chart1.Series.Add(Series1)' adds this newly formatted series to the graph, called 'Chart1'

Timer1_Tick – MainForm - Investu Development 1

The simulation will constantly be updating with up-to-date stock market information. Therefore, a timer is

```
Private Sub Timer1_Tick(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Timer1.Tick

        Timer1.Interval = UpdateIntervalComboBox.SelectedItem * 1000

        FetchStockDetails(SelectStockComboBox.SelectedItem)

        NameBox.Text = splitStockInfo(StockInfo, "name").ToString
        PriceBox.Text = splitStockInfo(StockInfo, "price").ToString
        ChangeBox.Text = splitStockInfo(StockInfo, "change").ToString

        GraphSettings()
        PlotNewPoint(DateTime.Now, PriceBox.Text)

    End Sub
```

needed so that information can be retrieved at a set interval.

```
Timer1.Interval = UpdateIntervalComboBox.SelectedItem * 1000
```

The first part of the Timer code makes sure that the timer interval is reset back to the time the user set it to. This is done incase an update needs to be forced at any point, in which case the timer would need to be set to an interval of 1 millisecond to cause the update to happen immediately. The timer is reset back to normal each time tick by checking a combobox called 'UpdateInterval' in which the user can select a time in seconds. This is then multiplied by 1000 and set as the timer interval.

```
FetchStockDetails(SelectStockComboBox.SelectedItem)
```

FetchStockDetails is a sub-routine that pulls the raw XML data from the internet and processes it. The relevant information is then stored under StockInfo. This sub-routine is only passed a single parameter, which is the stock symbol of the stock that is going to be queried.

```
NameBox.Text = splitStockInfo(StockInfo, "name").ToString
PriceBox.Text = splitStockInfo(StockInfo, "price").ToString
ChangeBox.Text = splitStockInfo(StockInfo, "change").ToString
```

The user interface for the simulation has 3 text boxes that display information about the stocks being views. These boxes show the company name, the current share price, and the intraday price change of the stock. The simulation therefore needs a method of extracting this information from the StockInfo string retrived via the FetchStockDetails sub-routine. This function is called SplitStockInfo, which can be passed two parameters, StockInfo and the attribute needed to be returned, and it will return that attribute. This is then made into a string and displayed in the textboxes on the user interface. StockInfo is global in this instance of the simulation however in future versions it will not be and it is therefore passed to the function as a argument to make it possible to remove it as a global variable in later versions.

```
GraphSettings()
```

GraphSettings is a sub-routine that adds further formatting to the graph displayed in the user interface. GraphSettings needs to be constantly called as it formats the graph depending on the current price of the stock, which is constantly changing.

```
PlotNewPoint(DateTime.Now, PriceBox.Text)
```

PlotNewPoint is a sub-routine that adds a point to the graph when a stock has been selected. It takes DateTime as the X co-ordinate, and the price of the stock as the Y co-ordinate.

```
Sub GraphSettings()


        Series1.BorderWidth = 2
        Chart1.ChartAreas(0).AxisY.Minimum = PriceBox.Text -
Val(GraphScaleComboBox.Text)
        Chart1.ChartAreas(0).AxisY.Maximum = PriceBox.Text +
Val(GraphScaleComboBox.Text)

End Sub
```

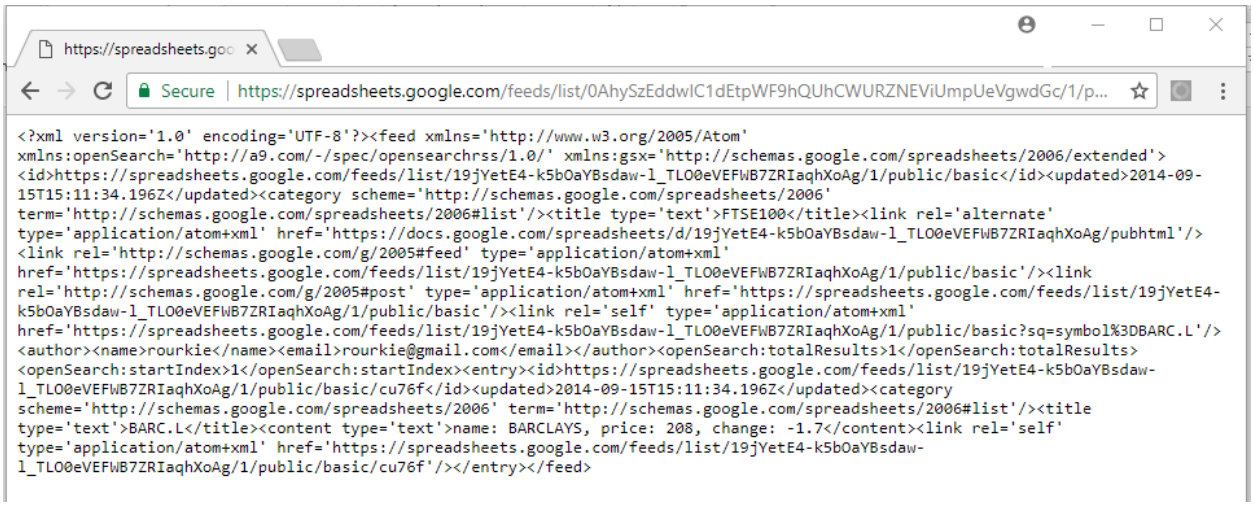
GraphSettings visually updates the graph by changing the scale of the axis. By changing the value of a combo box called GraphScale, the user can effectively zoom in or out of the graph.

Development of the sub-routine FetchStockInfo - MainForm - Investu Development 1

The section focuses on the development of the feature that fetches the relevant stock information for a symbol passed to it. This is vital for the simulation to function, as it will allow for the user to have accurate stock prices and information, and allow for the creation and manipulation of data displays.

The creating of this sub-routine relies on a spreadsheet discussed in Design, under Data Sources and Destinations. This spreadsheet contains all of the stock information for every company in the FTSE 100, and utilizes the GOOGLEFINANCE function to keep stock information up to date.

When we search the Google Sheets URL in a search engine, with a stock symbol appended, this is what is displayed:



When this XML is looked at more closely, a structure emerges:

```
<?xml version='1.0' encoding='UTF-8'?>

<feed xmlns='http://www.w3.org/2005/Atom' xmlns:openSearch='http://a9.com/-
/spec/opensearchrss/1.0/'
xmlns:gsx='http://schemas.google.com/spreadsheets/2006/extended'>

<id>https://spreadsheets.google.com/feeds/list/19jYetE4-k5bOaYBsdaw-
l_TLO0eVEFWB7ZRIaqhXoAg/1/public/basic</id>
<updated>2014-09-15T15:11:34.196Z</updated>


…
…
…
…

<entry>
```

```
<id>https://spreadsheets.google.com/feeds/list/19jYetE4-k5bOaYBsdaw-
l_TLO0eVEFWB7ZRIaqhXoAg/1/public/basic/cu76f</id>

<updated>2014-09-15T15:11:34.196Z</updated>

<category scheme='http://schemas.google.com/spreadsheets/2006'
term='http://schemas.google.com/spreadsheets/2006#list'/>

<title type='text'>BARC.L</title>

<content type='text'>name: BARCLAYS, price: 208, change: -1.7</content>

<link rel='self' type='application/atom+xml'
href='https://spreadsheets.google.com/feeds/list/19jYetE4-k5bOaYBsdaw-
l_TLO0eVEFWB7ZRIaqhXoAg/1/public/basic/cu76f'/>

</entry>

</feed>
```

This XML has one node called ‘entry’. Inside ‘entry’, there are 5 elements. The 4th element contains the information we need:

```
<content type='text'>name: BARCLAYS, price: 208, change: -1.7</content>
```

Therefore to extract this string, we need to get the element from the 4th child of the node ‘entry’. This can be done through this code:

```
StockInfo = node.ChildNodes.Item(4).InnerText
```

We have now successfully extracted the information from the XML and assigned it to the variable StockInfo. StockInfo now has this value:

```
name: BARCLAYS, price: 208, change: -1.7
```

FetchStockInfo – MainForm - Investu Development 1

```
Sub FetchStockDetails(ByVal StockSymbol As String)
        Try
            Dim Document As XmlDocument
            Dim Nodelist As XmlNodeList
            Dim Node As XmlNode

            Document = New XmlDocument()

Document.Load("https://spreadsheets.google.com/feeds/list/0AhySzEddwIC1dEtpWF9hQU
hCWURZNEViUmpUeVgwdGc/1/public/basic?sq=symbol=" & StockSymbol)

            Nodelist = Document.GetElementsByTagName("entry")

            For Each Node In Nodelist
                StockInfo = Node.ChildNodes.Item(4).InnerText
            Next

        Catch ErrorVariable As Exception
            Timer1.Stop()
            MsgBox(ErrorVariable.ToString())
        End Try
```

FetchStockDetails fetches XML data from an online feed, and then searches through the nodes to find relevant information.

```
Try

Catch ErrorVariable As Exception
            Timer1.Stop()
            MsgBox(ErrorVariable.ToString())
End Try
```

The code in this sub-routine is enclosed in a try/catch loop to avoid crashes. The code could potentially throw an exception if the sub-routine tries to fetch information from an invalid link, or if a node is searched that doesn't exist. If an exception is caught, the timer stops. This is to avoid a stack overflow error. A message box is then displayed detailing the error variable for debugging purposes.

```
Dim document As XmlDocument
Dim nodelist As XmlNodeList
Dim node As XmlNode
document = New XmlDocument()
```

Before fetching the XML data, a variable called 'document' is created. Its data type is XMLDocument[vi], and a new instance of it is created. A variable called 'nodelist' is also created, which will be used to hold a list of all the nodes from the file, and 'node' which will be used to hold each individual node.

```
document.Load("https://spreadsheets.google.com/feeds/list/0AhySzEddwIC1dEtpWF9hQUhCWU
RZNEViUmpUeVgwdGc/1/public/basic?sq=symbol=" & StockSymbol)
```

The simulation loads in the XML data from a link to a spreadsheet. This spreadsheet holds all of the information for every stock in the FTSE 100. This spreadsheet is broken down into more detail in the analysis section. At the end of the link, the stock symbol being queried is appended to the end. This means the XML data loaded is only information relevant to the stock selected.

```
nodelist = document.GetElementsByTagName("entry")
```

A list of nodes is constructed, which scans the XML and retrieves all nodes where the tag name is ‘entry’.

```
For Each node In nodelist
      StockInfo = node.ChildNodes.Item(4).InnerText
Next
```

This For-Loop contains the code for extracting the stock information, discussed in the previous section, FetchStockInfo – Development.
The XML file for each stock is structured in the same way. Because of this, we always know where to find the information we want, as it will always be in the same place. We know that if a node list is created that indexes every element where the tag name is ‘entry’, then there will be a single node in the list, and the 4th child node of that node will contain the relevant stock information needed for the simulation. Because we know that this is the same for each XML document, we can hard-code the pathway to this information. There only exists one node in the node list, and so it may seem redundant to use a For-loop to be used, however this allows for expansion later, if more information needs to be extracted from different nodes within the XML.

SplitStockInfo – MainForm - Investu Development 1

When FetchStockInfo is called, a string is returned, in the following format:

```
name: BARCLAYS, price: 208, change: -1.7
```

In the XML this is represented as text and as such it is not able to be broken down any further simply by selecting child nodes; information must be extracted by splitting the string. The purpose of SplitStockInfo is to allow the user to pass a string, StockInfo, and the information they wish to have extracted, e.g. Price, Change or Name, and then for the function to return the correct information.

```
Function splitStockInfo(ByVal StockInfo As String, ByVal Identifier1 As String)

        Dim ArrayList() As String = stockInfo.Split(":")
        Dim SubArrayList() As String = ArrayList(1).Split(",")
        Dim SubArrayList1() As String = ArrayList(2).Split(",")
        Dim StockChange As Decimal = 0
        Dim StockPrice As Decimal = 0
        Dim StockName As String = 0

        Select Case Identifier1

            Case "name"
                StockName = SubArrayList(0)
                Return StockName

            Case "price"
                StockPrice = SubArrayList1(0)
                Return StockPrice

            Case "change"
                StockChange = ArrayList(3)
                Return StockChange

            Case Else
                Return "ERROR RETREIVING INFORMATION"

        End Select
```

```
Function SplitStockInfo(ByVal StockInfo As String, ByVal Identifier1 As String)
```

The two arguments passed to SplitStockInfo are StockInfo - a string containing the information, and 'Identifier1' which tells the function which part of the string the program needs to extract. The 3 supported values for Identifier1 are price, name and change.

```
Dim ArrayList() As String StockInfo.Split(":")
Dim SubArrayList() As String = ArrayList(1).Split(",")
Dim SubArrayList1() As String = ArrayList(2).Split(",")

Dim ExtractedValue As String
```

```
name: BARCLAYS, price: 208, change: -1.7
```

The above string is an example of StockInfo. The first line of code (`Dim ArrayList() As String StockInfo.Split(":")`)' uses VB's built in .split function to split this string where there is a colon, and puts the resultant items into an array. In this case, the array would look as follows:

Array index	Value
0	Name
1	BARCLAYS, price
2	208, change
3	-1.7

The next line (`Dim SubArrayList() As String = ArrayList(1).Split(",")`) performs another split, and puts the items into a new array called SubArrayList(). This time, the split happens where there is a comma, and the string being split is not the whole string, but a part of the string that is found in ArrayList(1). In this case, that would be

```
: BARCLAYS, price
```

Splitting this string where there is an apostrophe would create an array with values as follows:

Array index	Value
0	BARCLAYS
1	price

We have now successfully extracted the name of the stock from the XML file and put it into the array SubArrayList() at index value 0.

Now we need to do the same for price and change, the other two parameters being extracted.

The third line of code (`Dim SubArrayList1() As String = ArrayList(2).Split(",")`) does the same as the second line, but instead of splitting the string in ArrayList(1), it splits the string in ArrayList(2). The value in ArrayList(2) in this case is as follows:

208, change

Splitting this value where there is an apostrophe would result in an array with the following values.

Array index	Value
0	208
1	change

We have therefore extracted all 3 pieces of required information:

Name – stored in SubArrayList(0)
Price – stored in SubArrayList1(0)
Change – stored in ArrayList(3)

```
Select Case Identifier1

            Case "name"
                ExtractedValue = SubArrayList(0)

            Case "price"
                ExtractedValue = SubArrayList1(0)

            Case "change"
                ExtractedValue = ArrayList(3)

            Case Else
                Return "Invalid Identifier"

End Select

Return ExtractedValue
```

Here a Select Case is used to identify which information needs to be returned, and then the value of ExtractedValue is set to the corresponding value – for example if the value of 'Identifier1' is "change", then the value of 'ExtractedValue' is set to ArrayList(3), which is where the value of change is stored, as discussed previously.
The value of ExtractedValue is then returned.

PlotNewPoint – MainForm - Investu Development 1

```
Sub PlotNewPoint(ByVal XValue As String, ByVal YValue As Decimal)
        s.Points.AddXY(XValue, YValue)
End Sub
```

PlotNewPoint is a sub-routine that adds a new points to the series, which in turn changes the appearence of the graph. This is done simply through the Points.Add feature of VB, which takes an X and a Y co-ordinate as arguments, and then creates a new point.

ClosePositionsButton – MainForm - Investu Development 1

```
    Private Sub ClosePositionsButton_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles ClosePositionsButton.Click

                    Dim PositionIndex As String =
OpenPositionsListBox.SelectedIndex

            If OpenPositionsListBox.CheckedItems.Count > 0 Then

                FetchStockDetails(OpenPositions(PositionIndex).stockSymbol)

                UpdateBalance(Balance, splitStockInfo(StockInfo, "price"),
OpenPositions(PositionIndex).stockQuantity)

                OpenPositions.RemoveAt(PositionIndex)

                UpdatePortfolio()

            Else

                MsgBox("Please select the position you'd like to close.")
            End If

        Catch ErrorVariable As Exception
            MsgBox(ex.ToString())
        End Try
    End Sub
```

As the user is able to buy stock in this development of the simulation, it only makes sense that they have the ability to sell stock and have their portfolio updated. This sub-routine performs the sale of stock, and updates the users balance, as well as removing the stock from the users' portfolio.

```
Try

…

Catch ErrorVariable As Exception
    MsgBox(ex.ToString())
End Try
```

This sub-routine has a try-catch in it to catch exceptions and display the error message when a run-time error occurs. These could occur if the user manages to try and sell a stock they don't have, or if there is an out-of-bounds exception.

```
Dim PositionIndex As String = OpenPositionsListBox.SelectedIndex
```

In the simulation, all of the users open positions are displayed in a box. The user can select an open position from this box to sell. This sub-routine works on the basis that the index value of item in the box is the same as the index value of the open position in the OpenPositions() list. This line of code fetches this index value and assigns it a name, 'PositionIndex' which is useful for reducing the visual complexity of this sub-routine.

```
If OpenPositionsListBox.CheckedItems.Count > 0 Then

…

Else
   MsgBox("Please select the position you'd like to close.")
End If
```

The code inside this conditional will only execute when two conditions are met. These conditions ensure a) that there exists an item in the list box, and b) of the items in the box, one of them is ticked. This can be condensed into one condition: `OpenPositionsListBox.CheckedItems.Count > 0,` because in order to have a ticked item in the list box it implies that an item exists to tick. If this condition is met then the following code executes.

```
FetchStockDetails(OpenPositions(PositionIndex).stockSymbol)
```

FetchStockDetails() is a sub-routine discussed earlier in Development 1. In this sub-routine, it is called to find out the value of the stock being sold. This is crucial as the whole concept of the simulation relies on the fact users can buy and sell stocks for different prices. The argument passed to FetchStockDetails() is the StockSymbol, which in this case is stored in the list OpenPositions(). 'PositionIndex' here refers to the index value of the stock being sold. Which this information we can find the stock symbol of the open position being closed, by retrieving the .StockSymbol attribute from OpenPositions, at list index PositionIndex. This then sets the value of StockInfo, a global variable discussed earlier, to the string containing the information about that stock. This can then be split to get the price.

```
UpdateBalance(Balance, splitStockInfo(StockInfo, "price"),
OpenPositions(PositionIndex).stockQuantity)
```

UpdateBalance is a sub routine that takes 3 arguments: the users balance, and the price and quantity of the stocks being sold, and then updates the balance buy performing a price * quantity multiplication and adding the result onto the balance.

```
OpenPositions.RemoveAt(PositionIndex)
```

OpenPositions() is the list holding all of the users currently open positions. This list is then displayed in a list box. To remove an item from the user's portfolio, and then from the list box, the item needs to be removed from OpenPositions() list. Because the list box displaying the open positions and the list holding the open positions both have the same index value, it is possible to use PositionIndex as the index value to remove at in OpenPositions, which will in turn allow the position to be removed from the list box displaying the open positions.
To do this, '.RemoveAt' is used, which removes an item from a list using an index value. The index value in this case is PositionIndex, defined earlier in the sub-routine.

```
UpdatePortfolio()
```

Calling update portfolio will update the visual display in the program that displays the open positions the user has.

UpdatePortfolio – MainForm - Investu Development 1

```
Sub UpdatePortfolio()

      OpenPositionsListBox.Items.Clear()

      For l = 0 To OpenPositions.Count - 1
           OpenPositionsListBox.Items.Add(OpenPositions(l).stockSymbol & " - " &
OpenPositions(l).stockQuantity & " - " & OpenPositions(l).stockValue & " - " &
OpenPositions(l).UniqueID & vbNewLine)
       Next

    End Sub
```

```
OpenPositionsListBox.Items.Clear()
```

In order to keep the index values of the list box and the OpenPositions() list the same, the list box containaing the open positions has to first be wiped, and then the content re-written, so that the index values continues to match up.

```
For l = 0 To OpenPositions.Count - 1
           OpenPositionsListBox.Items.Add(OpenPositions(l).stockSymbol & " - " &
OpenPositions(l).stockQuantity & " - " & OpenPositions(l).stockValue & " - " &
OpenPositions(l).UniqueID & vbNewLine)
Next
```

This for-loop simply lopops through all of the open positions and adds the details to the list box, so that the use has a graphical representation fo the positions they have open in the simualation.

UpdateBalance – MainForm - Investu Development 1

```
Sub UpdateBalance(ByVal Balance As Integer, ByVal Price As Integer, ByVal
Quantity As Integer)

    Balance = Balance + (Quantity * Price)
    BalanceBox.Text = "£" & Balance / 100

End Sub
```

This sub-routine is called in ClosePositions and is responsible for calculating the users new balance. The sub-routine simply performs a multiplication followed by an addition to calculate the new balance, and then updates the visual display to reflect the new balance.

BuyForm - Investu Development 1

When the user wishes to open a position on a stock, they enter the buy form. BuyForm is a display that allows the users to see the stock name and share price, and then select a quantity to buy.

Global Variables – BuyForm - Investu Development 1

```
Public Class BuyForm

    Dim Quantity As Integer
    Dim Price As Decimal = MainForm.PriceBox.Text
    Dim StockSymbol As String = MainForm.SelectStockComboBox.SelectedItem
    Dim StockName As String = MainForm.NameBox.Text
```

The variables declared here are Quantity, Price, StockSymbol and StockName. These values will be used throught BuyForm and so are declared globally here, however in future versions will most likely be defined locally and passed to where they are needed instead.

The values for the these variables are fetched from MainForm, excluding Quantity which is determined by the user in BuyForm. MainForm will be open in the background and so accessible to retrieve information from.

BuyForm_Load – BuyForm - Investu Development 1

```
    Private Sub BuyForm_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load

        Quantity = 1

        QuantityBox.Text = Quantity
        PriceBox.Text = "£" & Math.Round((Price * Quantity) / 100, 2)


        StockDisplayBox.Clear()
        StockPriceBox.Clear()

        StockDisplayBox.Text = Stockname
        StockPriceBox.Text = Price
    End Sub
```

The code here is self explanatory – it mostly deals with the visual appearence of BuyForm.

Quantity is set to 1, as it may cause unexpected results if the user tries to open a position on a stock with a share quantity of 0. The value in the total price box is converted to pounds and displayed rounded to 2 decimal places.

Values are cleared incase there exists values remaining from the last time the form was opened. The values of the StockDisplayBox and StockPriceBox are set to StockName and Price respectively, in order to show the user the stock name and price of the stock they are currently opening a position on.

TrackBar_Scroll – BuyForm - Investu Development 1

```
    Private Sub TrackBar1_Scroll(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles QuantitySlider.Scroll

        QuantitySlider.Maximum = Int(MainForm.Balance / Price)
        Quantity = QuantitySlider.Value
        QuantityBox.Text = Quantity
        PriceBox.Text = "£" & Math.Round((Price * Quantity) / 100, 2)

    End Sub
```

This sub-routine has 'Handles QuantitySlider.Scroll' appended in its declaration which means it is called when QuantitySlider is scrolled left or right. QuantitySlider is a track bar in BuyForm that will be used to determine the number of shares the user wishes to buy.

```
QuantitySlider.Maximum = Int(MainForm.Balance / Price)
```

The maximum value the track bar will scroll to is set to the maximum number of shares the user can buy with their current balance, which is worked out by truncating the value that is a result of dividing their balance by the current price.

```
Quantity = QuantitySlider.Value
QuantityBox.Text = Quantity
PriceBox.Text = "£" & Math.Round((Price * Quantity) / 100, 2)
```

When the track bar is scrolled, these lines of code update the value of Quantity to the current value of the track bar, and then set the QuantityBox value to quantity, so the user can see the number of shares they have currently selected. The total price of the purchase is then updated, using the Price * Quantity calculation

BuyButton_Click – BuyForm - Investu Development 1

The BuyButton in BuyForm is the button the user uses to enter a new position, after they have selected a quantity using the track bar. This button executes code to add the new position to a list of all currently open positions, which is then added to the portfolio box in the simulation, where the user can view all of their currently open positions.

```
    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click

       If MainForm.Balance > (Quantity * Price) Then

              MainForm.OpenPositions.Add(New StockAttributes With {.stockName =
       Stockname, .stockSymbol = StockSymbol, .stockValue = Price, .stockQuantity
       = Quantity})


              MainForm.Balance = MainForm.Balance - (QuantityBox.Text *
StockPriceBox.Text)
              MainForm.BalanceBox.Text = "£" & MainForm.Balance / 100
              MainForm.UpdatePortfolio()

              Me.Close()

       Else

              MsgBox("You don't have enough money to buy that many " & Stockname &
" shares.")

        End If

    End Sub

End Class
```

```
If MainForm.Balance > (Quantity * Price) Then
...
Else
       MsgBox("You don't have enough money to buy that many " & Stockname & "
      shares.")
End If
```

This conditional is used as another layer of error checking, in case the other counter measures were ineffective. The condition simply checks that the users balance is greater than the cost of the position they are trying to open.

```
MainForm.OpenPositions.Add(New StockAttributes With {.stockName =
Stockname, .stockSymbol = StockSymbol, .stockValue = Price, .stockQuantity =
Quantity})
```

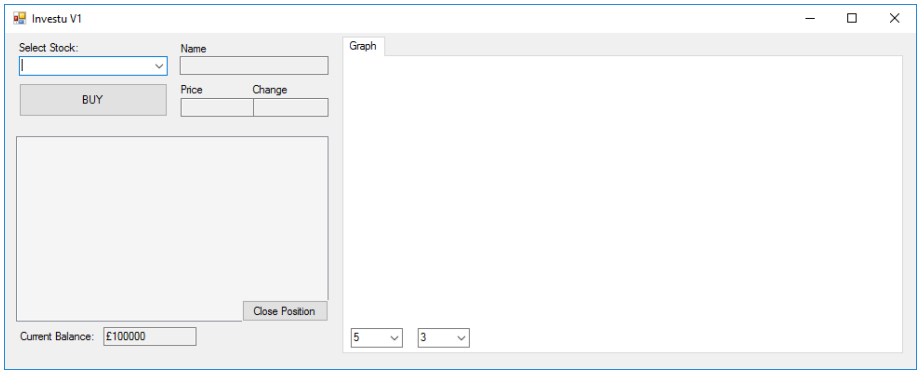
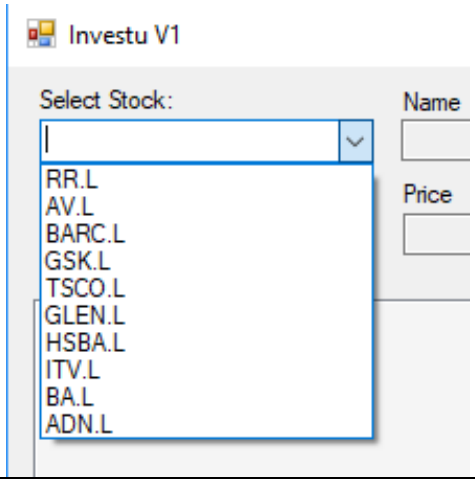
Once the validation has been passed, a new position is opened. This is done using .Add on the 'OpenPositions' list in MainForm. The 4 attributes of the list corrospond to the 4 variables declared earlier in BuyForm – StockName, StockSymbol, StockValue and StockQuantity.

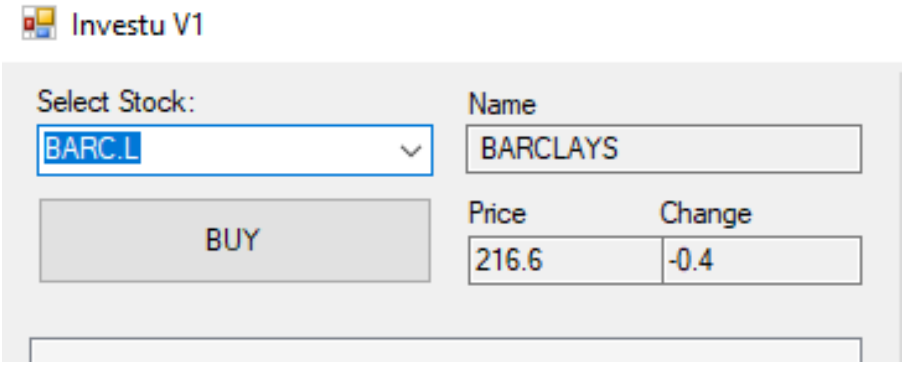
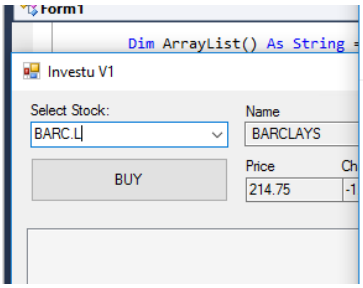
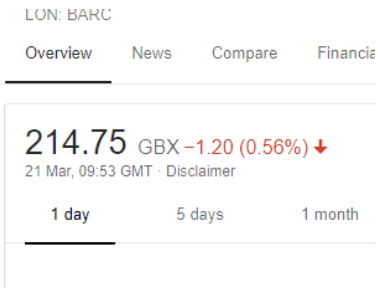
```
MainForm.Balance = MainForm.Balance - (QuantityBox.Text * PriceBox.Text)
MainForm.BalanceBox.Text = "£" & MainForm.Balance / 100
MainForm.UpdatePortfolio()
```

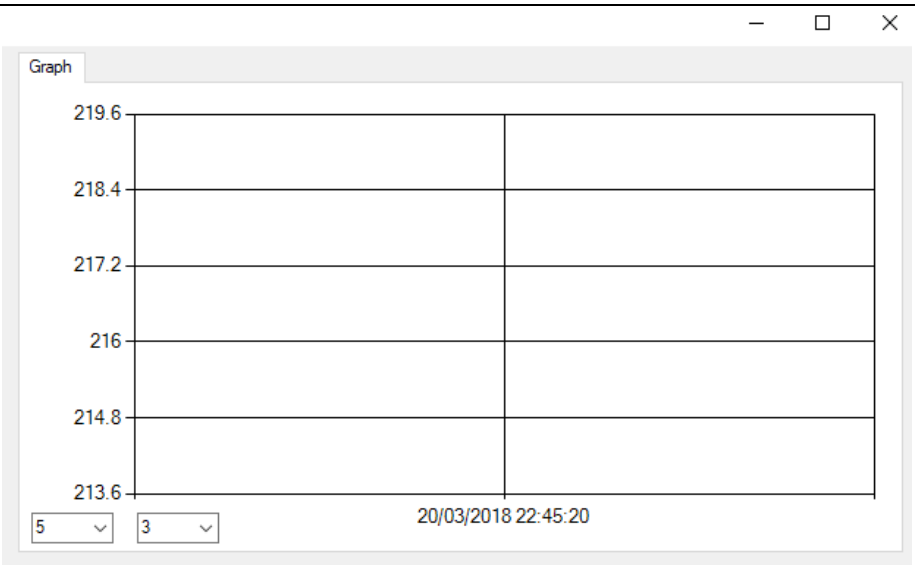
This code simply updates the balance and then sets it in MainForm, and then calls UpdatePortfolio to erase the current contents of the portfolio display box and populate it with the users new portfolio.

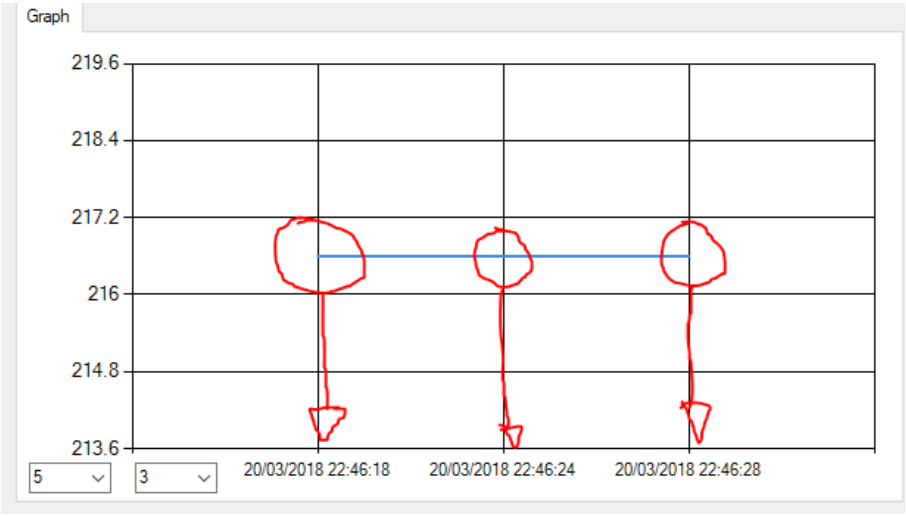
Testing 1 - Investu Simulation – Development 1

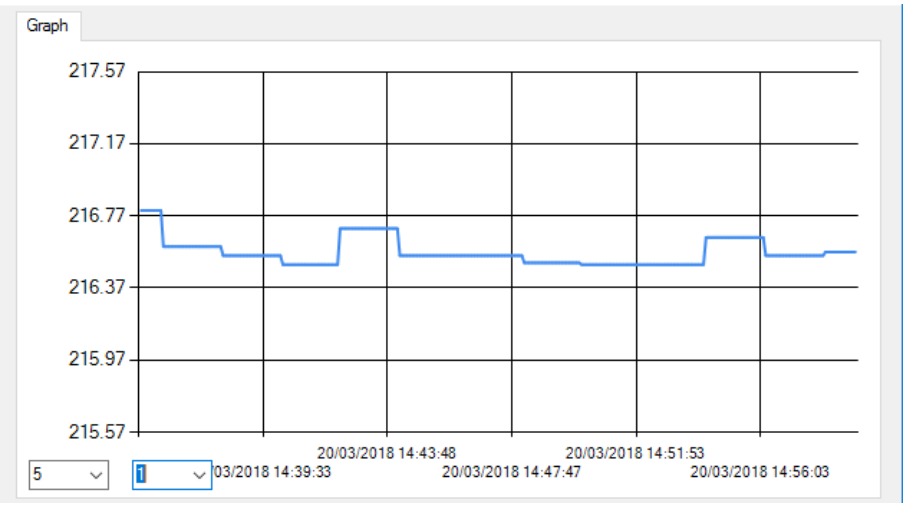
Every sub-routine has been tested individually to ensure it works independently, however these tests have not been shown. The tests displayed are tests that show multiple sub-routines and functions working together to produce the desired outcome.

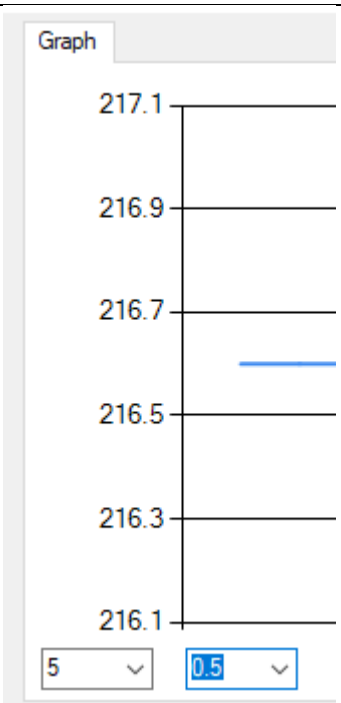
Test Objective	Evidence	Objectives met?
Show simulation loads all of the stock symbols in the Symbols() array to the program, and they are selectable	When the user loads the simulation, the user is presented with the following  user interface. In the top left corner of the simulation is a drop down box labelled 'Select Stock'. When the user selects the drop down menu, a list of stock symbols is presented. These symbols are those found in the Symbols() array discussed earlier in Development 1. Every stock symbol hard-coded into the Symbols() array, are present here in the drop-down menu.  This indicates that the requirements set out in the test description have been met and the test has therefore been passed.	

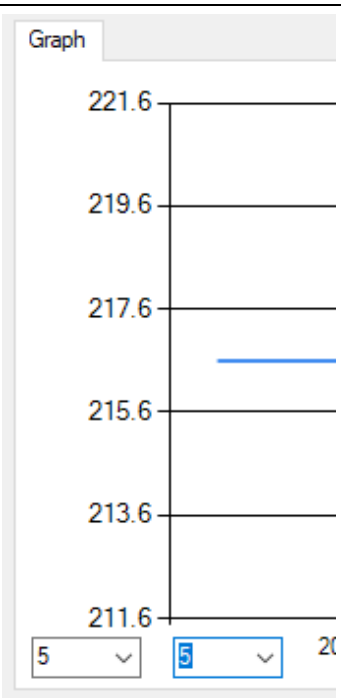
Show when stock symbols are selected, the simulation fetches the information related to that stock from the internet	 Upon selecting a stock symbol, in this case 'BARC.L', the information for the stock is loaded correctly. This can be tested by corroborating the information with another source, such as Googles' built in share pricing feature. (note the price of Barclays' shares changed between taking the above screenshot and the screenshot below)   In the above screenshot we can see that the price displayed in both Google and the Investu simulation is 214.75. This indicates that the requirements set out in the test description have been met and the test has therefore been passed.	

Show simulation structures a graph and styles it on selection of stock	 When a symbol is selected this graph is created. This graph was created when loading in BARC.L, when the price was 216.6. The graph scale goes up to 219.6 and goes down to 213.6, which is correct as the value selected in GraphScaleComboBox (bottom of the graph) is 3. There exists only one point, which is at 216.6 at the time 22:45:20. This graph only has one point because it has only just been loaded, and so not enough time has elapsed for any points to have been plotted, except the point plotted immediately after the graph is created. This indicates that the requirements set out in the test description have been met and the test has therefore been passed.	

Show there is an interval between plotting of graph points determined by the value selected by the user in UpdateIntervalComboBox	 This screenshot was taken just over 10 seconds after a stock symbol was selected. As indicated by the 3 red circles drawn on, there are 3 points plotted in the graph. The first was plotted immediately as the graph was loaded. From the scale on the X axis, we can see that the next point was plotted about 5 seconds later, at the third point was plotted exactly 10 seconds after the graph was loaded into the simulation. This data is in line with what is expected, as the update interval at the bottom of the graph (combo box on the left) is set to 5. This indicates that the requirements set out in the test description have been met and the test has therefore been passed.	

Show simulation populates graph with points, with the X axis measured in time and the Y axis measured in stock price. The price and time of the points is accurate.	 Above is the result of selecting the Barclays stock and leaving the program to run for 20 minutes. Each point represents 5 seconds passed. This graph therefore contains 400 points ((20*60)/5 = 240) This indicates that the requirements set out in the test description have been met and the test has therefore been passed.	
Show that the user can manipulate the graph scale in order to zoom in/out of the graph		

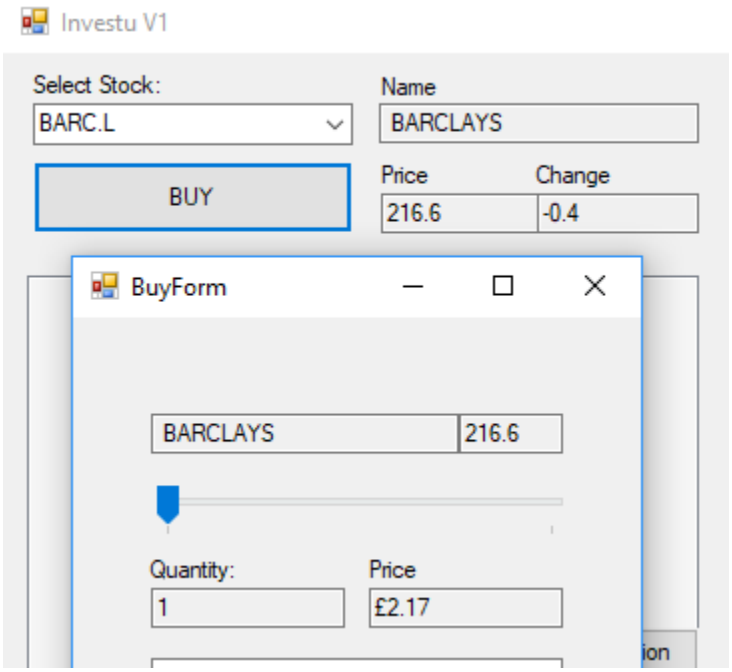


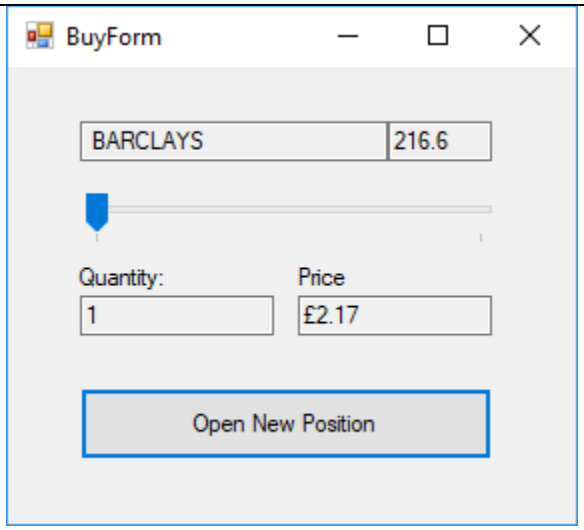


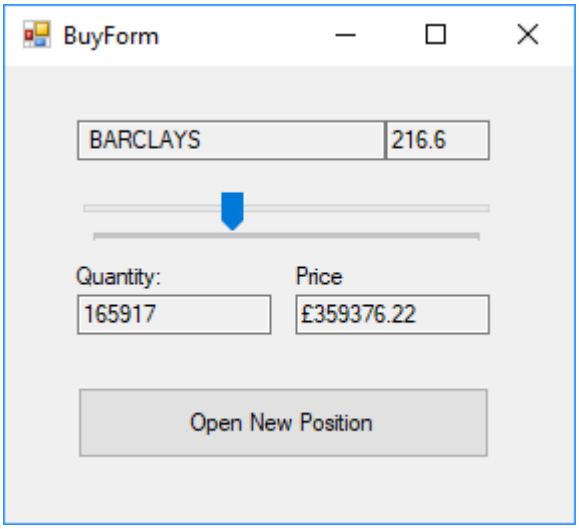
Above are two screenshots of the Y-axis of the graph. The screenshot on the left shows the graph when the GraphScaleComboBox value is set to 0.5. As the price of the stock is currently 216.6 when this screenshot was taken, this means the graph extends to 216.6 + 0.5 and 216.6 - 0.5. This has the effect of zooming in to the graph, so that changes in price look visually larger and are easier to see.

In the right-hand screenshot, the value of GraphScaleComboBox is set to 5, which means the graph is effectively zoomed out. This scale is able to be changed during run-time allowing the user to adjust how they view the graph.

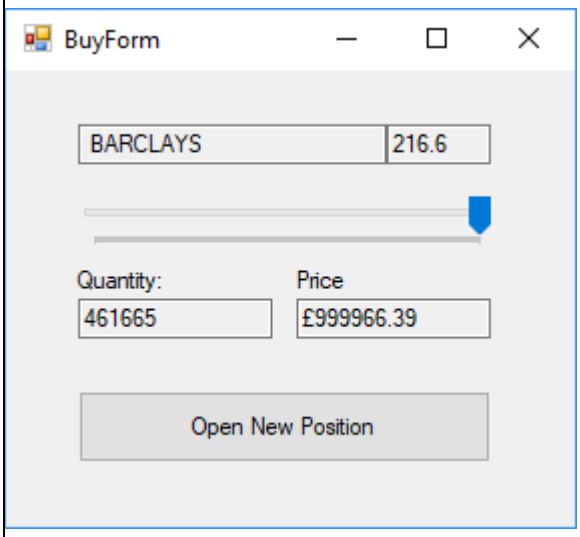
This indicates that the requirements set out in the test description have been met and the test has therefore been passed.

Show that BuyForm loads and displays information related to the stock currently selected in the simulation		

	When clicking on the 'BUY' button in the program, another form appears this form is BuyForm. The form contains the correct name for the stock currently being looked at, which is BARCLAYS in this case. The current price of one Barclays share is also displayed, which in this case is set to 216.6. The initial quantity is set to 1, which is the minimum the bar will slide to. The second price box shows the price displayed in pounds, rounded to the nearest penny. This indicates that the requirements set out in the test description have been met and the test has therefore been passed.	
Show that the BuyForm allows users to select the number of shares to buy, and that the price updates dynamically	 This screenshot shows the initial display of the form upon load. The quantity is set to 1 and the price has correctly been converted from pence to pounds.	

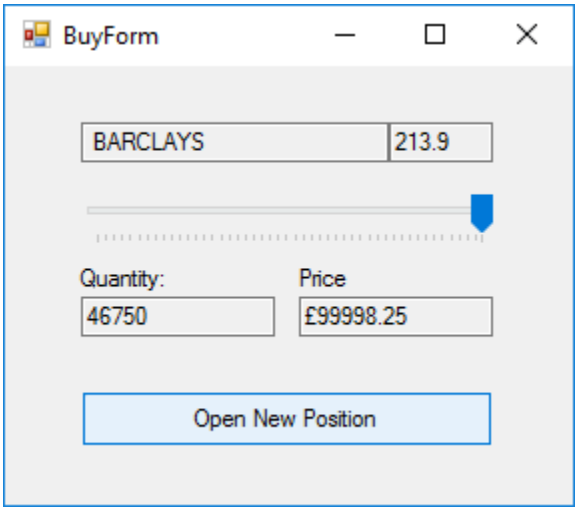
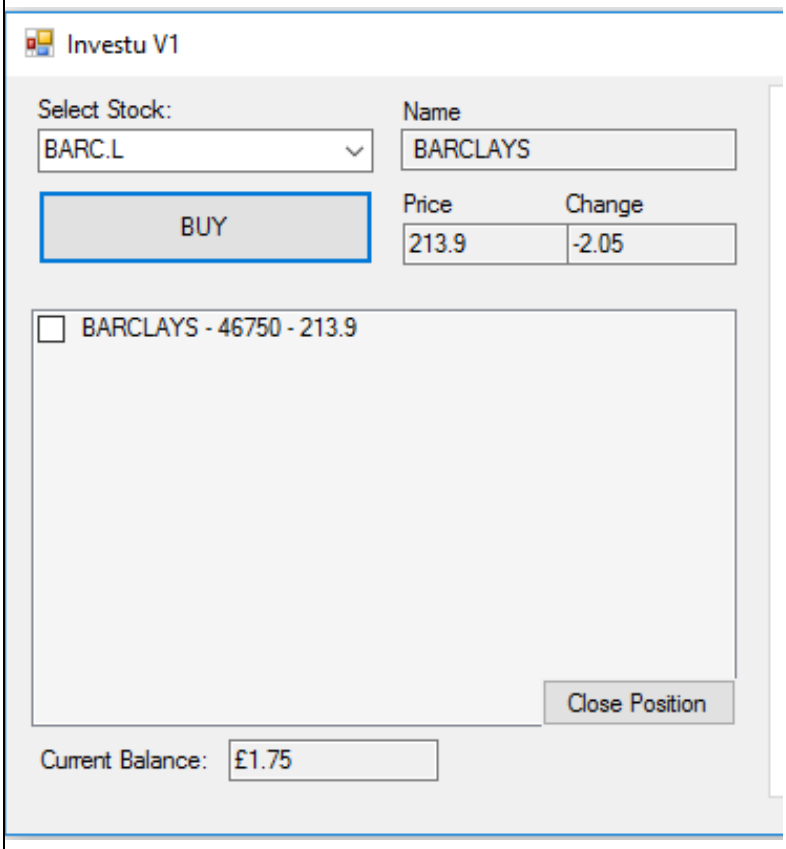


Sliding the scroll bar about 1/3 of the way causes the quantity to update. In this case the quantity is set to 165,917. The price of one share is currently to 216.6. The total price should therefore be displayed as (165917 * 216.6)/100 = 359376.222 = £359,376.22 – which it is.

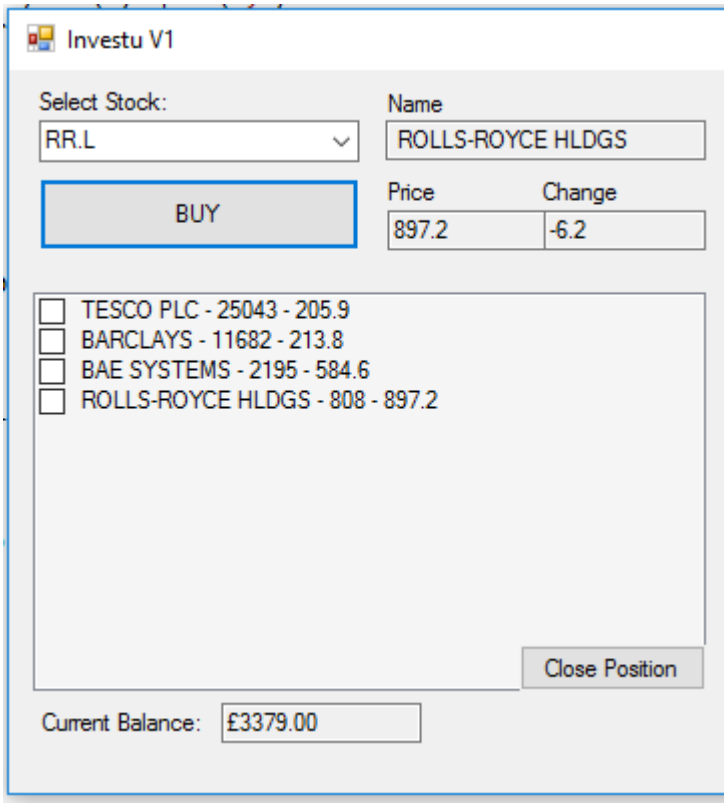


When the scroll bar is set to its maximum value, the quantity is 461665, which is the maximum value that can be bought with a balance of £1,000,000 – the value of balance defined in the development of the simulation.

This indicates that the requirements set out in the test description have been met and the test has therefore been passed.

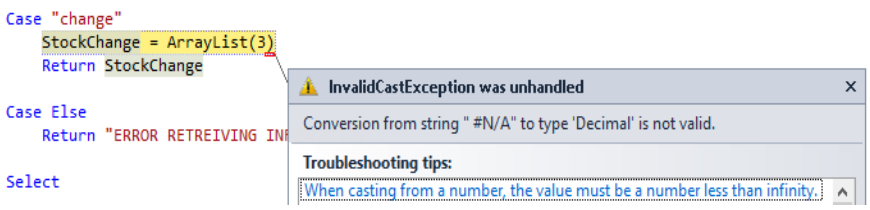
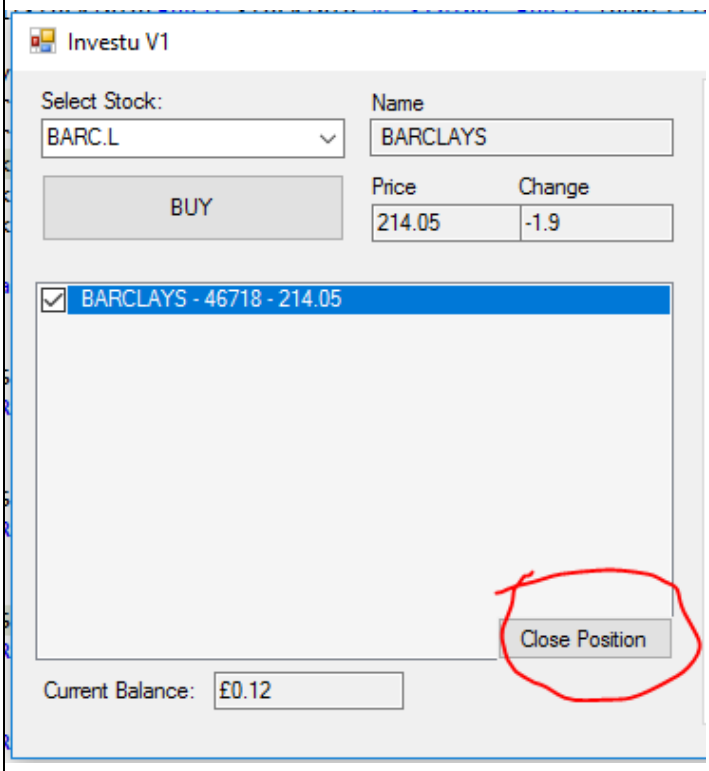
Show the user's buy registers and is displayed in the portfolio box, and the balance is updated correctly	 NOTE: In this test, the value of balance was set to 10000000, which is the same as £100,000, instead of £1,000,000 as in the previous test. In this screenshot, a buy offer is being created. The scroll bar is set to its maximum, and the total price is therefore just under £100,000.  When the button labelled 'Open New Position' is then selected, the selected quantity of stock is bought. 'Current Balance' at the bottom is now £1.75. This is because a single share of Barclays stock costs £2.14, and so £1.75 is the remainder left over that is less than the minimum amount required to buy one Barclays share.	

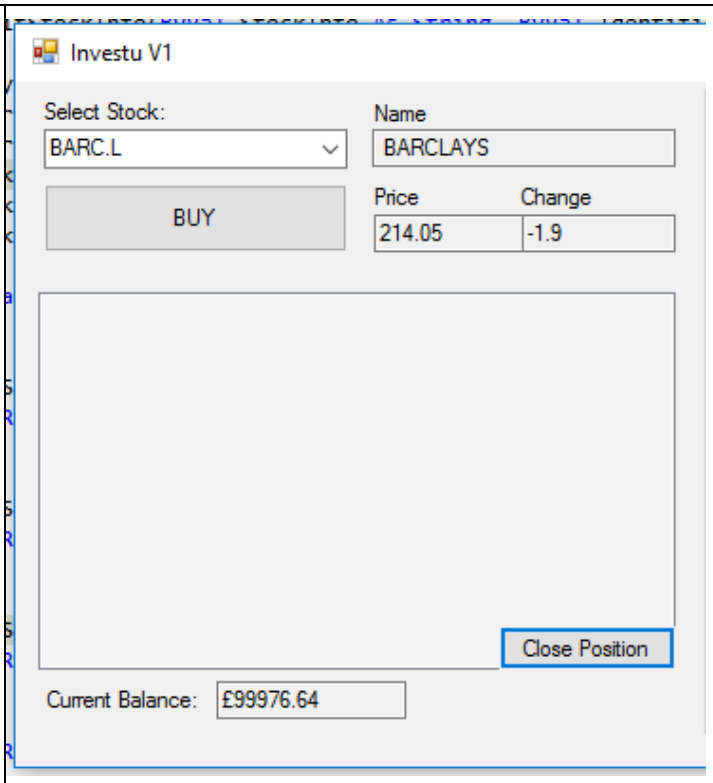
The new open position is then added to the portfolio section of the simulation. The new entry displays 'BARCLAYS – 46750 – 213.9' Which reflects the stock name, quantity bought, and buy price respectively. This format is perhaps not the most user-friendly, however this will be addressed in the following developments, as this development simply serves as proof of concept for the simulation.



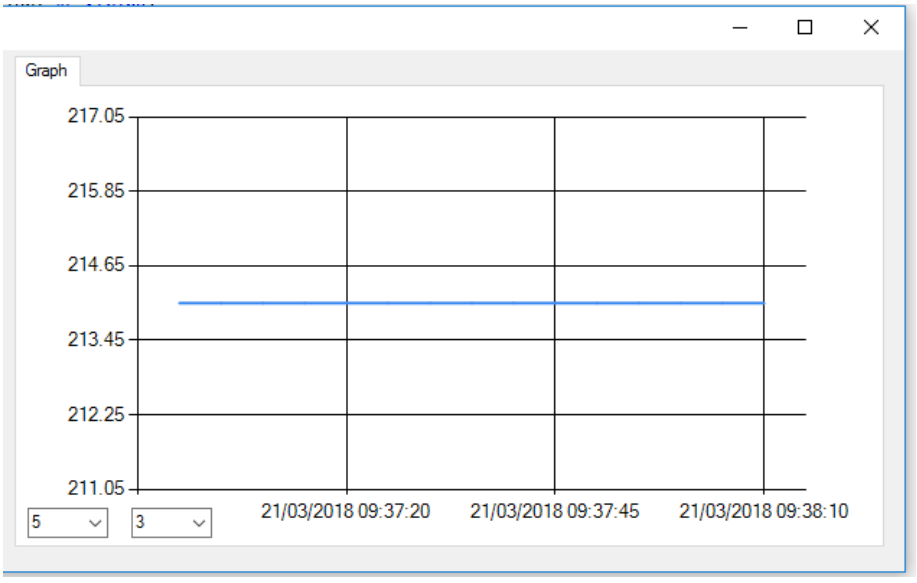
The above screenshot shows another portfolio. Instead of simply buying the maximum amount of Barclays stock possible with the balance, this portfolio represents a diversified portfolio, with some money remaining in the users balance.

During this test, when selecting the ADN.L symbol in an attempt to add shares to the portfolio, the program crashed and threw the following error:

	 The simulation was attempting to retrieve some information about the stock, however this resulted in a crash as the split of the information resulted in '#N/A' instead of a number as would be expected. As '#N/A' is a string and not a decimal, this crashed the program. This crash was caused because there existed no stock information for the ADN.L stock. This indicates that the company that corresponds to the ADN.L symbol is no longer indexed in the FTSE 100, and therefore it is not possible to retrieve information regarding that specific stock.	
Show the user is able to close open positions and that the simulation registers the sell action, updating the users balance and portfolio	 Selecting Barclays stock and buying the maxiumum amount results in the screensshot above. The user has one open position in their portfolio. The open position has been selected, as indicated by the tick in the box next to the position.	



Here, the ‘Close Position’ button has been clicked, and the position has been closed. The balance has returned close to £100,000, however not exactly. As the value was the same when the position was opened and closed (214.05 in both case), this value should be exactly £100,000.



The above graph shows the price during this transaction. As is visible, the price of the stock did not change over the running of the program. This means that the balance should have returned to £100,000. However this was not that case.

	This indicates that there exists a rounding error in the simualtion during the sell phase. This bug can be fixed by adding more accuracy to the stored decimals, and through avoiding truncation of stored number. This test has shown that although the feature works in the most part, there exists a bug that needs to be fixed in order for the simulation to have a smooth user experience.	

Testing 1 Findings – Investu Simulation - Development 1

From the test results above, we can see that the first development of the Investu simulation is working largely as expected, which indicates that the simulation has a good foundation to be built upon.

There does appear to be, however, two unexpected results in the last two tests;

1) The first unexpected result appears in test 9. When one of the stock symbols is selected from the drop-down list, the program crashes. After looking into this error, it appears to be because this company has fallen out of the FTSE 100 and is therefore their stock symbol is no longer supported by the GOOGLEFINANCE function in google sheets. This causes the string '#N/A' to be passed to the SplitStock function as he value for 'Change' which is expected to be a decimal value. Passing a string to a function where a decimal is expected creates an error.
2) The second unexpected result occurs when the user closes a position. Even when the user buys and sells at the same price, the end balance is not the same as the start balance, which indicates there is some sort of rounding error in the simulation. This could be because of a data type being stored as an integer instead of a decimal.

Fixing Errors - Investu Development 1

Error 1

In order to fix this error, a validation check needs to be added to make sure the value for ‘change’ is not ‘#N/A’. This was the code originally;

```
Function splitStockInfo(ByVal stockInfo As String, ByVal identifier1 As String)

        Dim ArrayList() As String = stockInfo.Split(":")
        Dim SubArrayList() As String = ArrayList(1).Split(",")
        Dim SubArrayList1() As String = ArrayList(2).Split(",")
        Dim StockChange As Decimal = 0
        Dim StockPrice As Decimal = 0
        Dim StockName As String = 0


        Select Case identifier1

            Case "name"

                StockName = SubArrayList(0)
                Return StockName

            Case "price"
                StockPrice = SubArrayList1(0)
                Return StockPrice

            Case "change"
                StockChange = ArrayList(3)
                Return StockChange

            Case Else
                Return "?"

        End Select
```

And the following is the new code;

```
Function SplitStockInfo(ByVal StringToSplit As String, ByVal DetailsToExtract As String)

        Dim ExtractedDetails As String = ""

        Dim ArrayList() As String = StringToSplit.Split(":")
        Dim SubArrayList() As String = ArrayList(1).Split(",")
        Dim SubArrayList1() As String = ArrayList(2).Split(",")

        Select Case DetailsToExtract

            Case "Name"
                If Trim(SubArrayList(0)) = "#N/A" Then
                    ExtractedDetails = "ERROR"
                Else
                    ExtractedDetails = Trim(SubArrayList(0))
                End If

            Case "Price"
                If Trim(SubArrayList1(0)) = "#N/A" Then
                    ExtractedDetails = "ERROR"
                Else
                    ExtractedDetails = Trim(SubArrayList1(0))
                End If

            Case "Change"
                If Trim(ArrayList(3)) = "#N/A" Then
                    ExtractedDetails = "ERROR"
                Else
                    ExtractedDetails = Trim(ArrayList(3))
                End If


        End Select

        Return ExtractedDetails
    End Function
```

The concept remains the same; the function uses a select-case to find and return the correct information, however this time there is a conditional on each case to check that the value is not '#N/A'. We know if there is an error, the value will always be '#N/A', so it is safe to hard wire this value into the condition. Also added is the use of the 'Trim' function, which removes any leading or trailing whitespace on the values. This is a precautionary addition just incase there happens to be whitespace that affects how data is displayed in future developments.

Error 2

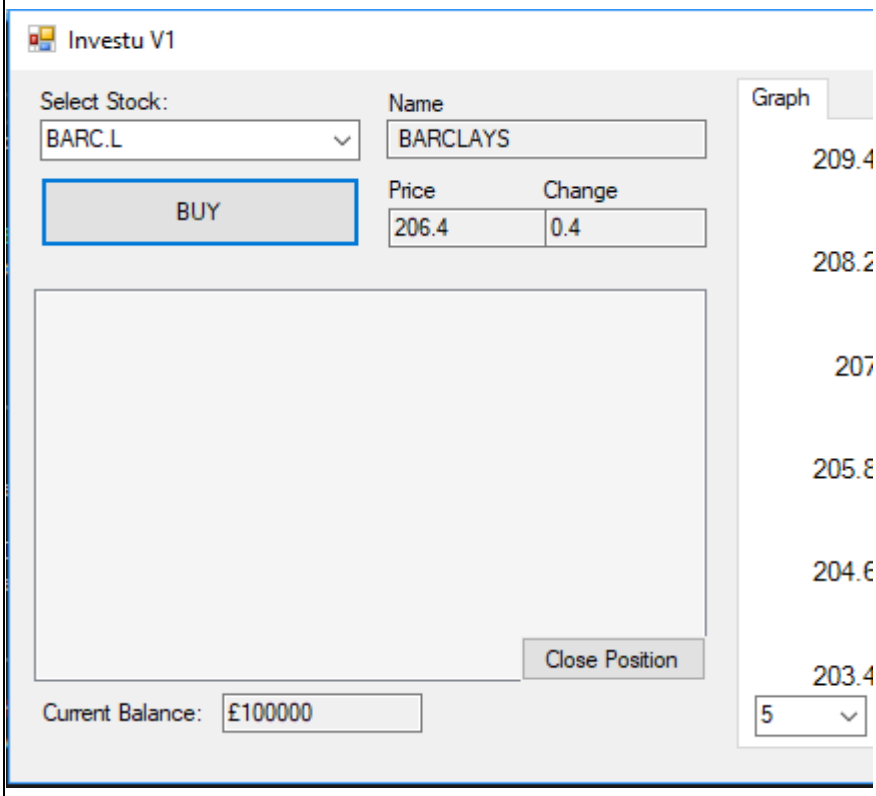
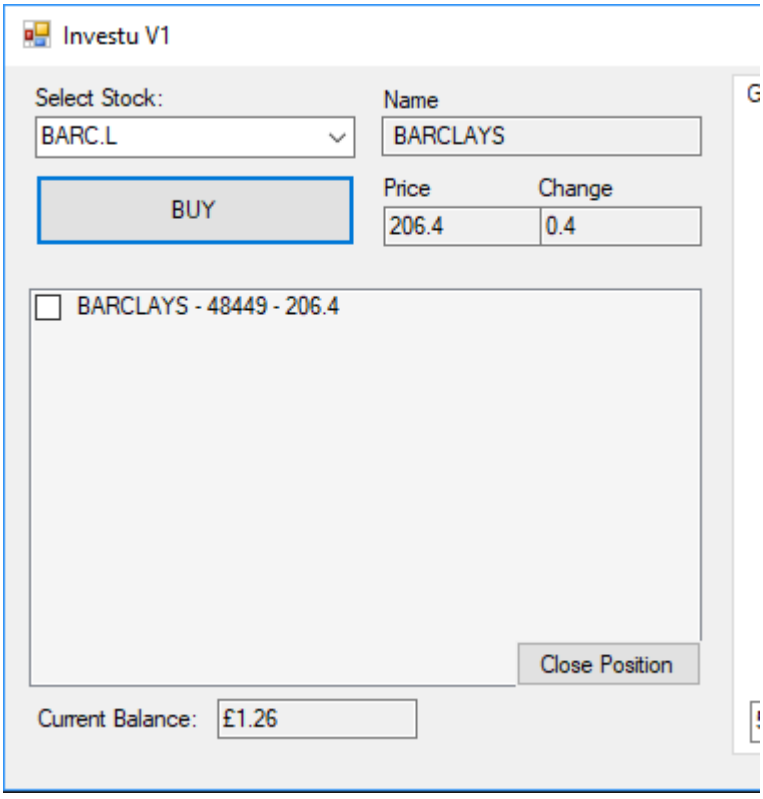
This error occured because of the ‘UpdateBalance’ sub-routine, which updates the users balance after a position is closed.

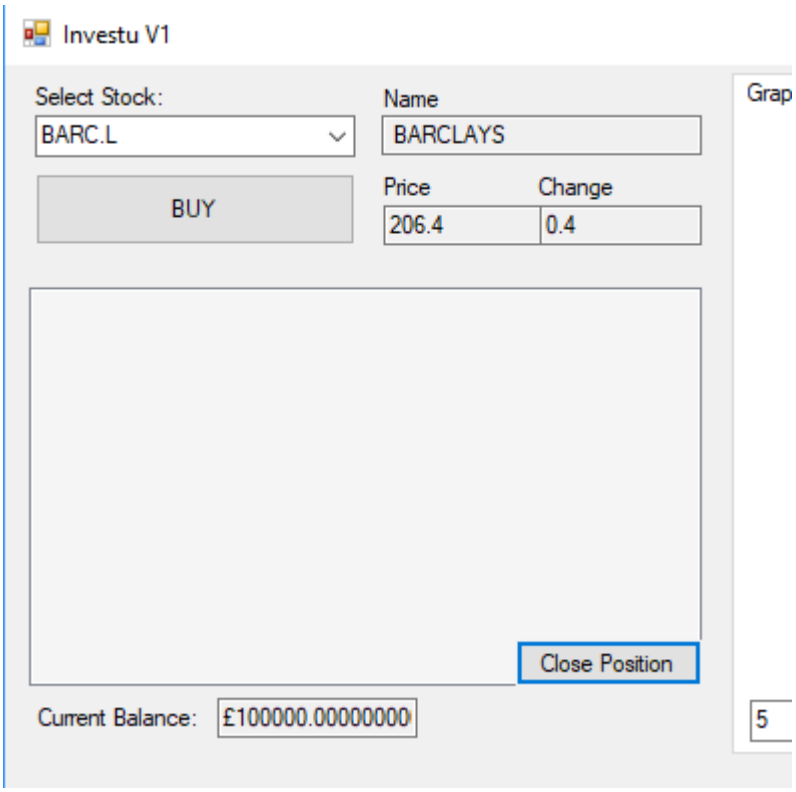
```
Sub UpdateBalance(ByVal Balance As Integer, ByVal Price As Integer, ByVal Quantity
As Integer)

    Balance = Balance + (Quantity * Price)
    BalanceBox.Text = "£" & Balance / 100

End Sub
```

By updating the UpdateBalance sub-routine to not accept a balance argument, and instead use the balance variable, and changing the data type of the price from integer to decimal, the simulation now accurately updates the balance to the correct value after a position is closed.

```
Sub UpdateBalance(ByVal Price As Decimal, ByVal Quantity As Integer)

        Balance = Balance + (Quantity * Price)
        BalanceBox.Text = "£" & Balance / 100

    End Sub
```

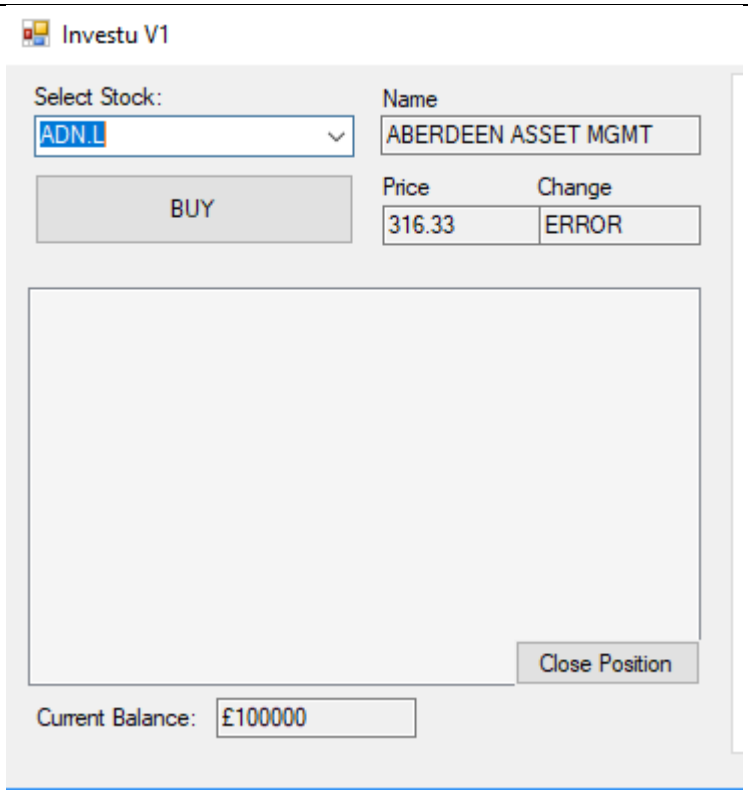
Testing 2 - Investu Development 1

Show the user is able to close open positions and that the simulation registers the sell action, updating the users balance and portfolio	 	



This sequence of three screenshots shows a successful buy and sell of a quantity of stock.

1) Before any trade is made, the balance is £100,000
2) After the maximum possible number of shares is bought, the user is left with £1.26
3) After the position is closed, the balance returns to exactly £100,000 (trailing 0's left to show accuracy) and the position is removed from the list of open positions.

Show the user's buy registers and is displayed in the portfolio box, and the balance is updated correctly. Show that no errors occur even when querying companies no longer in the FTSE 100.	 Previously, clicking on this option threw an error. After the changes to the code, the value of any value that is returned as '#N/A' will be set to 'ERROR', and the simulation will run as expected.	

Feedback #3 – Client – Investu Development 1

Development 1 provides the functionality for the simulation to perform at its most basic level. To ensure that the vision for this program is still on track and in line with that of the client and users, it is important to stay in communication. The following interview therefore took place to ensure that the final product of Development 1 is as expected and performing as intended, in the mind of the end user.

Ben, a student participating in the Student Investor Challenge, was given access to Development 1 in order to test the simulation. The following is a dialogue that took place afterwards. (dialogue editted for brevity)

"Ben – bearing in mind that this is an initial development of the simulation, what do you think of Investu so far?"

"After having a little look around I really like it. It's kind of similar to the SIC software but it feels a lot easier to use and looks a lot better. I really like the graph on the side, and the fact you can instantly fetch price information for each stock, with a single click. That makes it really clear."

"Have you tried trading on the simulation?"

"Yes – I've been buying and selling for a few minutes and it's been really interesting – the fact there are no fees yet means you can invest in a company and then watch the graph until the price changes and then sell it instantly – it's really fun anticipating whether or not it'll rise or fall. I haven't been able to profit much because the prices are only changing by a tiny bit eachchange, and I've only been using it a few minutes so there hasn't really been any noticeable price changes. To see any real returns I reckon you'd have to put your whole balance in and leave it a few minutes, or make an investment and leave it for hours or a few days. It's a shame that everything resets after closing it."

"What are you looking forward to in the future developments?"

"Well being able to keep progress would be great – I'm sure when thats possible it'll be really fun to try and keep a running progress over a few weeks. It's also going to be a lot easier to invest when theres some help choosing what to invest in, stuff like news, because at the moment it's a bit of a random guess. The graph is also good but it doesn't show much at the moment. It only starts showing anything from the time you load the program. It will be much clearer when it shows a bigger picture of the price of the stocks."

Final Conclusion – Investu Development 1

Development 1 has successfully fulfilled some of the basic criteria set out by the client, and is in a good position to be build upon in order to fulfil the remaining criteria required. Fundamentally, the simualtion now does what is needed, however without the features needed for the program to be a viable solution.

In the analysis of this simulation, a feature list and an objectives list was created using feedback from the client and the user. Now that development 1 has been implemented, we can see how many of this criteria have been met, and the goals for the second development of the program.

(The items highlighted in green have been successfully implemented into development 1, as shown in the development 1 testing phase in the previous section)

- Ability to create and login to accounts (client)
- Ability to join and trade on a team account (client)
- Ability for users to be designated as admins (client)
- Ability for account progress on team and personal accounts to be saved between sessions (inferred from client and user)
- Ability for admins to view teams list (inferred from client)
- Ability for admins to view team details and progress (inferred from client)
- Ability to view real-time information for all FTSE 100 stocks (client and user)
- Ability to select an amount of stocks and buy that amount using an up to date virtual balance, at real current price (client/user)
- Display for all stocks currently held in portfolio (client/SIC)
- Ability to sell stocks in portfolio at real current price (client)
- Graphs to display current day price trends of all stocks (user)
- Graph to show all time price changes of all stocks (inferred from user)
- Ability to create price alerts and be notified when stock reaches current price (inferred from user)
- Interface allowing users to see all current alerts on their account (inferred from user)
- Interface allowing user to see entire trade history (SIC)
- Interface allowing user to see all stocks in the FTSE 100 in a single screen, with details such as price (SIC)
-
- Notes section displayed in trade history and portfolio with reasons for trade decision (user)

In the next development of the program the aim is to successfully connect the database to the program, allowing for the storage of data related to the FTSE 100 and the user account. This will allow for a login system, as well as a teams system and the ability to save information relating to these two features. Furthermore, once the database is connected, it will be possible to begin collecting stock market data and storing it. This will require a small, additional program, that will operate on a server 24/7 in order to collect data.

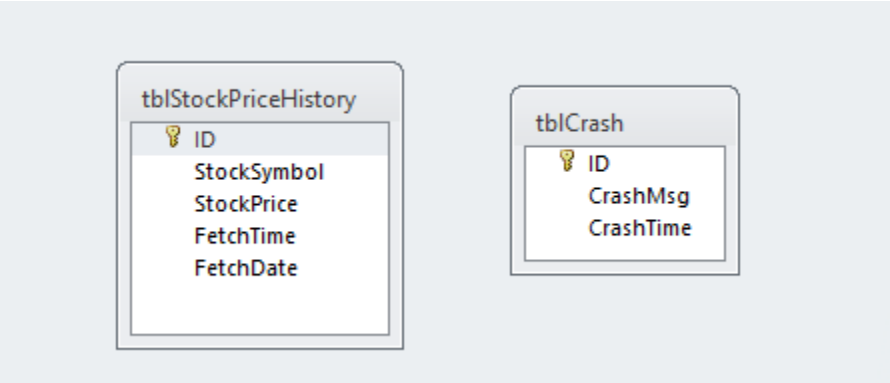
Development 2

Database – InvestuServerProgram - Development 2

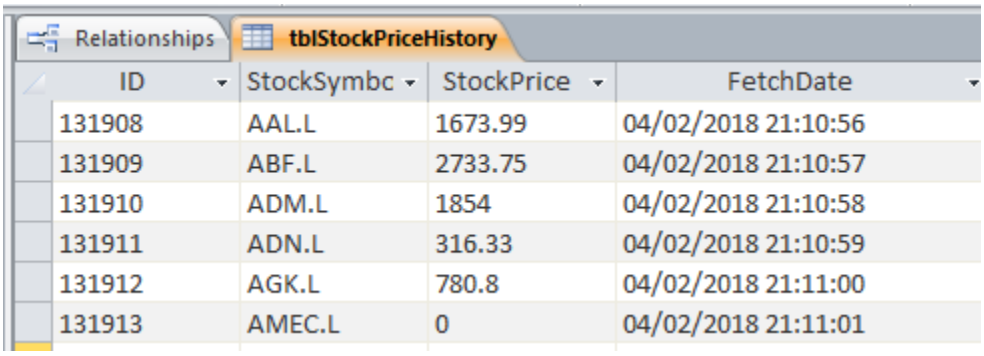
The server program will output data into a database. This database will be called ‘StockInfoDB’ and will be in the MDB file format.

The first table in the database will be called ‘tblStockPriceHistory’. This table will store all of the price history for each of the stock symbols in the FTSE100.

The following is a screenshot of the entity relationship diagram for the database. Currently there is only two tables, which are not linked, and so no relationships can be drawn.



Inside the main table, the format is as follows: (Some sample data has been written to the table)



ID	StockSymbc	StockPrice	FetchDate
131908	AAL.L	1673.99	04/02/2018 21:10:56
131909	ABF.L	2733.75	04/02/2018 21:10:57
131910	ADM.L	1854	04/02/2018 21:10:58
131911	ADN.L	316.33	04/02/2018 21:10:59
131912	AGK.L	780.8	04/02/2018 21:11:00
131913	AMEC.L	0	04/02/2018 21:11:01

Investu Server Program – Version 1 – Development 2

In addition to the main simulation, Development 2 brings about the need for an additional program. This program will be called InvestuServerProgram, and will run 24/7 on a server inside the school building. The purpose of this program is to collect stock market data for use in the simulation. By connecting the program to a database, information can be gathered and stored, which can then be used to extend information provided to the user in the main simulation, such as price history over the last X hours. Some other features proposed in the initial analysis will also require this server program; the alerts system for example, which will alert the users to when a stock reaches a certain price, even when they are logged out of the simulation. This feature will come later in the development of the simulation.

Imports/Namespaces – InvestuServerProgram - Development 2

```
Imports System.IO
Imports System.Xml
Imports System.Data.OleDb
```

'Imports' here allows for types that are contained in a given namespace to be referenced directly. 3 namespaces are utilized in InvestuServerProgram:

- System.IO – This namespace handles the manipulation of files
- System.XML – Handles the manipulation and processing of XML data
- System.Data.OleDB – OLE DB stands for Object Linking and Embedding Database, which is an API allowing the access of data from various sources, in this case Microsoft Access. Importing this namespace allows us to easily manipulate a database using SQL.

Global Variables – InvestuServerProgram - Development 2

```
Public Class MainForm

    Dim DBPath As String = "C:\Users\Joe\Documents\visual studio 2010\Investu
Writeup\Development 2\StockInfoDB.mdb"
    Public AccessDatabaseConnection As String = "Provider =
Microsoft.Jet.OLEDB.4.0;Data Source =" & DBPath

    Dim LoopCount As Integer = 0
    Public Symbols As New List(Of String)
```

InvestuServerPorgram has 3 global variables in this development.

```
    Dim DBPath As String = "C:\Users\Joe\Documents\visual studio 2010\Investu
Writeup\Development 2\StockInfoDB.mdb"
```

Firstly, we the information for connecting to the database is defined. ‘DBPath’ here is simply the file location of the database. In this case, the database is called ‘StockInfoDB.mdb’. ‘mdb’ stands for Microsoft Access Database and is the standard file type when use Microsoft Access.

```
Public AccessDatabaseConnection As String = "Provider = Microsoft.Jet.OLEDB.4.0;Data
Source =" & DBPath
```

Secondly, we define ‘AccessDatabaseConnection’ which is contains the configuration for connecting to the database. The syntax for this consists of ‘Provider=””Data Source=””’. ‘Provider’ in this case is `Microsoft.Jet.OLEDB.4.0` which is the standard format for the file type being used. ‘Data Source’ is simply the file pathway of the database, which was defined in the previous line as ‘DBPath’. The concatenation of these two strings results in a variable that can be called at any time to initiate a connection to the database.

```
Dim LoopCount As Integer = 0
```

‘LoopCount’ will be used to keep track of how many times the program has looped through a query, so that it is easy to keep track of when to perform a reset. In this case, the program will have to reset once every stock symbol has been queried. LoopCount has to be made a global variable because it will be used alongside a timer. If it was declared inside the [timer.tick’ sub-routine, then it would be effectively redefined and lose its value every tick. For this reason it is defined globally and simply referred back to in the timer.tick sub-routine.

```
Public Symbols As New List(Of String)
```

In this version of the program, the stock symbols for the companies in the FTSE 100 will be fetched from a .CSV file, instead of being hard-coded directly into the program. To prepare for this, a list object is created, that can have items appended to the end.

MainForm_Load – InvestuServerProgram - Development 2

```
Private Sub MainForm_Load(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles MyBase.Load

        RunningStoppedLabel.Text = "STOPPED"
        RunningStoppedLabel.ForeColor = Color.Red

        PopulateSymbolList("C:\Users\Joe\Documents\visual studio 2010\Investu
Writeup\Development 2\StockSymbols.csv")
    End Sub
```

When MainForm of InvestuServerProgram is loaded, some formatting occurs that visually displays the current state of the program – which is in this case stopped, and the symbols list declared earlier is populated with symbols. This is done by passing the file location of the .CSV file containing the symbols, to the sub-routine 'PopulateSymbolList'.

PopulateSymbolArray – InvestuServerProgram - Development 2

```
Public Sub PopulateSymbolList(ByVal FilePath As String)

        Dim CSVData() As String

        Using SR As New StreamReader(FilePath)
            While Not SR.EndOfStream
                CSVData = SR.ReadLine().Split(",")
                If String.IsNullOrEmpty(CSVData(0)) Then
                    MsgBox("Error Null Value")
                Else
                    Symbols.Add(CSVData(0).Trim)
                End If
            End While
        End Using

    End Sub
```

PopulateSymbolList is used to extract all of the symbols stored in StockSymbols.CSV, and use them to populate a list.

```
Using SR As New StreamReader(FilePath)
  ...
End Using
```

StreamReader is a text reader in the System.IO namespace, which reads characters from a byte stream. In this code, a 'Using' statement declares 'SR' as a StreamReader, and passes the value of FilePath to the stream reader. By calling StreamReader here, a new instance is initialized, with the stream specified as FilePath.

```
While Not SR.EndOfStream
    ...
End While
```

‘EndOfStream’ is a built in property of StreamReader that gives a boolean value indicating whether or not the current position in the stream is the end of the stream or not. This While loop will continue to loop while EndOfStream is false.

```
CSVData = SR.ReadLine().Split(",")
```

Inside the While loop, a value is assigned to CSVData, a variable defined earlier in the sub-routine. The value is equal to the value on the current line, split by a comma. Each value has been written to the .CSV file seperated by a comma, and so this line of code splits these values up, and assigns the value of the split symbol into CSVData.

```
If String.IsNullOrEmpty(CSVData(0)) Then
    MsgBox("Error Null Value")
Else
    Symbols.Add(CSVData.Trim)
End If
```

The next section of code is a conditional statement, which is used as a validation check to prevent erros later in the code. The condition checks to see if the value of CSVData is null or empty. This could be caused by incorrect values in the .CSV file, but they are avoided through this conditional. Assuming the value of CSVData is not null or empty, the value is trimmed to remove any leading or trailing whitespace, and then added to the Symbols list.

StartButton_Click – InvestuServerProgram - Development 2

```
Private Sub StartButton_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles StartButton.Click

        Timer1.Start()

        RunningStoppedLabel.Text = "RUNNING"
        RunningStoppedLabel.ForeColor = Color.Green

    End Sub
```

This sub-routine starts the timer about which this program revolves. Each tick of the timer will cause new information to be fetched from the internet and stored in the database. The visual display of the program is updated so that the user can see that the programs current state is running.

Timer1_Tick – InvestuServerProgram - Development 2

```
    Private Sub Timer1_Tick(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Timer1.Tick

        If LoopCount > 99 Then
            LoopCount = 0
            RecentPricesBox.Clear()

        End If

        FetchLatestStockInfo()
        LoopCount += 1

    End Sub
```

The tick of this timer is integeral to the operation of the server program. The basic premise is that for every tick of the timer, the information for the symbol whos value is stored at the index value LoopCount in SybolArray, will be fetched, and stored to the database. LoopCount is incremented, so that the next tick, a new symbol is queried, and so on until all of the symbols have been queried, at which point the value of LoopCount resets to 0, and the symbols are queried again from the beginning.

The timer interval is set to one second. One query per second is managable for the program, and will mean that there is little threat of a stack overflow error occuring. If the value of the timer interval was any lower than one second, there is a chance that before one query has finished, the next query would be called, which would eventually lead to a stack overflow.

FetchLatestStockInfo – InvestuServerProgram - Development 2

```
Sub FetchLatestStockInfo()

        Timer1.Interval = 1000
        Dim StockPrice As Decimal = GetStockPrice(Symbols(LoopCount))
        Dim StockChange As Decimal = GetStockChange(Symbols(LoopCount))

        If TimeOfDay.Hour > 8 And TimeOfDay.Hour < 16.5 Then
            Try

                UpdateDatabase(Symbols(LoopCount), StockPrice, StockChange)
                RecentPricesBox.Text += FormatString(LoopCount, Symbols(LoopCount),
StockPrice, StockChange)

            Catch errorVariable As Exception

                StoreCrashInfo(errorVariable.ToString(), DateTime.Now)
                If ShowErrorCheckBox.Checked = True Then
                    Timer1.Stop()
                    MsgBox(errorVariable.ToString())
                    RunningStoppedLabel.Text = "STOPPED"
                End If
            End Try
        End If

    End Sub
```

This sub-routine is called every time the timer ticks. It fetches the information relating to the stock currently stored at the index value of **LoopCount in the Symbols array.**

```
Timer1.Interval = 1000
```

Initially, the interval of the timer is reset back to 1 second. This is important so that if the interval for any reason changes, it is reset as quick as possible so that the interval between collecting data from each symbol is consistent.

```
Dim StockPrice As Decimal
Dim StockChange As Decimal
StockPrice = GetStockPrice(Symbols(LoopCount))
StockChange= GetStockChange(Symbols(LoopCount))
```

Next, StockPrice and StockChange are declared and set to their current value, using the GetStockPrice and GetStockChange functions. These functions both accept one argument, which in both cases is ‘Symbols(LoopCount). This means that the symbol whose information is retrieved is the symbol whos index value in the Symbols list is ‘LoopCount’.

```
If TimeOfDay.Hour > 8 And TimeOfDay.Hour < 16.5 Then
   ...
End If
```

This conditional uses TimeOfDay.Hour, a built in VB value, to check the time of day. If the time is after 8:00am and 16:30pm, then the code in the conditional will execute. If the current time is not within those times, then the code will not execute. This check is here because these times are the FTSE100 trading hours. Outside of these hours, the prices will never change as it is not possible to trade on any stock in the FTSE100. By adding this conditional, we can avoid wasteful processing on the server and save memory in the database by not adding redundant data.

```
Try
    ...

Catch ErrorVariable As Exception

    StoreCrashInfo(ErrorVariable.ToString(), DateTime.Now)
    If ShowErrorCheckBox.Checked = True Then
         Timer1.Stop()
         MsgBox(ErrorVariable.ToString())
         RunningStoppedLabel.Text = "STOPPED"
    End If
End Try
```

This Try-Catch is used to avoid errors that arise during the process of fetching and storing stock information in the InvestuServerProgram.

Firstly, the program attempts to store the error information into the database, using a sub-routine called StoreCrashInfo, which accepts two arguments; the error variable and the current time. This information is useful for debugging purposes as this program will run for days and weeks without human interaction, and so it can be difficult to know when or why the program crashed, if an error does occur. By storing the error and time debugging is made easier.

Next, a conditional checks if a checkbox on the user interface is checked. If this checkbox is ticked, the program will stop, because timer1.stop is called. Then, a message box will appear showing the error variable, and the interface will be updated to show that program is in a stopped state. If the checkbox is not ticked, then the error variable is stored into the database but none of the other code executes. It is useful to have these two modes, having the program stop if an error occurs can be useful for debugging, but would not be useful when the simulation is in use by users, as it would mean their simulations would stop receiving data.

```
UpdateDatabase(Symbols(LoopCount), StockPrice, StockChange)
RecentPricesBox.Text += FormatString(LoopCount, Symbols(LoopCount), StockPrice,
StockChange)
```

Inside the Try-Catch are two calls to sub-routines. The first of these calls is a call to UpdateDatabase. UpdateDatabase accepts 3 arguments; A symbol, about which information is to be stored, a price value, and a change value. This then adds the relevant information to the database.

The next call is to a sub-routine called FormatString, which simply takes information and formats it. This newly formatted string is added to a text box which works as a visual display for the user to see how the server program is operating.

FormatString – InvestuServerProgram - Development 2

```
Function FormatString(ByVal A As Integer, B As String, C As String, D As String)
        Dim FormattedString As String

        B = B & Space(10 - B.Length)
        C = C & Space(10 - C.Length)
        D = D & Space(10 - D.Length)

        FormattedString = A & Space(5) & B & C & D & vbCrLf
        Return FormattedString
    End Function
```

FormatString simply takes the values passed and spaces them equally, to create a clear visual display for the user. The function takes and integer and two strings, and then formats them in such a way that the start of each piece of information.

UpdateDatabase – InvestuServerProgram - Development 2

```
Sub UpdateDatabase(ByVal StockSymbol As String, ByVal StockPrice As Decimal, ByVal
StockChange As Decimal)

        Dim ConnectionDb As New OleDbConnection(AccessDatabaseConnection)
        If ConnectionDb.State = ConnectionState.Closed Then ConnectionDb.Open()

        Dim cmd As OleDbCommand = ConnectionDb.CreateCommand

        cmd.CommandText = "INSERT INTO tblStockPriceHistory (StockSymbol, StockPrice,
FetchTime, FetchDate) VALUES ('" & StockSymbol & "','" & StockPrice & "','" &
TimeOfDay & "','" & Date.Now & "')"
        cmd.ExecuteNonQuery()


        ConnectionDb.Close()

    End Sub
```

UpdateDatabase takes three arguments and uses them to create a new entry into the database.

```
Dim ConnectionDb As New OleDbConnection(AccessDatabaseConnection

If ConnectionDb.State = ConnectionState.Closed Then ConnectionDb.Open()
   ...
ConnectionDb.Close()
```

This code declares as variable, connectionDB, as a new connection the database. ‘OleDbConnection’ is a type built into the OleDb namespace that was imported earlier. It takes one value, which is the connection string. This was defined earlier in the code. Once this connection is created, its default state is closed. Therefore, the next line of code checks the state, and if it is found to be closed, the connection is opened, meaning the database is now accessible to edit. The final line of code makes sure that the connection is closed after the database has been editted. This is important, as trying to access the database multiple times without closing connections first can result in concurrency issues and crashes.

```
Dim cmd As OleDbCommand = ConnectionDb.CreateCommand
```

Once a connection has been established, a new command can be created. This is done via the ‘ConnectionDb.CreateCommand’ command. Once the command is created, it needs text that will be executed. This is done via ‘cmd.CommandText =’. In this case, data is being written to the database, and so an insert command is used.

```
        cmd.CommandText = "INSERT INTO tblStockPriceHistory (StockSymbol, StockPrice,
FetchDate) VALUES ('" & StockSymbol & "','" & StockPrice & "','" & Date.Now & "')"
        cmd.ExecuteNonQuery()
```

The sytax of an ‘Insert’ command is as follows;
INSERT INTO tableName (columnName) VALUES (value)

Therefore in this statment, we are inserting into the table ‘tblStockPriceHistory’, specifically the three columns StockSymbol, StockPrice and FetchTime. The value inserted are StockSymbol, StockPrice and the current date.

SplitStockInfo – InvestuServerProgram - Development 2

```
Function SplitStockInfo(ByVal StringToSplit As String, ByVal DetailsToExtract As
String)

        Dim ExtractedDetails As String = ""

        Dim ArrayList() As String = StringToSplit.Split(":")
        Dim SubArrayList() As String = ArrayList(1).Split(",")
        Dim SubArrayList1() As String = ArrayList(2).Split(",")

        Select Case DetailsToExtract

            Case "Name"
                ExtractedDetails = Trim(SubArrayList(0))

            Case "Price"
                ExtractedDetails = Trim(SubArrayList1(0))
                If ExtractedDetails = "#N/A" Then
                    ExtractedDetails = "0"
                End If

            Case "Change"
                ExtractedDetails = Trim(ArrayList(3))
                If ExtractedDetails = "#N/A" Then
                    ExtractedDetails = "0"
                End If

        End Select
        Return ExtractedDetails
    End Function
```

The code here is taken from Testing 2 in Development 2 on page 96.

GetStockChange – MainForm - InvestuServerProgram - Development 2

```
    Function GetStockChange(ByVal StockSymbol As String)

        Dim StockChange As Decimal
        Try

            Dim document As XmlDocument
            Dim nodelist As XmlNodeList
            Dim node As XmlNode

            document = New XmlDocument()

document.Load("https://spreadsheets.google.com/feeds/list/0AhySzEddwIC1dEtpWF9hQUhCWUR
ZNEViUmpUeVgwdGc/1/public/basic?sq=symbol=" & StockSymbol)
            nodelist = document.GetElementsByTagName("entry")

            For Each node In nodelist
                StockChange = SplitStockInfo(node.ChildNodes.Item(4).InnerText,
"Change")
            Next
        Catch errorVariable As Exception
            Timer1.Stop()
        End Try
        Return StockChange
    End Function
```

'GetStockChange' is a function that has a single parameter – 'StockSymbol', which the function uses to retrieve the 'Change' value for the relevant stock. This is done via querying the Google Sheets sheet with the symbol link appended to the end, and then splitting the resulting string for the 'Change' value.

This process of retrieving stock information from the internet using XML is detailed between pages 27 and 31.

The origin of this sub-routine including a detailed code explaination can be found in Development 1 on page 64.

GetStockName – MainForm - InvestuServerProgram - Development 2

```
Function GetStockName(ByVal StockSymbol As String)

        Dim StockName As String = "Error"
        Try

            Dim document As XmlDocument
            Dim nodelist As XmlNodeList
            Dim node As XmlNode

            document = New XmlDocument()

document.Load("https://spreadsheets.google.com/feeds/list/0AhySzEddwIC1dEtpWF9hQUhCWUR
ZNEViUmpUeVgwdGc/1/public/basic?sq=symbol=" & StockSymbol)
            nodelist = document.GetElementsByTagName("entry")

            For Each node In nodelist
                StockName = SplitStockInfo(node.ChildNodes.Item(4).InnerText, "Name")

            Next
        Catch errorVariable As Exception
            Timer1.Stop
        End Try
        Return StockName
    End Function
```

'GetStockName' is a function that has a single parameter – 'StockSymbol', which the function uses to retrieve the 'Change' value for the relevant stock. This is done via querying the Google Sheets sheet with the symbol link appended to the end, and then splitting the resulting string for the 'Change' value.

This process of retrieving stock information from the internet using XML is detailed between pages 27 and 31.

The origin of this sub-routine including a detailed code explaination can be found in Development 1 on page 64.

GetStockPrice – MainForm - InvestuServerProgram - Development 2

```
Function GetStockPrice(ByVal StockSymbol As String)

        Dim StockPrice As Decimal
        Try
            Dim document As XmlDocument
            Dim nodelist As XmlNodeList
            Dim node As XmlNode
            document = New XmlDocument()

document.Load("https://spreadsheets.google.com/feeds/list/0AhySzEddwIC1dEtpWF9hQUhCWUR
ZNEViUmpUeVgwdGc/1/public/basic?sq=symbol=" & StockSymbol)
            nodelist = document.GetElementsByTagName("entry")

            For Each node In nodelist
                StockPrice = SplitStockInfo(node.ChildNodes.Item(4).InnerText,
"Price")
            Next
        Catch errorVariable As Exception
            Timer1.Stop()

        End Try
        Return StockPrice

    End Function
```

'GetStockPrice' is a function that has a single parameter – 'StockSymbol', which the function uses to retrieve the 'Change' value for the relevant stock. This is done via querying the Google Sheets sheet with the symbol link appended to the end, and then splitting the resulting string for the 'Change' value.

This process of retrieving stock information from the internet using XML is detailed between pages 27 and 31.

The origin of this sub-routine including a detailed code explaination can be found in Development 1 on page 64.

StoreCrashInfo – MainForm - InvestuServerProgram - Development 2

```
Sub StoreCrashInfo(ByVal CrashMsg, ByVal CrashTime)

        Dim ConnectionDb As New OleDbConnection(AccessDatabaseConnection)
        If ConnectionDb.State = ConnectionState.Closed Then ConnectionDb.Open()

        Dim cmd As OleDbCommand = ConnectionDb.CreateCommand


        cmd.CommandText = "INSERT INTO TblCrash (CrashMsg, CrashTime) VALUES ('" &
CrashMsg & "','" & CrashTime & "')"

        cmd.ExecuteNonQuery()
        ConnectionDb.Close()
    End Sub

End Class
```

Store crash info is a sub-routine used to store in the information relating to crashes in 'InvestuServerProgram'. This sub-routine will be useful for debugging purposes, as the resulting database entries can be looked at for information relating to crashes, which can then help to fix bugs.

```
Dim ConnectionDb As New OleDbConnection(AccessDatabaseConnection)
If ConnectionDb.State = ConnectionState.Closed Then ConnectionDb.Open()

Dim cmd As OleDbCommand = ConnectionDb.CreateCommand
```

The sub-routine begins by connecting to the database. A new command is created via the 'CreateCommand' method.

```
cmd.CommandText = "INSERT INTO TblCrash (CrashMsg, CrashTime) VALUES ('" & CrashMsg &
"','" & CrashTime & "')"
```

The command text is an Insert command, that inserts two values into the table 'tblCrash'. These values are the exeception that was thrown, or 'CrashMsg', and the current time, or 'CrashTime'.

```
cmd.ExecuteNonQuery()
ConnectionDb.Close()
```

The command is executed and the connection to the database is closed.

Investu Simulation – Development 2

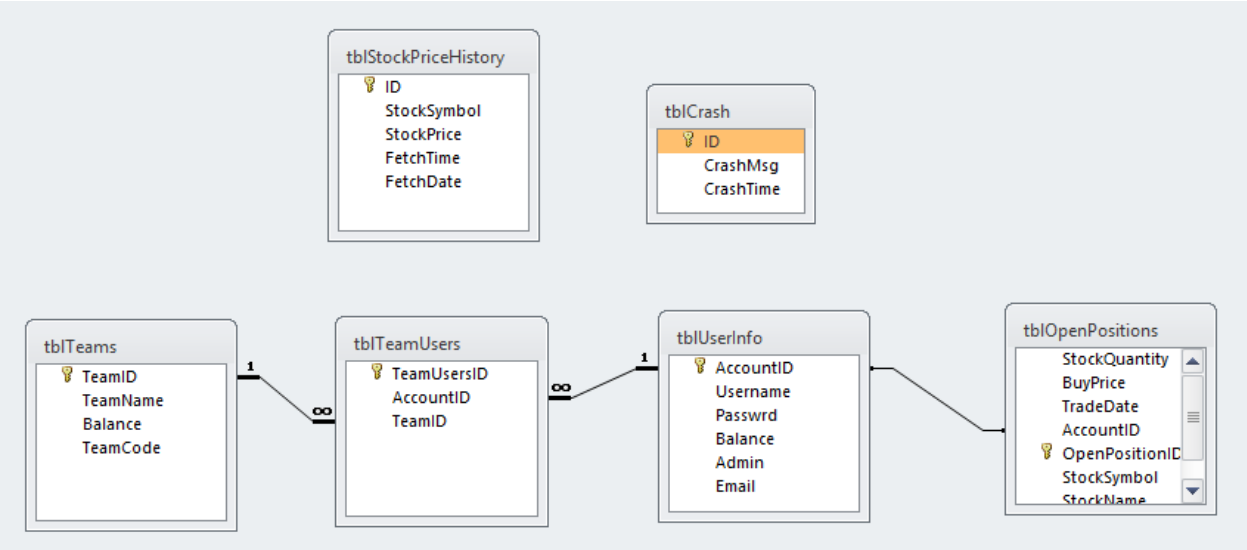
This development of the simulation will include the ability for users to log into their personal and team accounts. This means that Development 2 requires a form that allows users to create an account, and a form that allows users to login to their account. These will be call SignUpForm and LoginForm respectively.

This development of the simulation will also incorporate the team system which works as follows:

Teams can be created; which users can join. When a user is part of a team, they will have the choice upon logging in, to either load into the team account – an account accessible by themselves and the other 3 members of their team, or to load into their personal account. This choice will be determined by a checkbox in LoginForm. When a user logs into a team account, their progress will be saved to that account rather than their personal account.

Database 2 – Investu – Development 2

With the addition of SignUpForm and LoginForm, comes the need for additional tables in the database.



The additional tables are as follows:

- tblUserInfo – A table for storing the data relating to individual users in the simulation. Every new user has their own entry in the table, with data such as name, password and balanced stored.
- tblTeams – A table for storing the data relating to the teams in the simulation.
- tblTeamUsers – A link table for linking users to teams and vice versa. This tracks which users are members of each team.
- tblOpenPositions – Used for storing the currently open positions that users have. The position has an attribute that states the accountID that executed the trade, which will allow the simulation to load open positions that were made in the past.

SignUpForm – Investu - Development 2

SignUpForm untilises the database to create new accounts, that will then be accessible later on. These accounts will allow users to save their information such as balance and portfolios, meaning they can keep their progress even after logging out of the simulation. This will help to improve the user experience of the simulation, and the effectiveness of the program in reaching its desired goal.

Global Variables – SignUpForm – Investu Development 2

```
Public Class SignUpForm

    Dim AccessDatabaseConnection As String = MainForm.AccessDatabaseConnection
```

In this form, the is only one global variable – the database connection string. The value of this string is fetched from MainForm. If the location of the database changes, then by fetching the connection string from MainForm, it means that only the string in MainForm needs to be changed.

SignUpButton_Click – SignUpForm - Investu Development 2

```
    Private Sub SignUpButton_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles CreateAccountButton.Click


        If ProceedToSignUp() Then
            CreateNewAccount(UsernameBox.Text, PasswordBox.Text, EmailBox.Text)

            LoginForm.UsernameTextBox.Text = UsernameBox.Text
            LoginForm.PasswordTextBox.Text = PasswordBox.Text

            MsgBox("Your account has been created! Click login to proceed.")

            Me.Close()
        End If
```

This sub-routine occurs when the user clicks the ‘Sign Up’ button. The user interface will have a series of input boxes, followed by a ‘Sign Up’ button. Upon clicking this button, this code is executed.

```
If ProceedToSignUp() Then
   ....
End If
```

First, the conditional calls the function ‘ProceedToSignUp’ to check that the user credentials entered are valid. This validation check is made to ensure that a user does not try to create an account that already exists.

Once the condition of the conditional has been met, the sub-routine ‘CreateNewAccount’ is called. This is the sub-routine that is responsible for inputting the users new account information into the database.

```
CreateNewAccount(UsernameBox.Text, PasswordBox.Text, EmailBox.Text)

LoginForm.UsernameTextBox.Text = UsernameBox.Text
LoginForm.PasswordTextBox.Text = PasswordBox.Text

MsgBox("Your account has been created! Click login to proceed.")

Me.Close()
```

The following 2 lines of code affect LoginForm. They change the values of the username and password boxes to contain the information of the account just created. This is a quality-of-life addition to the code that simply makes the sign-up and login experience easier for the user. A message box appears that informs the user that they have successfully created an account, and then the form is closed, displaying the LoginForm behind.

ProceedToSignUp – SignUpForm - Investu Development 2

```
    Function ProceedToSignUp()

        If ValidatePassword(PasswordBox.Text) Then
            If ValidUsername(UsernameBox.Text) Then
                If ValidEmail(EmailBox.Text) Then
                    Return True
                Else
                    MsgBox("The email you have entered is invalid")
                End If
            Else
                MsgBox("The username you have entered is already taken.")
            End If
        Else
            MsgBox("Invalid Password - Passwords must have at least 1 upper case
character, 1 number and 8 total characters.")
        End If

        Return False
    End Function
```

The purpose of this function is to validate the inputs of the user. The function consists of three nested conditionals, each one checking if one aspect of the user information is valid; username, password and email. The reason there are three conditional instead of one with the conditions joined together by ‘And’, is to make it easy to differentiate between errors, so that the user can be advised which section of their sign up process did not pass validation.

ValidUsername – SignUpForm - Investu Development 2

```
Function ValidUsername(ByVal NewUsername As String)

        If UsernameBox.Text = "" Or PasswordBox.Text = "" Then
            ErrorMsg = "The Username and Password are required fields."
            Return False
        Else

            Dim ConnectionDb As New OleDbConnection(AccessDatabaseConnection)
            If ConnectionDb.State = ConnectionState.Closed Then ConnectionDb.Open()

            Dim cmd As OleDbCommand
            Dim SQLReply As OleDbDataReader

            cmd = ConnectionDb.CreateCommand
            cmd.CommandText = "SELECT Username FROM tblUserInfo"
            SQLReply = cmd.ExecuteReader

            For Each Record In SQLReply

                If Record.item("Username") = NewUsername Then
                    Return False
                End If
            Next

            ConnectionDb.Close()

        End If
        Return True
    End Function
```

```
If UsernameBox.Text = "" Or PasswordBox.Text = "" Then
      ErrorMsg = "The Username and Password are required fields."
      Return False
Else
```

The first validation check inside CheckUsername checks that neither the Username box or Password box are empty.

Then, a select query is set up. This constitutes first connecting to the database, and then creating a new command in SQL, which is this case is:

```
SELECT Username FROM tblUserInfo
```

```
Dim ConnectionDb As New OleDbConnection(AccessDatabaseConnection)
            If ConnectionDb.State = ConnectionState.Closed Then ConnectionDb.Open()

            Dim cmd As OleDbCommand
            Dim SQLReply As OleDbDataReader

            cmd = ConnectionDb.CreateCommand
            cmd.CommandText = "SELECT Username FROM tblUserInfo"
            SQLReply = cmd.ExecuteReader
```

This SQL command will select every username from the table ‘tblUserInfo’. Now that all of the usernames have been fetched, a linear search can be carried out in order to find whether or not the username already exists.

```
For Each Record In SQLReply
       If Record.item("Username") = NewUsername Then
                Return False
       End If
Next
ConnectionDb.Close()
```

This For-loop takes every username returned from the database after the query, and compares it to ‘NewUserName’, a value passed to the function, that contains the username the user is trying to sign up with. If a successful comparison between a fetched username and the new username is made, then false is returned. The connection to the database is then closed.

After this check, if the function has still not exited by returning false, it means the value of NewUsername is valid and the function returns true.

ValidatePassword - SignUpForm – Investu Development 2

```
Function ValidatePassword(ByVal Password As String, Optional ByVal MinLength As
Integer = 8, Optional ByVal NumUpper As Integer = 1, Optional ByVal NumLower As
Integer = 1, Optional ByVal NumNumbers As Integer = 1, Optional ByVal NumSpecial As
Integer = 0) As Boolean

        Dim UpperCase As New System.Text.RegularExpressions.Regex("\p{Lu}")
        Dim LowerCase As New System.Text.RegularExpressions.Regex("[a-z]")
        Dim Numbers As New System.Text.RegularExpressions.Regex("[0-9]")

        Dim Specials As New System.Text.RegularExpressions.Regex("[^a-zA-Z0-9]")

        If Len(Password) < MinLength Then Return False

        If UpperCase.Matches(Password).Count < NumUpper Then Return False
        If LowerCase.Matches(Password).Count < NumLower Then Return False
        If Numbers.Matches(Password).Count < NumNumbers Then Return False
        If Specials.Matches(Password).Count < NumSpecial Then Return Fals

        Return True
End Function
```

```
Function ValidatePassword(ByVal Password As String, Optional ByVal MinLength As Integer =
8, Optional ByVal NumUpper As Integer = 1, Optional ByVal NumLower As Integer = 1,
Optional ByVal NumNumbers As Integer = 1, Optional ByVal NumSpecial As Integer = 0) As
Boolean
```

This function has 6 parameters that allow for the customization of the requirements of the users password. The parameters are Password, which contains the users chosen password; MinLength, which determines how long the chosen password must be; NumUpper, which determines the number of upper case letters that the password must have; NumLower which determines the number of lower case letters that the password must have, NumNumbers which determines the number of numbers that the password must have and NumSpecial which determines the number of special characters that the password must have.

Firstly, these values are declared. They corrospond to the parameters of the function. These values are declared as RegularExpressions. RegularExpressions are a way of defining the syntax of strings. At the end of the declaration, in the brackets, the syntax of the string is written. Each section defines certain values, for example `[a-z]` means all values between a-z, and `[^a-zA-Z0-9]` means ‘all values not included in the sets ‘a-z, A-Z, 0-9’; therefore this defines the syntax for special characters, which do not fit into those categories.

```
Dim UpperCase As New System.Text.RegularExpressions.Regex("\p{Lu}")
Dim LowerCase As New System.Text.RegularExpressions.Regex("[a-z]")
Dim Numbers As New System.Text.RegularExpressions.Regex("[0-9]")
Dim Specials As New System.Text.RegularExpressions.Regex("[^a-zA-Z0-9]")
```

CreateNewAccount – SignUpForm – Investu Development 2

```
Sub CreateNewAccount(ByVal Username As String, ByVal Password As String, ByVal Email
As String)

        Dim Balance As Integer = 10000000

        Dim ConnectionDb As New OleDbConnection(AccessDatabaseConnection)
        If ConnectionDb.State = ConnectionState.Closed Then ConnectionDb.Open()

        Dim cmd As OleDbCommand
        cmd = ConnectionDb.CreateCommand

        cmd.CommandText = "INSERT INTO tblUserInfo (Username, Balance, Passwrd) VALUES
('" & Username & "','" & Balance & "','" & Password & "')"

        cmd.ExecuteNonQuery()

        ConnectionDb.Close()

    End Sub
```

CreatenewAccount is responsible for inserting the new, validated account information into the database. It does this through an 'INSERT' SQL command, into which the values of the relevant attributes are inserted.

```
cmd.CommandText = "INSERT INTO tblUserInfo (Username, Balance, Passwrd) VALUES ('" &
Username & "','" & Balance & "','" & Password & "')"
```

This is the SQL command. It is a basic Insert command that takes 3 values, corrosponding to the 3 values passed as arguments to the sub-routine when it is called.

LoginForm – Investu - Development 2

Global Variables - LoginForm – Investu Development 2

```
Public Class LoginForm

    Dim AccessDatabaseConnection As String = MainForm.AccessDatabaseConnection

    Public AccountID As Integer
    Public Admin As Boolean
    Public TeamName As String
    Public TeamMode As Boolean = False
    Public Username As String
```

Similarly to SignUpForm, login form has a global variable called ‘AccessDatabaseConnection’. This is global for the same reasons as discussed in SignUpForm – Version 1.

The 5 following variables are all attributes of the account information. These values will all need to be accessible to MainForm when it loads, and so they are defined globally here, using ‘Public’ so that they can be accesed when needed.

Login_Click - LoginForm – Investu Development 2

```
Private Sub Login_Click(sender As System.Object, e As System.EventArgs) Handles
Login.Click
        Username = UsernameTextBox.Text
        If ValidUserLogin(Username, PasswordTextBox.Text) Then
            LoadUserInfo(AccountID)

            If Admin = True Then
                AdminView.Show()
                Me.Close()
            Else
                If TeamModeCheckBox.Checked Then
                    If TeamName = "" Then
                        MsgBox("You don't have a team!”)
                    Else
                        TeamMode = True
                    End If
                End If
                MainForm.Show()
                Me.Close()
            End If
        Else
            MsgBox("Invalid Username or Password.")
        End If
    End Sub
```

Login_Click is a sub-routine that has the handle ‘Login.click’ which means it will be called when the ‘Login’ button is clicked. This is the button the user will press when they have entered all of their login credentials.

```
If ValidUserLogin(Username, PasswordTextBox.Text) Then
    ...
Else
    MsgBox("Invalid Username or Password.")
End If
```

The sub-routine is based off of a series of nested conditionals that work to validate the users login credentials and work out which aspects of the simulation to load.

The first conditional calls the function ‘ValidUserLogin’, and passes two arguments; the users username and pasword.

```
LoadUserInfo(AccountID)
If Admin = True Then
       AdminView.Show()
       Me.Close()
Else
       If TeamModeCheckBox.Checked Then
              If TeamName = "" Then
                     MsgBox("You don't have a team!")
              Else
                     TeamMode = True
              End If
       End If

       MainForm.Show()
       Me.Close()
```

If the value of 'ValidUserLogin' returns as true, the above code runs. Firstly, the information of the user is loaded through the sub-routine 'LoadUserInfo'. This takes an argument that contains the user ID of the account being logged into. The ID of the account is stored in a global variable called AccountID, which is given a value inside the function 'ValidUserLogin', after a username and password match is confirmed. Next, the admin status of the user is checked. This is an attribute of the account that is stored as a boolean value in the database, and determines whether or not the account has admin priveleges. If the value is true, then a seperate form is loaded, called 'AdminView'. This will be developed later, as it is not the main focus of Development 2.

If the 'Admin' attribute of the account has a value of false, then a different conditional runs. This conditional checks whether or not the user has decided to log into their team account, or their personal account. When the user information is loaded, their team name is loaded. This value goes into the variable TeamName. If 'TeamName' is empty, then it means that the user does not belong to a team. The user would then be loaded into their personal account. Else, the value of 'TeamMode' is set to true, and the MainForm is loaded.

ValidUserLogin - LoginForm – Investu Development 2

```
Function ValidUserLogin(ByVal Username As String, ByVal Password As String)

        Dim ConnectionDb As New OleDbConnection(AccessDatabaseConnection)
        If ConnectionDb.State = ConnectionState.Closed Then ConnectionDb.Open()
        Dim cmd As OleDbCommand = ConnectionDb.CreateCommand
        cmd.CommandText = "SELECT AccountID FROM tblUserInfo WHERE Username='" &
Username & "' AND Passwrd='" & Password & "'"

        Dim SQLReply As OleDbDataReader = cmd.ExecuteReader

        For Each Record In SQLReply
            AccountID = Record.item("AccountID")
            Return True
        Next
        ConnectionDb.Close()
        Return False

    End Function
```

'ValidUserLogin' is a function that has two parameters; usrename and password. It uses these to validate the account credentials input by the user. It does this through a Select SQL command.

```
SELECT AccountID FROM tblUserInfo WHERE Username='" & Username & "' AND Passwrd='" &
Password & "'"
```

The item selected is the account ID. This will be useful later in the program, as it is the primary key used for identifying the users account. The account ID will be only be selected from the rows where the 'username' column is the value of the username input by the user, and the 'passwrd' column is the same as the password input by the user, in the login phase. If these two values match, the account ID is fetched, and the function returns true.

There could be concern that if there were two entities in the database with the same username and password, then two account ID's would be fetched. However, thanks to the validation checks done during the sign-up phase, it is not possible to create two accounts with the same username.

LoadUserInfo - LoginForm – Investu Development 2

```
Sub LoadUserInfo(ByVal AccountID As Integer)

        Dim UserValid As Boolean = False
        TeamName = ""

        Dim ConnectionDb As New OleDbConnection(AccessDatabaseConnection)
        If ConnectionDb.State = ConnectionState.Closed Then ConnectionDb.Open()
        Dim cmd As OleDbCommand = ConnectionDb.CreateCommand
        cmd.CommandText = "SELECT tblTeams.TeamName FROM tblTeams, tblTeamUsers,
tblUserInfo WHERE tblTeams.TeamID = tblTeamUsers.TeamID AND tblTeamUsers.AccountID =
tblUserInfo.AccountID AND tblUserInfo.AccountID=" & AccountID & ""

        Dim SQLReply As OleDbDataReader = cmd.ExecuteReader

        For Each Record In SQLReply
            TeamName = Record.item("TeamName")
        Next
        SQLReply.Close()

        cmd.CommandText = "SELECT Admin FROM tblUserInfo WHERE AccountID=" & AccountID
& ""

        Dim SQLReply1 As OleDbDataReader = cmd.ExecuteReader

        For Each Record In SQLReply1
            Admin = Record.item("Admin")
        Next
        ConnectionDb.Close()
    End Sub
```

This sub-routine uses SQL commands, as previously discussed, to load the information of the user. The sub-routine takes only one argument, which is the account ID of the account about which information is being retrieved.

The syntax for this section of code is the same as previous Select commands discussed earlier.

MainForm – Investu - Development 2

The second development of MainForm will allow the simulation to fulfil the specific objectives for Development 2 set out in the beginning of the section. This includes integrating the account system, which has been started by the creating of 'LoginForm' and 'SignupForm' earlier, and adding team capabilities, as well as connecting the database to allow for saved progress.

Imports/Namespaces– MainForm – Investu Development 2

```
Imports System.IO
Imports System.Xml
Imports System.Windows.Forms.DataVisualization.Charting
Imports System.Data.OleDb
```

The imports of this version of simulation have already been discussed in the Version 1; They include System.IO for reading text files, system.XML for reading XML data, System.Windows.Forms.DataVisualization.Charting for graphing stock market data, and System.Data.OleDb for accessing the database using SQL.

Global Variables – MainForm – Investu Development 2

```
Public Class MainForm

    Public AccessDatabaseConnection As String = "Provider=Microsoft.ACE.OLEDB.12.0;Data
Source=; Data Source=C:\Users\Joe\Documents\visual studio 2010\Investu
Writeup\Development 2\StockInfoDB.mdb"

    Public OpenPositions As New List(Of StockAttributes)
    Public Symbols As New List(Of String)

    Dim Series1 As New Series
    Dim LastValue As Decimal

    Public AccountID As Integer
    Public TeamMode As Boolean
    Public TeamName As String
    Public Balance As Decimal
```

Inside the header of version 2 of MainForm, some changes have been made in comparison to version 1. The variables 'Balance', 'StockInfo', 'TimerInterval' have all been removed, and are now integrated into the code locally in their respective sub-routines.

The list 'Symbols' has been updated from a static array to a list structure. This works exactly the same way as described in InvestuServerProgram, in which the symbols for the FTSE100 stocks are read from a .CSV file.

```
Public AccountID As Integer
Public TeamMode As Boolean
Public TeamName As String
Public Balance As Decimal
```

These variables are used for holding and passing account information. The value of these variables is passed from LoginForm and so for now it is easier to declare them globally. In the next development, this could change to remove the need of global variables.

```
Public ErrorMsg As String
```

‘ErrorMsg’ is a variable, to which errors will be passed when they arise, and then the value of ‘ErrorMsg’ will be output at a set point, instead of having message boxes throughout the code. This allows for a more stream lined error checking process, as all of the error messages are contained within a single variable.

MainForm_Load – MainForm – Investu Development 2

This sub-routine is run on loading of MainForm, which occurs after a successful login attempt by the user.

```
     Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles MyBase.Load

        AccountID = LoginForm.AccountID
        TeamMode = LoginForm.TeamMode

        If TeamMode = True Then
            TeamName = LoginForm.TeamName
            LoginLabel.Text = "Welcome, " & LoginForm.Username & ". (" & TeamName & ")"
        ElseIf TeamMode = False Then
            TeamName = "0"
            LoginLabel.Text = "Welcome, " & LoginForm.Username & ""
        End If

        Balance = Math.Round(FetchBalance(), 2)
        BalanceBox.Text = "£" & Math.Round((Balance / 100), 2)

        FetchOpenPositions()

        CreateChart()
        GraphSettings()


        GraphScaleComboBox.SelectedItem = "2"

        PopulateSymbolArray("C:\Users\Joe\Documents\visual studio 2010\Investu
Writeup\Development 2\StockSymbols.csv")
        For L = 0 To Symbols.Count - 1
            SelectStockComboBox.Items.Add(Symbols(L))
        Next
    End Sub
```

```
        AccountID = LoginForm.AccountID
        TeamMode = LoginForm.TeamMode
```

Two of the global variables declared earlier are given values. These values are taken from LoginForm.

```
If TeamMode = True Then
            TeamName = LoginForm.TeamName
            LoginLabel.Text = "Welcome, " & LoginForm.Username & ". (" & TeamName & ")"
ElseIf TeamMode = False Then
            TeamName = "0"
            LoginLabel.Text = "Welcome, " & LoginForm.Username & ""
End If
```

If the newly defined value of ‘TeamMode’ is true, then the value of ‘TeamName’ is retrieved, and a label is updated in order to tell the user they are in team mode, by appending their team name inside brackets. If the value of ‘TeamMode’ is false, then the user is signed into their personal accounts. ‘TeamName’ is set to “0” and the value of the label is simply set to their username, without a team name.

PopulateSymbolArray– MainForm – Investu Development 2

```
Public Sub PopulateSymbolArray(ByVal FilePath As String)

        Dim CSVData() As String

        Using SR As New StreamReader(FilePath)

            While Not SR.EndOfStream
                CSVData = SR.ReadLine().Split(",")
                If String.IsNullOrEmpty(CSVData(0)) Then
                    MsgBox("Error loading FTSE 100")
                Else
                    Symbols.Add(CSVData(0).Trim)
                End If

            End While
        End Using

    End Sub
```

This sub-routine is used to populate the ‘Symbols’ list. This is the same code as used in ‘InvestuServerProgram’ on page 106

Timer1_Tick – MainForm – Investu Development 2

```
Private Sub Timer1_Tick(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles Timer1.Tick

        Dim StockInfoString

        Try
            Timer1.Interval = 60000
            StockInfoString = FetchStockDetailsString(SelectStockComboBox.SelectedItem)
            NameBox.Text = SplitStockInfo(StockInfoString, "Name")
            PriceBox.Text = SplitStockInfo(StockInfoString, "Price")
            ChangeBox.Text = SplitStockInfo(StockInfoString, "Change")

            Series1.Points.Clear()
            Plot24hrData()
            UpdatePortfolio()
            GraphSettings()

            VolatilityBox.Text = CalculateVolatility(PriceBox.Text, ChangeBox.Text) &
"%"
        Catch ex As Exception
            MsgBox(ex.ToString())
        End Try
    End Sub
```

Timer1_Tick has been changed since development 1. In this development, the timer has a different interval – 60000 milliseconds as opposed to 5000 in the earlier version. This is because the way in which data is plotted to the graph has been changed to work with the new InvestuServerProgram. The data for the graphs is now retrieved from the data base, instead of directly through the simulation itself, with the data of course being supplied through the server program. The server program works by retrieving the stock information of a new stock every second for one hundred seconds, and then once the last of the 100 symbols is reached, the process repeats, and the first symbol is queried. This effectively means that each stock only receives new data every 100 seconds. Therefore, when the user is looking at a graph, there is no reason to update it every 5 seconds, as it will only show a change every 20 updates, which is a waste of processing power.

The logic behind retrieving data from the database instead of through the simulation itself, is that the database can store hundreds of thousands of data points, whereas the program can only plot data points that it has collected during its runtime.

CalculateVolatility – MainForm – Investu Development 2

```
Function CalculateVolatility(ByVal Price As Decimal, ByVal Change As Decimal)

        Dim Volatility As Decimal
        Price = Math.Abs(Price)
        Change = Math.Abs(Change)

        If Price <> 0 And Change <> 0 Then
            Volatility = (Change / Price) * 100
            Volatility = Math.Round(Volatility, 2)
        Else
            Volatility = 0
        End If

        Return Volatility
    End Function
```

Another feature provided to the user is the volatility of a stock. This is a feature outlined in the features list, derived from the interaction with the client and user. To show the volatility of a stock, all that needs to be done is to work out out the change in a stock as a percentage of its current price. For example, if a share of a company starts the trading day at 100 pence, and then changes to 101, then the intra-day change is 1, which is 1% of the current price, meaning it has a 1% volatility for the day so far.
This is done by passing ‘Price’ and ‘Change’ to the function as decimals, and then validating that both the values are greater than 0. Then, the value of ‘Volatility’ is worked out as a percentage using a simple calculation. This value is rounded and returned.

FetchBalance – MainForm – Version 2

```
Function FetchBalance()

        Dim CommandString As String

        If TeamMode = True Then
            CommandString = "SELECT Balance FROM tblTeams WHERE TeamName='" & TeamName
& "'"
        Else
            CommandString = "SELECT Balance FROM tblUserInfo WHERE AccountID=" &
AccountID & ""
        End If

        Using Connection As New OleDbConnection(AccessDatabaseConnection)

            Dim Command As New OleDbCommand(CommandString, Connection)
            Connection.Open()
            Dim reader As OleDbDataReader = Command.ExecuteReader()

            While reader.Read()
                Balance = reader(0)
            End While

            reader.Close()
        End Using

        Return Balance
    End Function
```

```
            CommandString = "SELECT Balance FROM tblTeams WHERE TeamName='" & TeamName &
"'"
        Else
            CommandString = "SELECT Balance FROM tblUserInfo WHERE AccountID=" &
AccountID & ""
        End If
```

Now that the team system has been integrated, it is no longer possible to simply retrieve information using an account ID. Instead, the users team mode has to be checked. If it is found to be true, then the balance value is fetched from 'tblTeams', whereas if it is false, the value for the balance is fetched from the tblUserInfo table.

FetchOpenPositions – MainForm – Investu Development 2

```
Sub FetchOpenPositions()

        Dim ConnectionDb As New OleDbConnection(AccessDatabaseConnection)
        If ConnectionDb.State = ConnectionState.Closed Then ConnectionDb.Open()

        Dim cmd As OleDbCommand = ConnectionDb.CreateCommand

        If TeamMode = True Then
            cmd.CommandText = "SELECT * FROM tblOpenPositions WHERE
tblOpenPositions.TeamName='" & TeamName & "'"
        Else
            cmd.CommandText = "SELECT * FROM tblOpenPositions WHERE AccountID='" &
AccountID & "' AND TeamName='0'"

        End If



        Dim SQLReply As OleDbDataReader = cmd.ExecuteReader

        For Each Record In SQLReply
            OpenPositions.Add(New StockAttributes With {.StockSymbol =
Record.item("StockSymbol"), .StockValue = Record.item("BuyPrice"), .StockQuantity =
Record.item("StockQuantity"), .OpenPositionID =
Record.item("OpenPositionID"), .StockName = Record.item("StockName"), .BuyDate =
Record.item("TradeDate")})
            UpdatePortfolio()
```

FetchOpenPositions is the sub-routine that allows the user to keep their trading progress. The sub-routine fetches all of the open positions that the user has. This will occur on logging into the simulation. Combined with the ability to save and fetch the users balance, the user will now be able to maintain progress after logging out of their team or personal account.

```
        If TeamMode = True Then
            cmd.CommandText = "SELECT * FROM tblOpenPositions WHERE
tblOpenPositions.TeamName='" & TeamName & "'"
        Else
            cmd.CommandText = "SELECT * FROM tblOpenPositions WHERE AccountID='" &
AccountID & "' AND TeamName='0'"

        End If
```

The sub-routine begins with the standard connection to database code, which is followed by a conditional. Within the conditional, the value of the command text, or SQL statement, is set. The value of the

command text again is dependent on the value of the boolean ‘TeamMode’. If the value is true, then the following command text is set;

```
SELECT * FROM tblOpenPositions WHERE tblOpenPositions.TeamName='" & TeamName & "'
```

Which simply selects all information from the table ‘tblOpenPositions’ where the value of ‘TeamName’ is the value stored in the variable ‘TeamName’ within the program.

```
SELECT * FROM tblOpenPositions WHERE AccountID='" & AccountID & "' AND TeamName='0'
```

If the value of ‘TeamMode’ is false, then the users personal trades are fetched. To find these, two values are checked; AccountID and TeamName. First, the value of the column ‘AccountID’ is compared with the account ID within the simulation. However, another check needs to be made, as if only this check was used, it would return trades made by the user not only on their personal account, but also on the team account. Therefore we must make sure the trades that are fetched were not done on a team account. To do this, the value of the ‘TeamName’ column is checked. When a user logs into a personal account, the value of ‘TeamName’ is set to ‘0’. This is an arbitrary value that simply denotes that the user is not in a team. When the user makes executes a trade, this value will go into the ‘TeamName’ column in open trades, to show that the trade was not made on a team account, but a personal account. Therefore by also checking that ‘TeamName’ is ‘0’, it can be made certain that all of the trades fetches are those made on the users personal account.

```
        Dim SQLReply As OleDbDataReader = cmd.ExecuteReader

        For Each Record In SQLReply
            OpenPositions.Add(New StockAttributes With {.StockSymbol =
Record.item("StockSymbol"), .StockValue = Record.item("BuyPrice"), .StockQuantity =
Record.item("StockQuantity"), .OpenPositionID = Record.item("OpenPositionID"), .StockName
= Record.item("StockName"), .BuyDate = Record.item("TradeDate")})
            UpdatePortfolio()
```

Once the command text is set, it is executed. ‘SQLReply’ is a variable with the data type ‘OleDbDataReader’, from the System.OleDb namespace. This variable is then set to the value of ‘cmd.ExecuteReader’ which effectively executes the SQL command on the database that is currently connected.

This returns a series of results. A For-Loop is used to iterate through the results, within which, ‘.add’ is used on the ‘OpenPositions’ list, with the value of each of the attributes set to the corrosponding value. e.g.

```
.StockSymbol = Record.item("StockSymbol")
```

This sets the StockSymbol attribute of the new addition to OpenPositions equal to the value of ‘StockSymbol’ of the current item being iterated through.

After this For-Loop terminates, every open position that the user has will have been added to the list ‘OpenPositions’. From here, the open positions can be manipulated, appended, editted and displayed, depending on what is required by the user.

FetchStockDetailsString – MainForm – Investu Development 2

```
Function FetchStockDetailsString(ByVal StockSymbol As String)
```

This function is the same as seen in Development 1.

SplitStockInfo – MainForm – Investu Development 2

```
Function SplitStockInfo(ByVal StringToSplit As String, ByVal DetailsToExtract As String)
```

This function is the same as seen in Development 1.

BuyButton_Click – MainForm – Investu Development 2

```
Private Sub BuyButton_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles BuyButton.Click
```

This function is the same as seen in Development 1.

UpdatePortolio – MainForm – Investu Development 2

```
Sub UpdatePortfolio()

        Dim TotalTradePrice As Decimal
        Dim CurrentTotalPrice As Decimal

        OpenPositionsListBox.Items.Clear()

        For l = 0 To OpenPositions.Count - 1
            TotalTradePrice = Math.Round(((OpenPositions(l).StockValue *
OpenPositions(l).StockQuantity) / 100), 2)

            CurrentTotalPrice = Math.Round(((GetStockPrice(OpenPositions(l).StockSymbol)
* OpenPositions(l).StockQuantity) / 100), 2)
            OpenPositionsListBox.Items.Add(OpenPositions(l).StockName & " - Bought " &
OpenPositions(l).StockQuantity & " FOR " & TotalTradePrice & " (" &
OpenPositions(l).StockValue & " each)" & vbNewLine)
        Next

End Sub
```

The sub-routine 'UpdatePortfolio' is called to populate the visual display of the simulation with the open positions of the user. It is called on load of the simulation and upon the execution of a new trade. It first clears the visual display that holds the open positions, and then repopulates it with the contents of 'OpenPositions', along with relevant, recent information regarding prices.

```
Dim TotalTradePrice As Decimal
Dim CurrentTotalPrice As Decimal
```

Two variables that will be displayed are 'TotalTradePrice' – the total price of the shares at the time of the execution of the trade, and 'CurrentTotalPrice' which is the total price of the shares at the current time. This gives two different numbers, the difference between which will be the total profit or loss on that particular position.

```
For l = 0 To OpenPositions.Count – 1
    …
Next
```

This For-Loop loops through every open position in 'OpenPositions'

```
TotalTradePrice = Math.Round(((OpenPositions(l).StockValue *
OpenPositions(l).StockQuantity) / 100), 2)
```

Then, the value of 'TotalTradePrice' is calculated for each item inside the list. This is done by fetching the value the ''StockValue' property, of the current item in 'OpenPositions'. This is then multiplied by the value of the 'StockQuantity' property. This gives the total price of the trade in pence. This is divided by 100 and rounded to give a value in pounds.

```
CurrentTotalPrice = Math.Round(((GetStockPrice(OpenPositions(l).StockSymbol) *
OpenPositions(l).StockQuantity) / 100), 2)
```

A similar process happens with 'CurrentTotalPrice', however this time the price is replaced by 'GetStockPrice' instead of the 'StockValue' property. This function takes a single argument: the stock symbol of the stock being inspected. This is retrieved from OpenPositions, as it is one of the properties of the items within the list. This function returns the current most recent price of the stock in question, which is then multiplied by the quantity to give an updated total price.

```
OpenPositionsListBox.Items.Add(OpenPositions(l).StockName & " - Bought " &
OpenPositions(l).StockQuantity & " FOR £" & TotalTradePrice & " (" &
```

```
OpenPositions(l).StockValue & " each) PROFIT=£" & CurrentTradePrice - TotalTradePrice
vbNewLine)
```

The item at index value 'L' of the list is then written to a list box called 'OpenPositionsListBox'. An example of the string that would be output would be as follows:

```
BARC.L – Bought 3450 FOR £45,600 (1,321 EACH) PROFIT=£2,346
```

SelectStockComboBox_SelectedIndexChanged – MainForm – Investu Development 2

```
Private Sub SelectStockComboBox_SelectedIndexChanged(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles SelectStockComboBox.SelectedIndexChanged

        Timer1.Interval = 1
        Timer1.Start()
        Series1.Points.Clear()
        Plot24hrData()
    End Sub
```

When the selected index of the 'SelectedStock' combo box is changed – e.g. the user selects a stock symbol to inspect, this sub-routine triggers, due to its handle of '`SelectStockComboBox.SelectedIndexChanged`'. The code is similar to that in Version 1 of this sub-routine, found in Development 1, with the addition of 'Plot24hrData()'. This sub routine plots the data of the last 24 hours of the stock data to the graph, read from the database.

Plot24hrData – MainForm – Investu Development 2

```
Public Sub Plot24hrData()

        Dim Query As String = "SELECT FetchDate, StockPrice FROM tblStockPriceHistory
WHERE StockSymbol = '" & SelectStockComboBox.SelectedItem & "' ORDER BY FetchDate"

        Using connection As New OleDbConnection(AccessDatabaseConnection)
            Dim command As New OleDbCommand(Query, connection)

            connection.Open()

            Dim reader As OleDbDataReader = command.ExecuteReader()

            While reader.Read()

                If reader(0) >= DateTime.Today Then
                    PlotNewPoint((reader(0)).ToOADate(), reader(1))
                    LastValue = reader(1)
                End If

            End While
            reader.Close()
        End Using
End Sub
```

```
SelectStockComboBox.SelectedItem & "' ORDER BY FetchDate
```

The SQL command for this retrieval simply finds the rows in the table whose 'StockSymbol' value is the same as the currently selected stock symbol (selected from the drop down box 'SelectStockComboBox') and then retrieves the values of the columns 'FetchDate' and 'StockPrice' for those instances. The information is ordered by 'FetchDate' to make sure that the information plotted to the graph is in chronological order.

```
While reader.Read()
     If reader(0) >= DateTime.Today Then
            PlotNewPoint((reader(0)).ToOADate(), reader(1))
            LastValue = reader(1)
     End If
End While
```

After a standard database connection is established, and the command text set earlier is executed, a While-Loop is initiated. This effectively loops through every row of information returned. Inside the loop is a conditional which checks the value in reader(0) against the current time. The value in reader(0) is the value of 'FetchDate'. What this is essentially doing is only allowing data from the current day to be plotted to the graph. This could be edited in further versions to be controllable by the user, however for this development the graph will only show data for the current day of trading.

```
PlotNewPoint((reader(0)).ToOADate(), reader(1))
LastValue = reader(1)
```

Inside the conditional, the sub-routine 'PlotNewPoint' is called. This sub-routine takes two arguments – a value for the X axis and a value for the Y axis, which in this case is the date and the price respectively.

Then, the value of a variable 'LastValue' is set to reader(0). This will be overwritten every loop, until the final loop, meaning that the value after the loop exits will be the most recent price of the stock. Having this value will be useful for manipulation of the graph visuals later.

PlotNewPoint – MainForm – Investu Development 2

```
Sub PlotNewPoint(ByVal XValue As Decimal, ByVal YValue As Decimal)
      Series1.Points.AddXY(XValue, YValue)
End Sub
```

This sub-routine adds a new point to the series called 'Series1', using values of the arguments 'XValue' and 'YValue' as the X and Y values respectively.

GetStockPrice – MainForm – Investu Development 2

```
Function GetStockPrice(ByVal StockSymbol As String)
```

This function is taken from the development of Investu Server Program at the start of Development 2.

GetStockChange – MainForm – Investu Development 2

```
Function GetStockChange (ByVal StockSymbol As String)
```

This function is taken from the development of Investu Server Program at the start of Development 2.

GetStockName – MainForm – Investu Development 2

```
Function GetStockName(ByVal StockSymbol As String)
```

This function is taken from the development of Investu Server Program at the start of Development 2.

ClosePositionbutton_Click – MainForm – Investu Development 2

```
Private Sub ClosePositionsButton_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles ClosePositionsButton.Click

        Try
            Dim NewStockPrice As Decimal
            Dim SelectedStock As String = OpenPositionsListBox.SelectedIndex

            If OpenPositionsListBox.CheckedItems.Count = 1 Then
                NewStockPrice =
GetStockPrice(OpenPositions(SelectedStock).StockSymbol)
                Balance = Balance + (OpenPositions(SelectedStock).StockQuantity *
NewStockPrice)
                BalanceBox.Text = "£" & Math.Round((Balance / 100), 2)

                Dim CommandString As String

                If TeamMode Then
                    CommandString = "UPDATE tblTeams SET tblTeams.Balance=" & Balance
& " WHERE tblTeams.TeamName='" & TeamName & "';"
                Else
                    CommandString = "UPDATE tblUserInfo SET tblUserInfo.Balance=" &
Balance & " WHERE (((tblUserInfo.AccountID)=" & AccountID & "));"
                End If

                Dim ConnectionDb As New OleDbConnection(AccessDatabaseConnection)
                If ConnectionDb.State = ConnectionState.Closed Then
ConnectionDb.Open()

                Dim cmd As OleDbCommand = ConnectionDb.CreateCommand

                cmd.CommandText = CommandString
                cmd.ExecuteNonQuery()

                cmd.CommandText = "DELETE * FROM tblOpenPositions WHERE
OpenPositionID='" & OpenPositions(SelectedStock).OpenPositionID & "' "
                cmd.ExecuteNonQuery()
                ConnectionDb.Close()

                OpenPositions.RemoveAt(OpenPositionsListBox.SelectedIndex)
                UpdatePortfolio()
        Catch ex As Exception
            MsgBox(ex.ToString())
        End Try
End Sub
```

The second development of this sub-routine develops on the idea of keeping records to keep track of the users progress.

```
If OpenPositionsListBox.CheckedItems.Count = 1 Then
  ....
Else
   MsgBox("Please select the position you'd like to close.")
End If
```

A conditional first checks whether there is a selected position to close. If there are 0 or 2 or more selected positions then the following code will throw an error, and so it is important that this is checked.

```
NewStockPrice = GetStockPrice(OpenPositions(SelectedStock).StockSymbol)
Balance = Balance + (OpenPositions(SelectedStock).StockQuantity * NewStockPrice)
BalanceBox.Text = "£" & Math.Round((Balance / 100), 2)
```

The following 3 lines of code work out new prices and balances. ‘NewStockPrice’ is set to the latest price value of the current stock, retrieved using the ‘GetStockPrice’ function. Then, this value is mutliplied by the stock quantity to work out the users new balance. Obviously, as this sub-routine represents the execution of a sale, the value is added to the balance, not taken away like in the ‘BuyForm’ sub-routine. Finally, the balance is formatted and displayed.

```
Dim CommandString As String

If TeamMode Then
      CommandString = "UPDATE tblTeams SET tblTeams.Balance=" & Balance & " WHERE
tblTeams.TeamName='" & TeamName & "';"
Else
      CommandString = "UPDATE tblUserInfo SET tblUserInfo.Balance=" & Balance & " WHERE
(((tblUserInfo.AccountID)=" & AccountID & "));"
End If
```

Similarly to other sub-routines, ClosePositionsButton_Click now incorporates a conditional to check the value of ‘TeamMode’, and uses different SQL command texts depending on the value. If the player is in team mode, then the following SQL is used:

```
UPDATE tblTeams SET tblTeams.Balance=" & Balance & " WHERE tblTeams.TeamName='" &
TeamName & "';
```

This command updates the balance of the team, by making use of the ‘UPDATE’ command in SQL. The ‘Balance’ column in ‘tblTeams’ is set to the value of ‘balance’ which was just changed in the previous few lines of code. This change occurs where there is a match between the column value of ‘TeamName’ and the ‘TeamName’ that is stored in the variable of the same name inside the simulation.

If the player is not in team mode, then alternative SQL is used:

```
UPDATE tblUserInfo SET tblUserInfo.Balance=" & Balance & " WHERE
(((tblUserInfo.AccountID)=" & AccountID & "));
```

This performs the same task, however uses updating the table ‘tblUserInfo’ instead of ‘tblTeams’. The balance is set where the exists a match between the column ‘AccountID’ and the variable ‘AccountID’ within the simulation.

```
Dim ConnectionDb As New OleDbConnection(AccessDatabaseConnection)
```

```
If ConnectionDb.State = ConnectionState.Closed Then ConnectionDb.Open()

Dim cmd As OleDbCommand = ConnectionDb.CreateCommand

cmd.CommandText = CommandString
cmd.ExecuteNonQuery()
```

After the value of ‘CommandString’ is determined, the connection to the database is opened with the standard connection procedure discussed in earlier sections. The command text is set to ‘CommandString’ and the NonQuery is executed. The balance has now been updated to reflect the changes caused by the execution of the trade.

```
cmd.CommandText = "DELETE * FROM tblOpenPositions WHERE OpenPositionID='" &
OpenPositions(SelectedStock).OpenPositionID & "' "
cmd.ExecuteNonQuery()

ConnectionDb.Close()
```

Before the connection to the database is closed, another SQL command is executed. This one simple deletes any trace of that position from the database, in order to keep the information in the table correct. This is done with a ‘DELETE *’ SQL command, acting on the ‘OpenPositions’ table, where the ID’s of the positions match.

```
OpenPositions.RemoveAt(OpenPositionsListBox.SelectedIndex)
```

The final two lines of code in this sub-routine are used for updating the visual display. Because of the fact that the items in the list box are written using the items in the OpenPositions list, the index values of both the list and box match. For example, OpenPositions(5) is the same item as is stored in OpenPositionsListBox(5). Therefore, by removing knowing the index value of an item in the list box, the index value of that item in the OpenPositions list is automatically known. This line uses the ‘.RemoveAt’ feature of lists which removes an item at a given index, the given index in this case being the index value of the currently selected item in the list box.

```
UpdatePortfolio()
```

Now that an item has been removed from the ‘OpenPositions’ list, the index values of the box and the list no longer match up. To rectify this, ‘UpdatePortfolio’ is called, which clears the box and rewrites the contents from the ‘OpenPositions’ list.

LogoutButton_Click – MainForm – Investu Development 2

```
    Private Sub LogoutButton_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles LogoutButton.Click

        LoginForm.Show()
        Me.Close()

    End Sub
```

This sub routine simply allows the user to log out of their account. This is done by first showing the login form, and then closing the main form, with 'me.close()'

GraphScaleComboBox_SelectedIndexChanged – MainForm – Investu Development 2

```
Private Sub GraphScaleComboBox_SelectedIndexChanged(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles GraphScaleComboBox.SelectedIndexChanged
        Timer1.Interval = 1
End Sub
```

To make the user experience smoother, this sub-routine is called when the user selects a new stock symbol from the drop down menu, and sets the timer interval to 1 millisecond. This effectively forces a timer tick, causing the visual display to instantly update with information about the selected stock – this stops the using having to wait for the timer to tick naturally, which could take up to 5 seconds.

OpenPositionsListBox_ItemCheck – MainForm – Investu Development 2

```
Private Sub OpenPositionsListBox_ItemCheck(ByVal sender As Object, ByVal e As
System.Windows.Forms.ItemCheckEventArgs) Handles OpenPositionsListBox.ItemCheck

        If e.NewValue = CheckState.Checked Then

            For i As Integer = 0 To Me.OpenPositionsListBox.Items.Count - 1 Step 1
                If i <> e.Index Then Me.OpenPositionsListBox.SetItemChecked(i, False)
            Next
        End If
End Sub
```

This routine is reused from Development 1. It ensures only one box can be checked at a time in 'OpenPositionsListBox'. This will help to mitigate errors caused by users executing multiple trades at once.

CreateChart() – MainForm – Investu Development 2

```
Sub CreateChart()

        Series1.Name = SelectStockComboBox.SelectedItem
        Series1.ChartType = SeriesChartType.Line
        Series1.BorderWidth = 4
        Chart1.Series.Add(Series1)
        Chart1.Legends.Clear()
        Series1.XValueType = ChartValueType.DateTime
        Series1.BorderWidth = 2

    End Sub
```

This sub-routine is taken from Development 1.

GraphSettings()_SelectedIndexChanged – MainForm – Investu Development 2

```
    Sub GraphSettings()

        Chart1.ChartAreas(0).AxisY.Minimum = LastValue - Val(GraphScaleComboBox.Text)
        Chart1.ChartAreas(0).AxisY.Maximum = LastValue + Val(GraphScaleComboBox.Text)
        Chart1.Update()

    End Sub
```

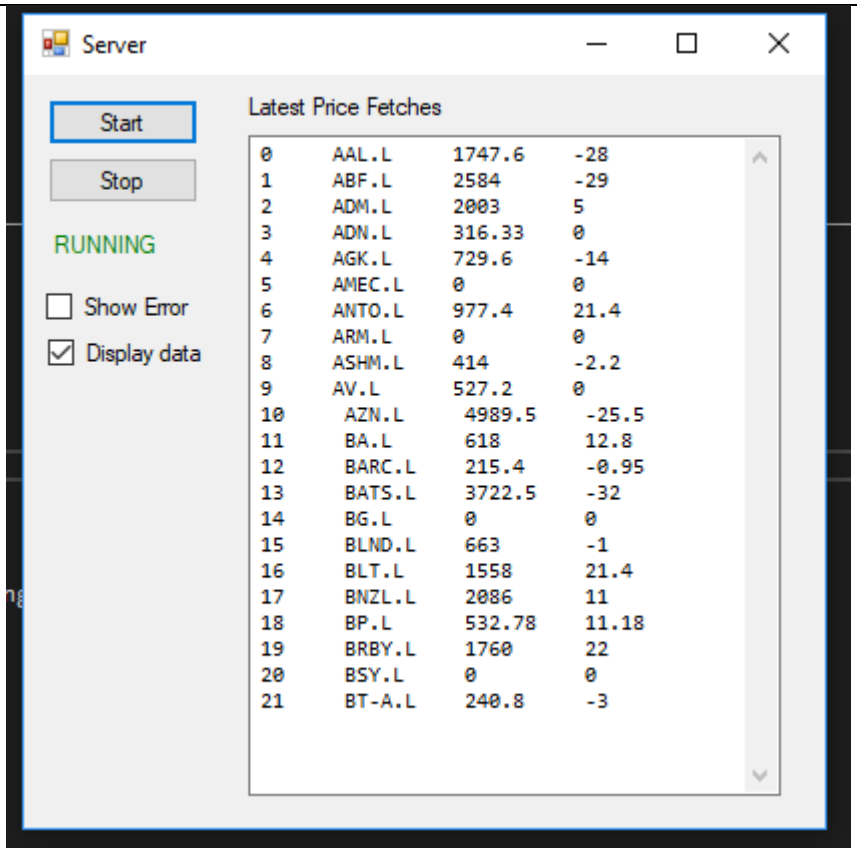
These two sub-routines are reused from Development 1, and are responsible for creating and formatting the graph when it is initialised upon loading into the simulation.

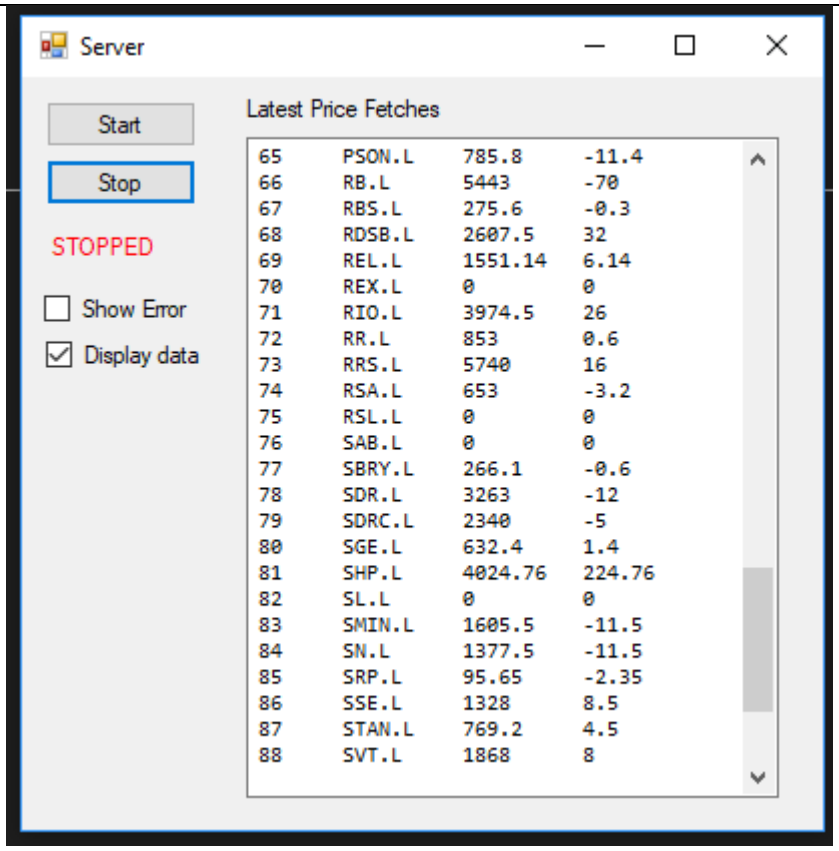
Testing 1 - Investu Server Program – Development 2

(The FTSE100 trading hours are between 9:00am and 16:29pm, so the server program would normally only collect data between those times. For these tests, that time restrcition has been removed.)

Every sub-routine has been tested individually to ensure it works independently, however these tests have not been shown. The tests displayed are tests that show multiple sub-routines and functions working together to produce the desired outcome.

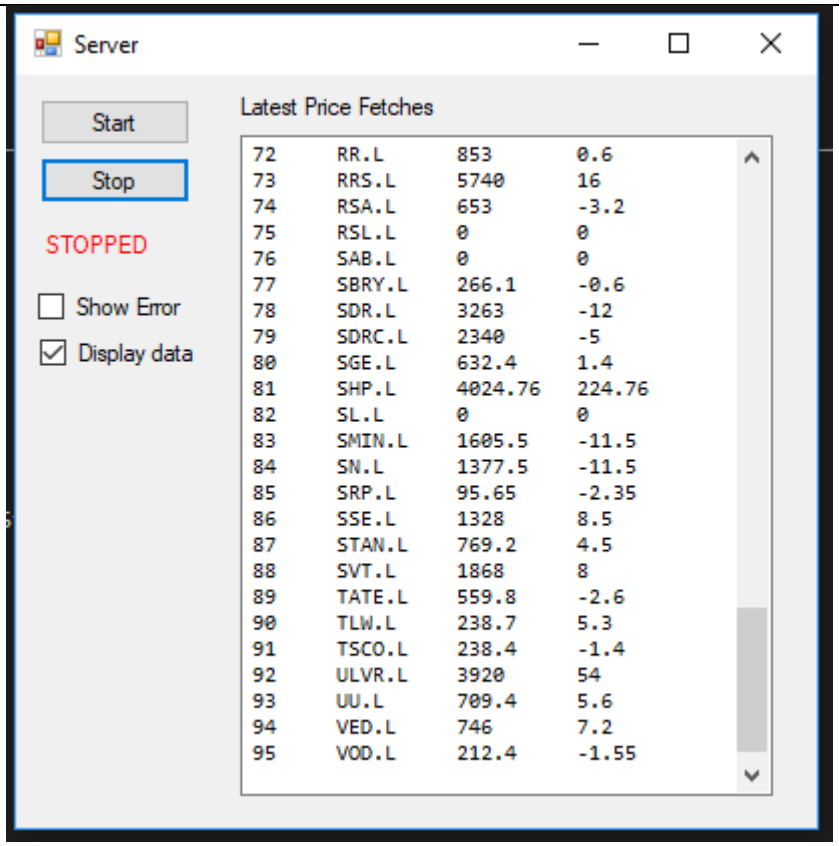
Test Objective	Evidence	Objectives met?

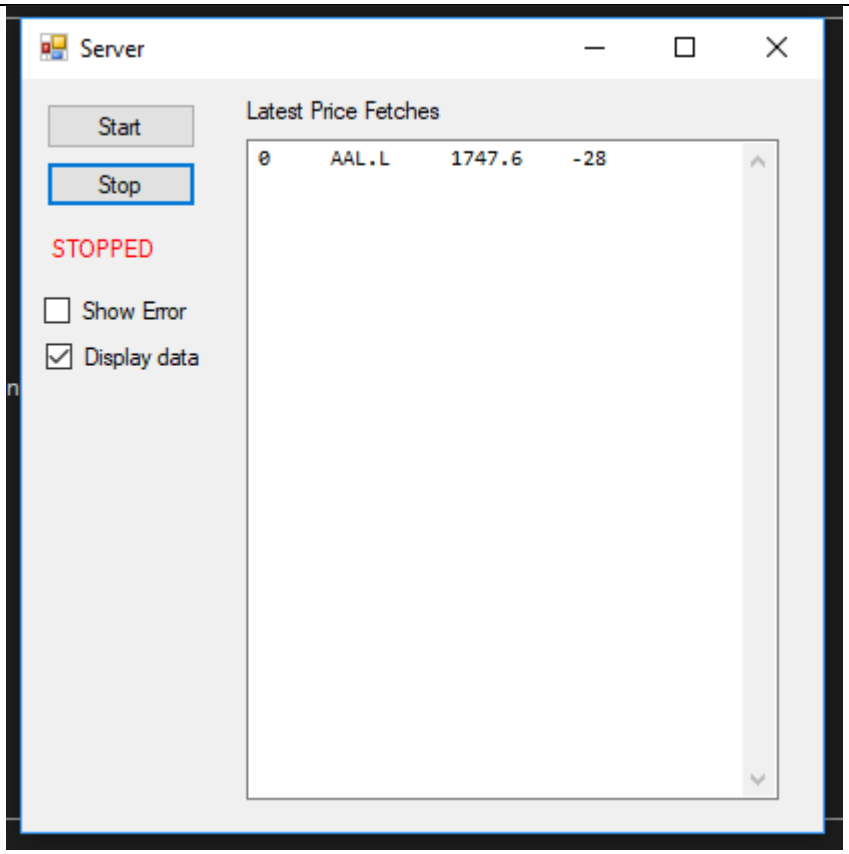
Show that stock information for each stock symbol in the StockSymbols.csv file is queried and the information is displayed	 Above is the result of loading the program, clicking the 'Start' button, and waiting around 20 seconds. The program has looped just over 20 times, indicating that the idea of checking a new symbol every second is working correctly.	



The program continues to fetch the data, incrementing through the symbols, all the way to the end of the list.

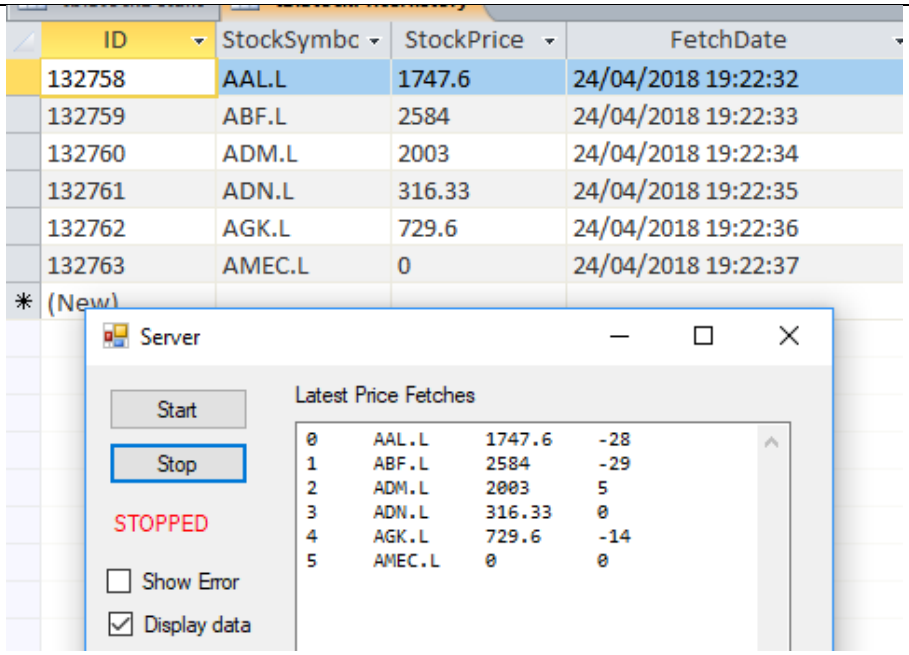
This indicates that the requirements set out in the test description have been met and the test has therefore been passed.

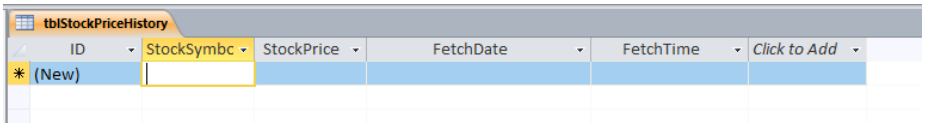
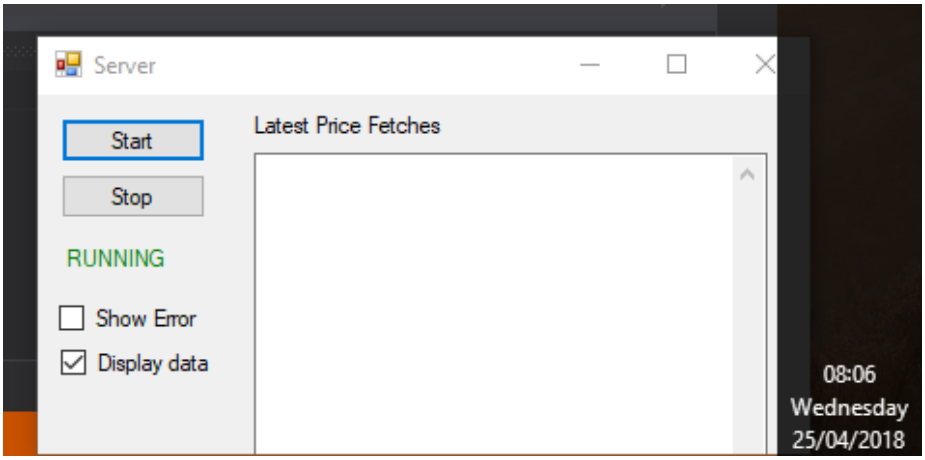
Show that the when every symbol has been queried, the loop resets, and the first symbol is queried again	 The program nears the end of the list of symbols.



The program then resets back to the first symbol, and begins looping through again. This indicates that the requirements set out in the test description have been met and the test has therefore been passed.

Show that this data is written to the database



	This indicates that the requirements set out in the test description have been met and the test has therefore been passed.	
Show that data starts being collected when the FTSE100 opens for trading.	For the test, the server program will be left running from before 9:00am, until a few minutes after 9:00am. The table is currently empty as shown below:  And the server program is started at 8:06am:  	

tblStockPriceHistory | tblCrash | tblUserInfo | tblAlerts | tblT

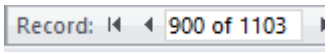
ID	StockSymbc	StockPrice	FetchDate
132764	TATE.L	559.8	25/04/2018 09:00:00
132765	TLW.L	225.9	25/04/2018 09:00:04
132766	TSCO.L	238.9	25/04/2018 09:00:05
132767	ULVR.L	3942	25/04/2018 09:00:06
132768	UU.L	710.52	25/04/2018 09:00:07
132769	VED.L	735.56	25/04/2018 09:00:08
132770	VOD.L	211.9	25/04/2018 09:00:09
132771	WEIR.L	2202	25/04/2018 09:00:10
132772	WOS.L	0	25/04/2018 09:00:11
132773	WPP.L	1123	25/04/2018 09:00:13
132774	WTB.L	4197	25/04/2018 09:00:13
132775	AAL.L	1712.8	25/04/2018 09:00:14
132776	ABF.L	2576	25/04/2018 09:00:15
132777	ADM.L	1998	25/04/2018 09:00:16
132778	ADN.L	316.33	25/04/2018 09:00:17

After leaving the program for a while, data begins to enter the database. Notice that the 'FetchDate' value for the first entry is '09:00:00', and no data before, despite the fact the program was started at 08:06:00am.

Show that the program collects the correct amount of data

133624	ITV.L	143.9	25/04/2018 09:14:49
133625	JMAT.L	3231	25/04/2018 09:14:51
133626	KAZ.L	896.73	25/04/2018 09:14:51
133627	KGF.L	302	25/04/2018 09:14:52
133628	LAND.L	973.45	25/04/2018 09:14:53
133629	LGEN.L	278	25/04/2018 09:14:55
133630	LLOY.L	65.78	25/04/2018 09:14:56
133631	MGGT.L	458.2	25/04/2018 09:14:58
133632	MKS.L	279.9	25/04/2018 09:14:58
133633	MRW.L	234.4	25/04/2018 09:14:59
133655	SGE.L	628.6	25/04/2018 09:15:00

After 15 minutes of collecting data, the above image shows the final entries into the database before 15 minutes has been elapsed.

	 After 15:00 minutes, there are 900 entries. One entry is made into the database per second. 15 minutes is equivalent to 900 seconds. Therefore, there are the correct number of entries in the database. This indicates that the requirements set out in the test description have been met and the test has therefore been passed.	

Show that data for the current day is plotted to the graph in the simulation

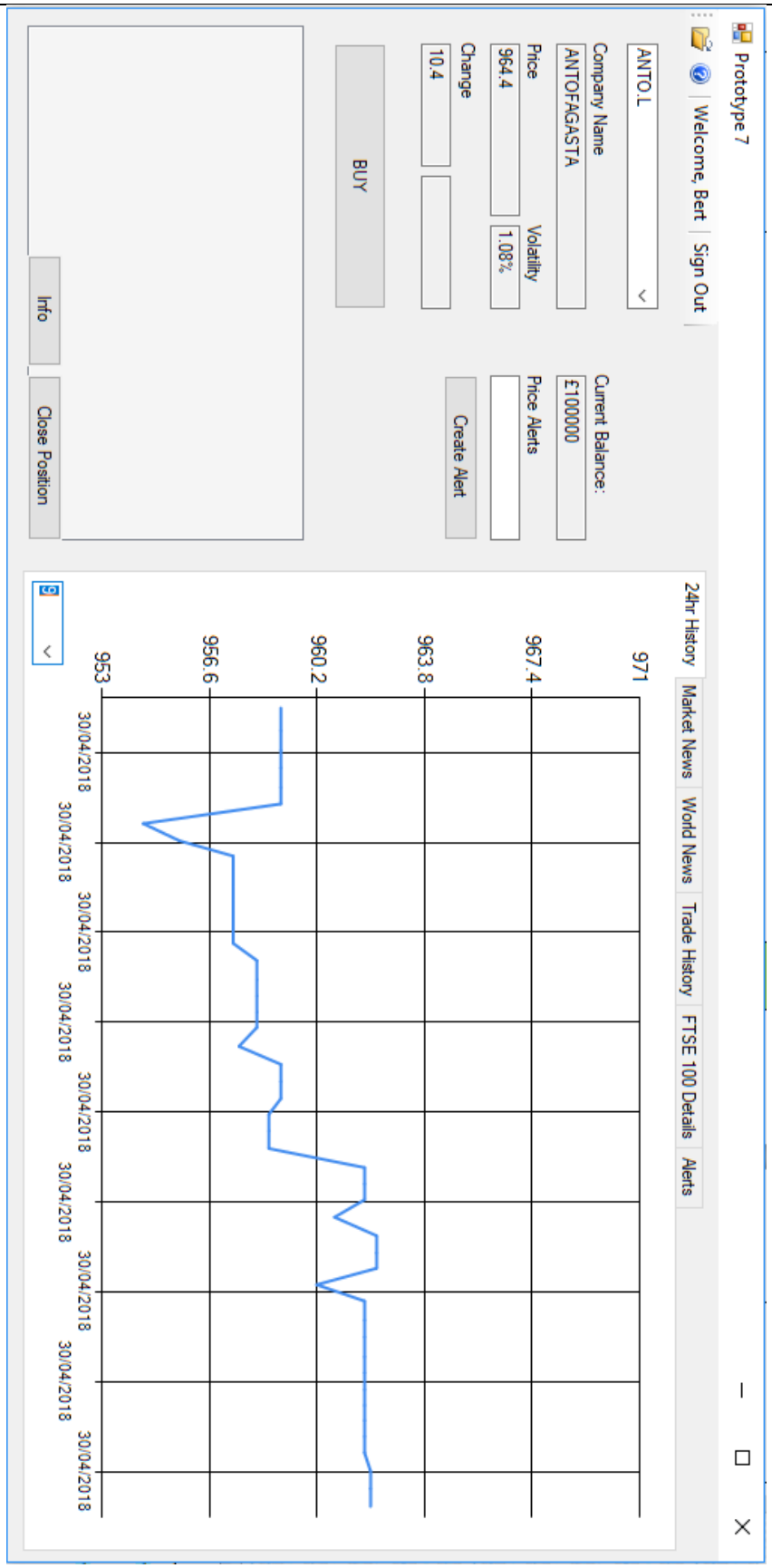
tblStockPriceHistory

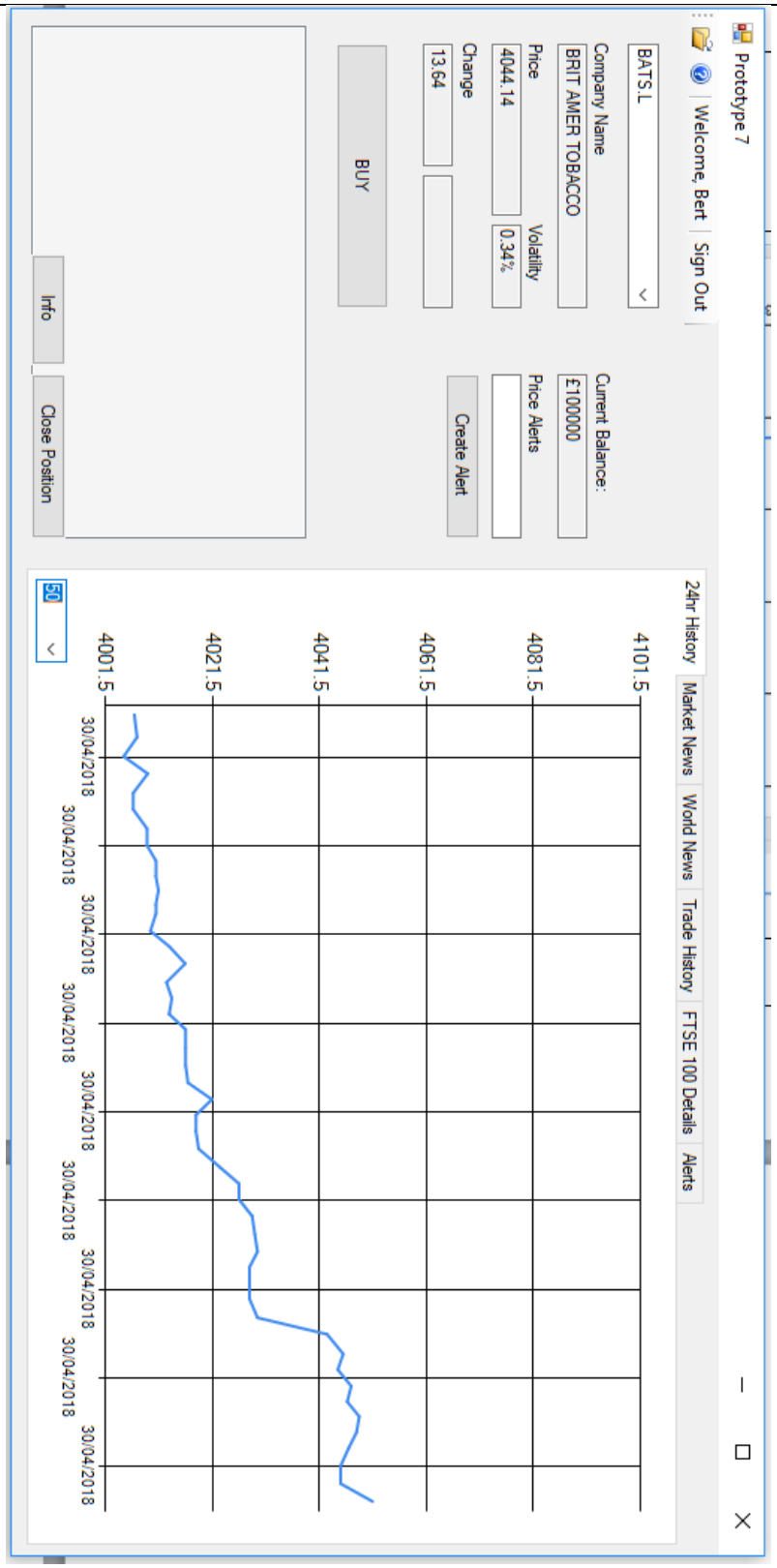
ID	StockSymbc	StockPrice	FetchTime	FetchDate
132655	EMG.L	181.7	09:03:27	30/04/2018 09:03:27
132656	ENRC.L	0	09:03:28	30/04/2018 09:03:28
132657	EVR.L	454.3	09:03:29	30/04/2018 09:03:29
132658	EXPN.L	1673.5	09:03:30	30/04/2018 09:03:30
132659	FRES.L	1286	09:03:31	30/04/2018 09:03:31
132660	GFS.L	261.8	09:03:45	30/04/2018 09:03:45
132661	GKN.L	464.95	09:03:46	30/04/2018 09:03:46
132662	GLEN.L	358.15	09:03:47	30/04/2018 09:03:47
132663	GSK.L	1464.4	09:03:47	30/04/2018 09:03:47
132664	HL.L	1784	09:03:48	30/04/2018 09:03:48
132665	HMSO.L	549.4	09:03:49	30/04/2018 09:03:49
132666	HSBA.L	727.8	09:03:55	30/04/2018 09:03:55
132667	IAG.L	631.8	09:03:56	30/04/2018 09:03:56
132668	IAP.L	0	09:03:56	30/04/2018 09:03:56
132669	IHG.L	4603.75	09:03:57	30/04/2018 09:03:57
132670	IMI.L	1101.34	09:03:59	30/04/2018 09:03:59
132671	IMT.L	0	09:04:00	30/04/2018 09:04:00
132672	IPR.L	417.7	09:04:01	30/04/2018 09:04:01
132673	ITRK.L	4940	09:04:02	30/04/2018 09:04:02
132674	ITV.L	149.75	09:04:03	30/04/2018 09:04:03
132675	JMAT.L	3309	09:04:04	30/04/2018 09:04:04
132676	KAZ.L	921.2	09:04:05	30/04/2018 09:04:05
132677	KGF.L	305.2	09:04:06	30/04/2018 09:04:06
132678	LAND.L	994.95	09:04:07	30/04/2018 09:04:07
132679	LGEN.L	271.5	09:04:08	30/04/2018 09:04:08
132680	LLOY.L	65.03	09:04:09	30/04/2018 09:04:09
132681	MGGT.L	477.6	09:04:10	30/04/2018 09:04:10
132682	MKS.L	284.9	09:04:11	30/04/2018 09:04:11
132683	MRW.L	238.1	09:04:12	30/04/2018 09:04:12
132684	NG.L	844.6	09:04:13	30/04/2018 09:04:13
132685	NXT.L	5272	09:04:14	30/04/2018 09:04:14
132686	OML.L	257.8	09:04:15	30/04/2018 09:04:15
132687	PFC.L	610.2	09:04:16	30/04/2018 09:04:16
132688	POLY.L	731.8	09:04:17	30/04/2018 09:04:17
132689	PRU.L	1885.5	09:04:18	30/04/2018 09:04:18
132690	PSON.L	831.2	09:04:19	30/04/2018 09:04:19
132691	RB.L	5683	09:04:20	30/04/2018 09:04:20
132692	RBS.L	270.26	09:04:21	30/04/2018 09:04:21
132693	RDSB.L	2590	09:04:22	30/04/2018 09:04:22
132694	REL.L	1565.5	09:04:23	30/04/2018 09:04:23
132695	REX.L	0	09:04:24	30/04/2018 09:04:24
132696	RIO.L	3956	09:04:25	30/04/2018 09:04:25
132697	RR.L	840.8	09:04:25	30/04/2018 09:04:25

Record: 1 of 4587 No Filter Search

The server program was left to run for a while. Notice the ‘4587’ at the bottom – this shows how many entries are in the database (around an hours worth of data)

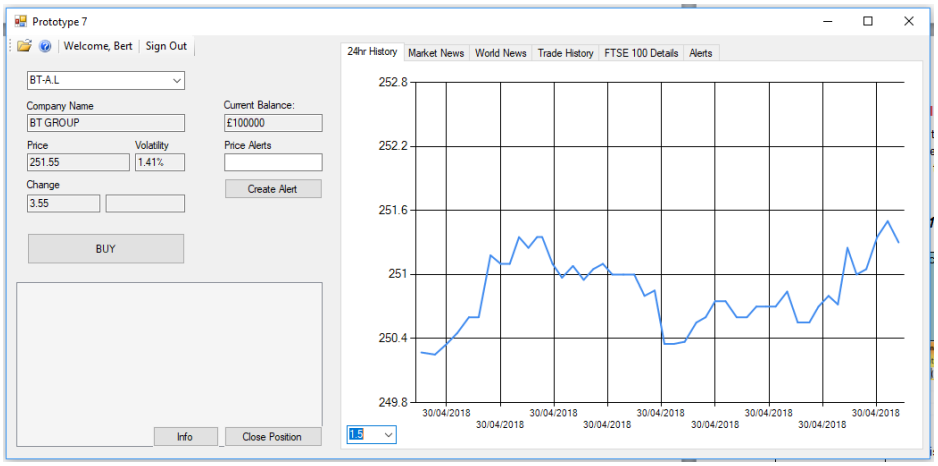
Now, when the main simulation is loaded, the user is able to see graphs:

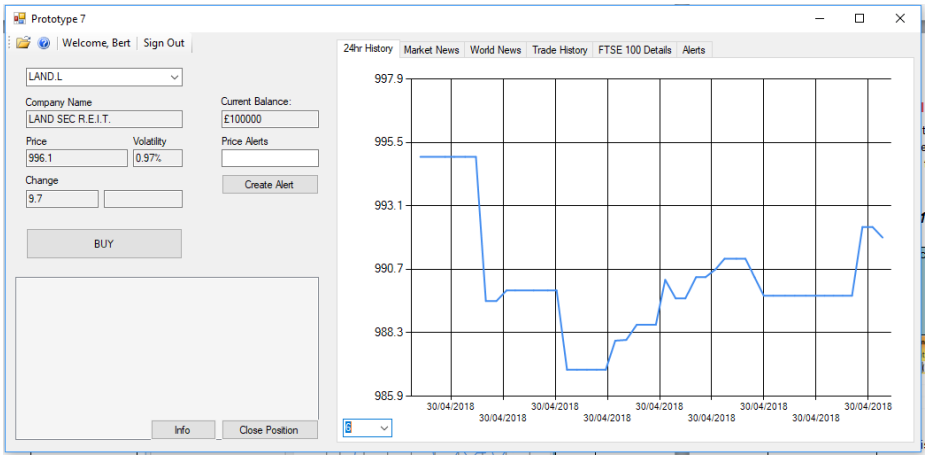

Prototype 7
Welcome, Bert
Sign Out
ANTO.L
Company Name
ANTOFAGASTA
Price
964.4
Volatility
1.08%
Change
10.4
BUY
Info
Close Position
Current Balance:
£100000
Price Alerts
Create Alert
24hr History
Market News
World News
Trade History
FTSE 100 Details
Alerts
971
967.4
963.8
960.2
956.6
953
30/04/2018

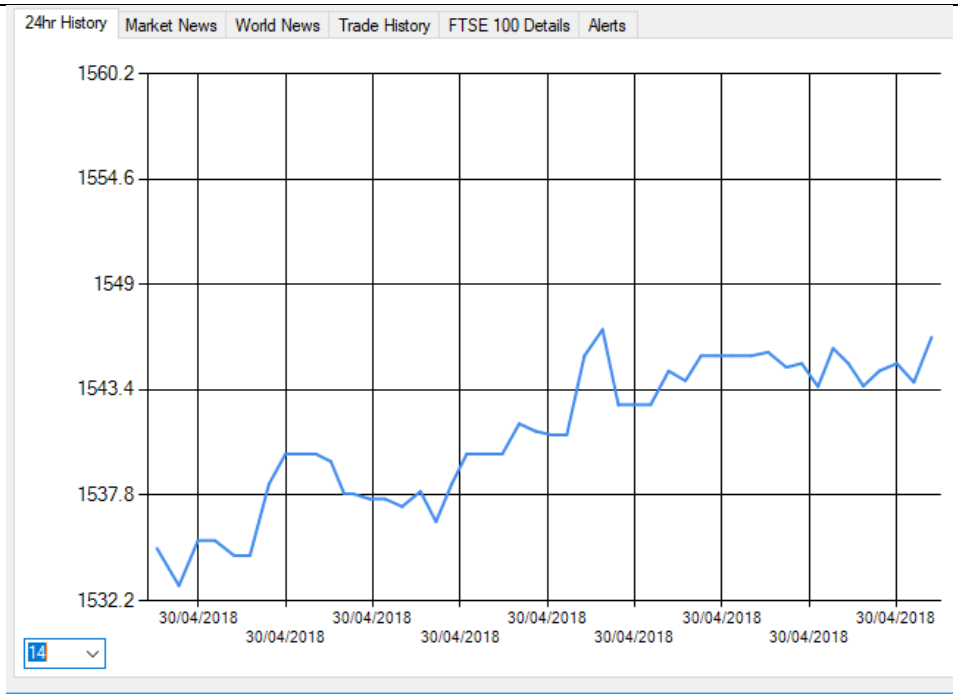


The previous two screenshots show the graphs for Antofagasta and Brit Amer Tobacco.

Below are screenshots showing the graph feature working for various other stocks.

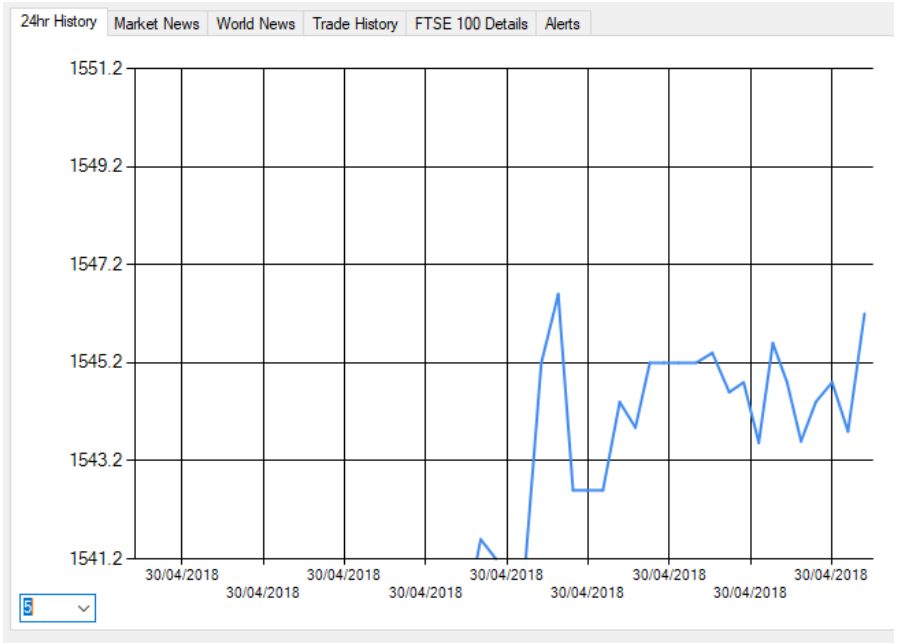






Note that the scale box (A feature tested in Development 1) is different for each screenshot – this is necessary to fit the graph on the the page (otherwise it would be too zoomed in or too zoomed out)

This is demonstrated below, where the same stock as above is shown, but with a smaller scale:

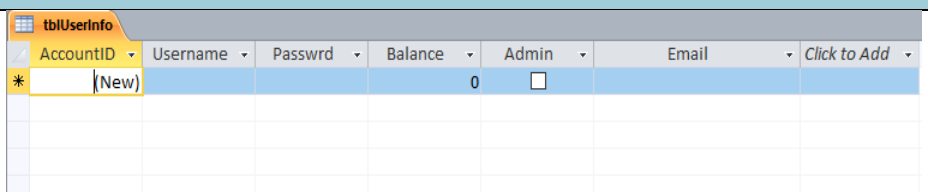


Despite being the same graph, the different scale means the second screenshot is so zoomed in that half the data is off the graph. That is the reason why the scale box is different for every screenshot.

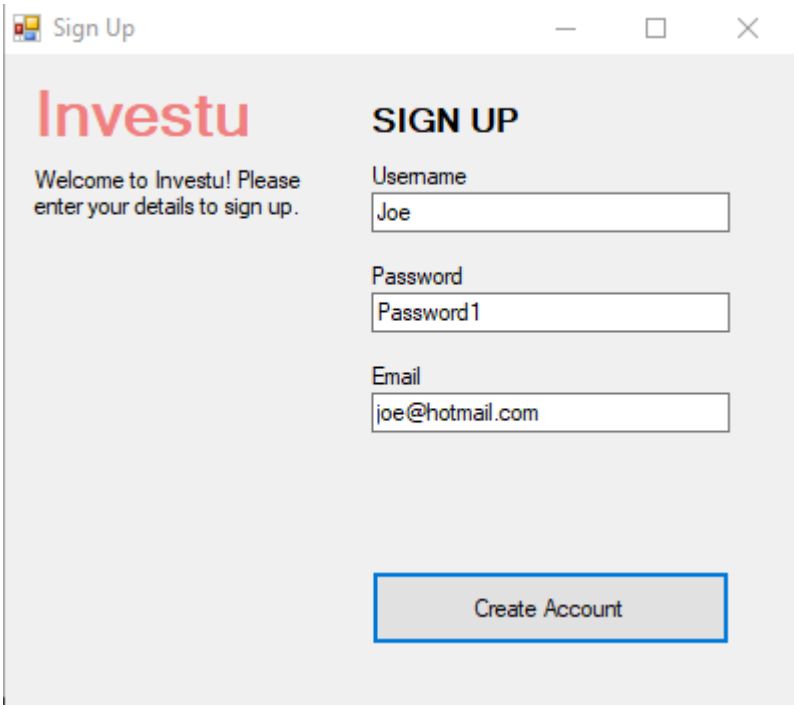
Testing 1 – Investu Simulation – Development 2

Every sub-routine has been tested individually to ensure it works independently, however these tests have not been shown. The tests displayed are tests that show multiple sub-routines and functions working together to produce the desired outcome.

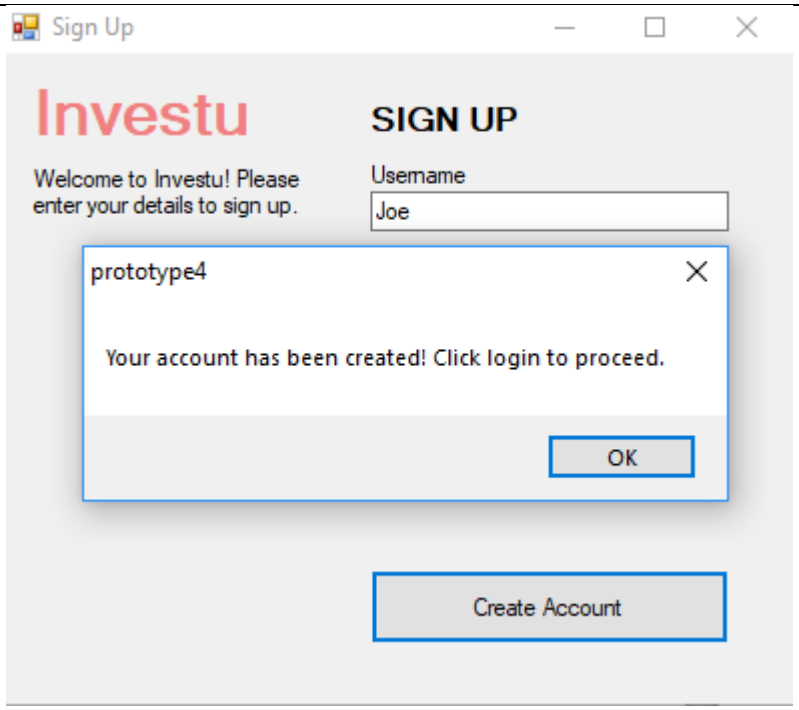
SignUpForm Testing 1

<table>
<tr><th>Test Objective</th><th>Evidence</th><th>Objectives met?</th></tr>
<tr><td>Show that sign up attempt will be accepted and an account will be written to the database, when tried with standard inputs</td><td>


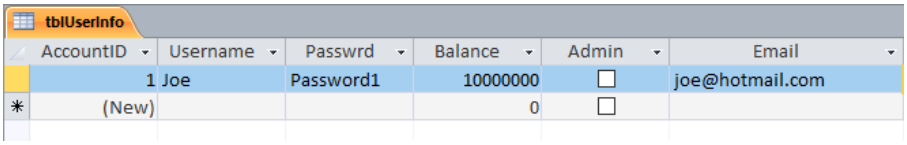
Above is a screenshot of the table ‘tblUserInfo’ in the database, to show that no accounts currently exist



We know that the regular expressions used require passwords to consist of 8 or more characters, with one upper case character and one number. Therefore the password ‘Password1’ would be a valid password.</td><td></td></tr>
</table>



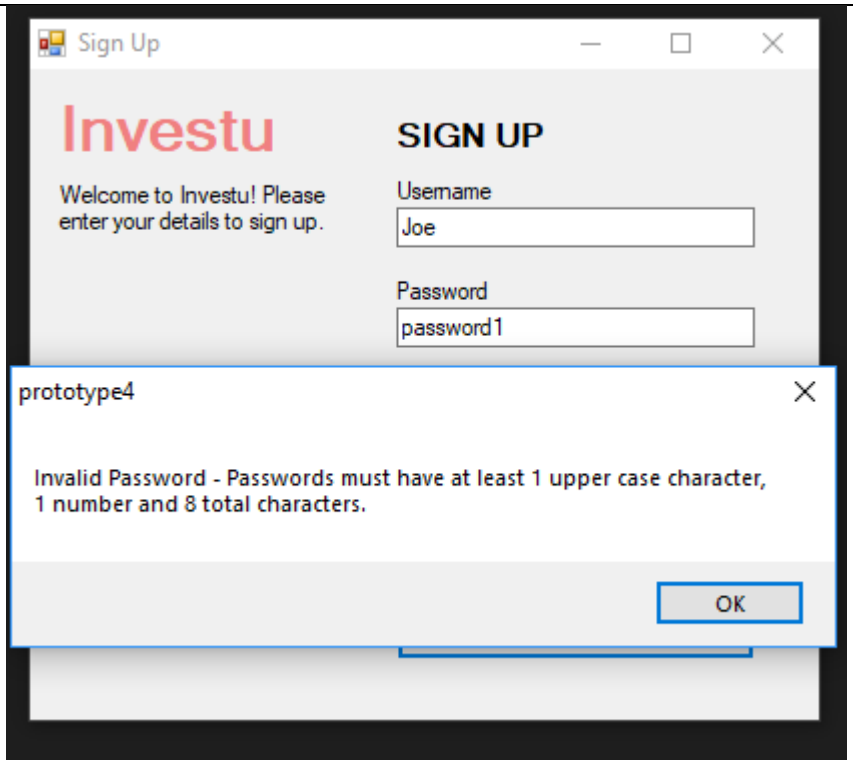
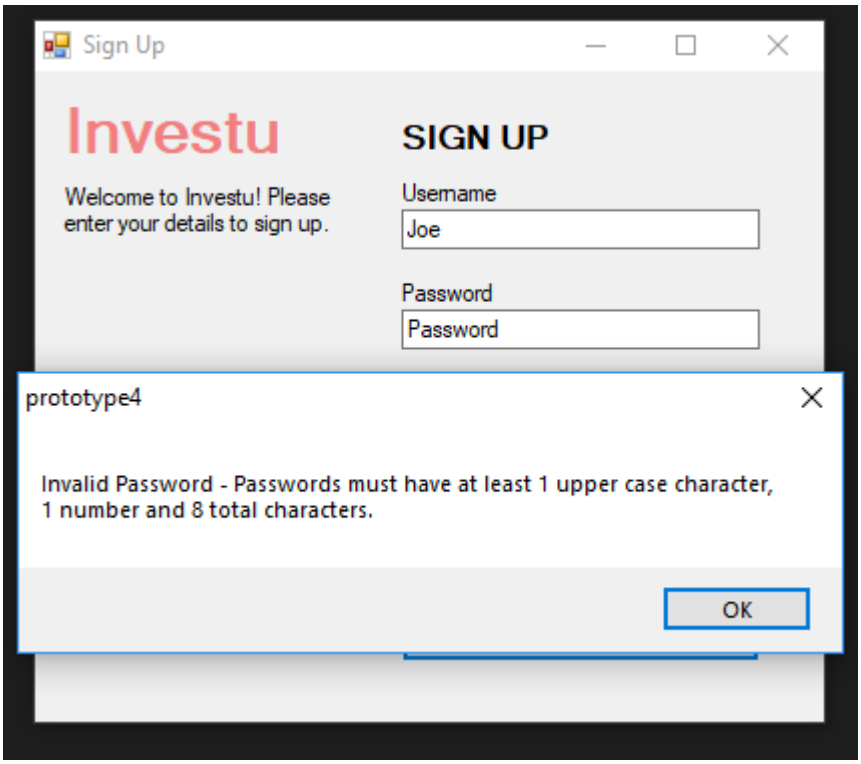
After pressing the ‘Create Account’ button, a confirmation message appears to confirm that the account creation was successful. The account should now appear in the database in the table ‘tblUserInfo’



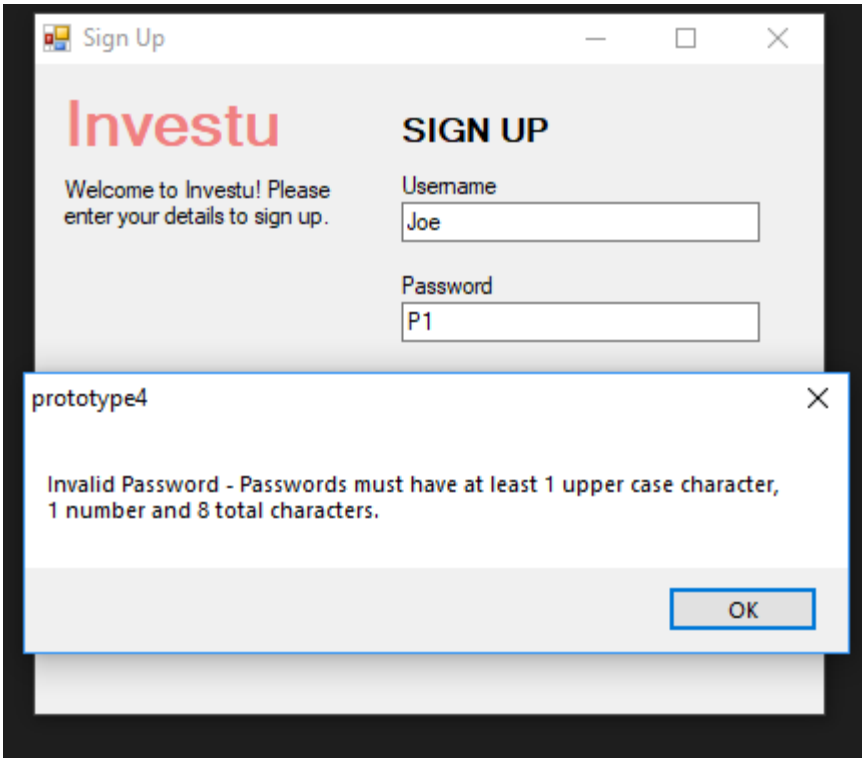
tblUserInfo

AccountID	Username	Passwrd	Balance	Admin	Email
1	Joe	Password1	10000000	☐	joe@hotmail.com
(New)			0	☐	

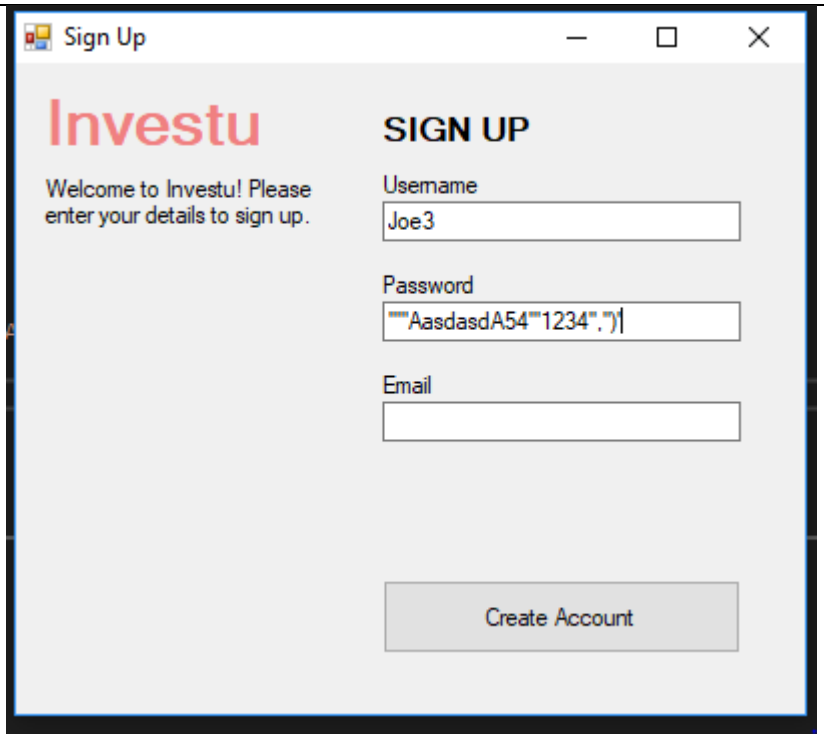
And here we see that the account has successfully been written into the database. This indicates that the requirements set out in the test description have been met and the test has therefore been passed.

Show sign up is rejected when password does not meet the parameters of the regular expressions	 This password is 9 characters with a number, but does not contain any uppercase characters. 	

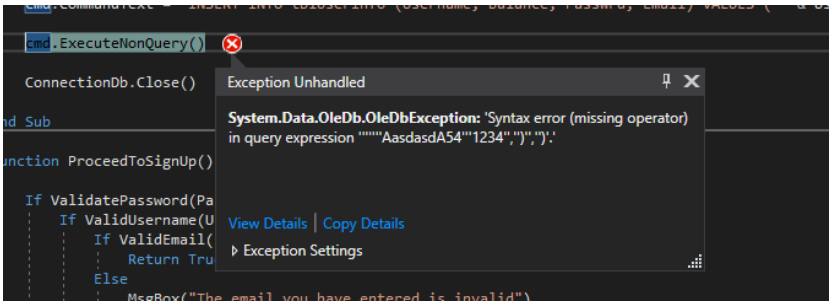
This password does not contain a number, but has 8 characters and an uppercase letter



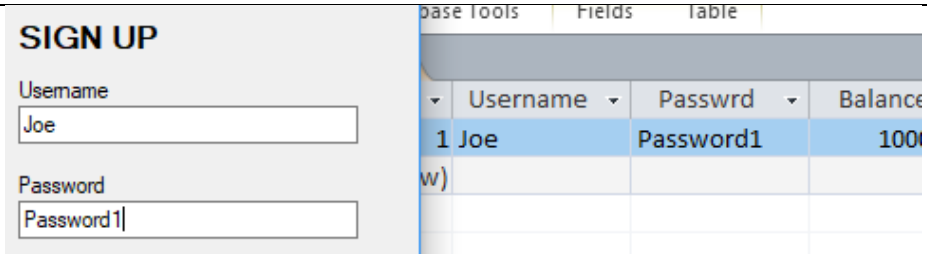
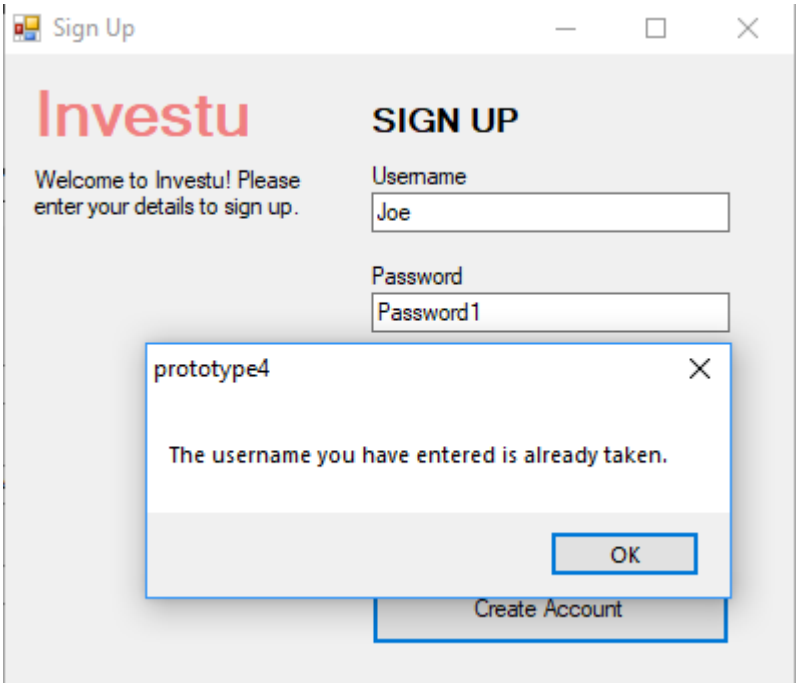
This password contains an uppercase letter and a number, but does not have 8 characters.

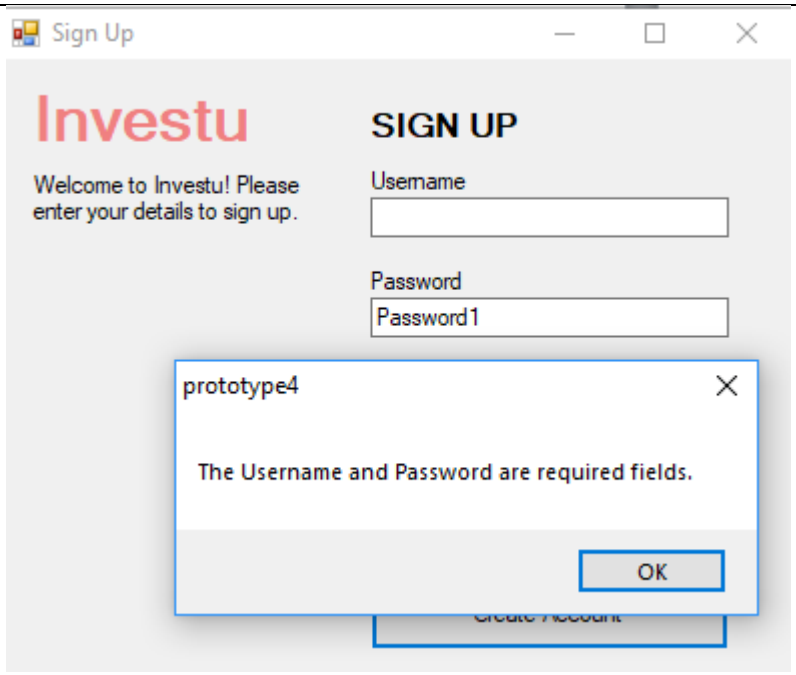
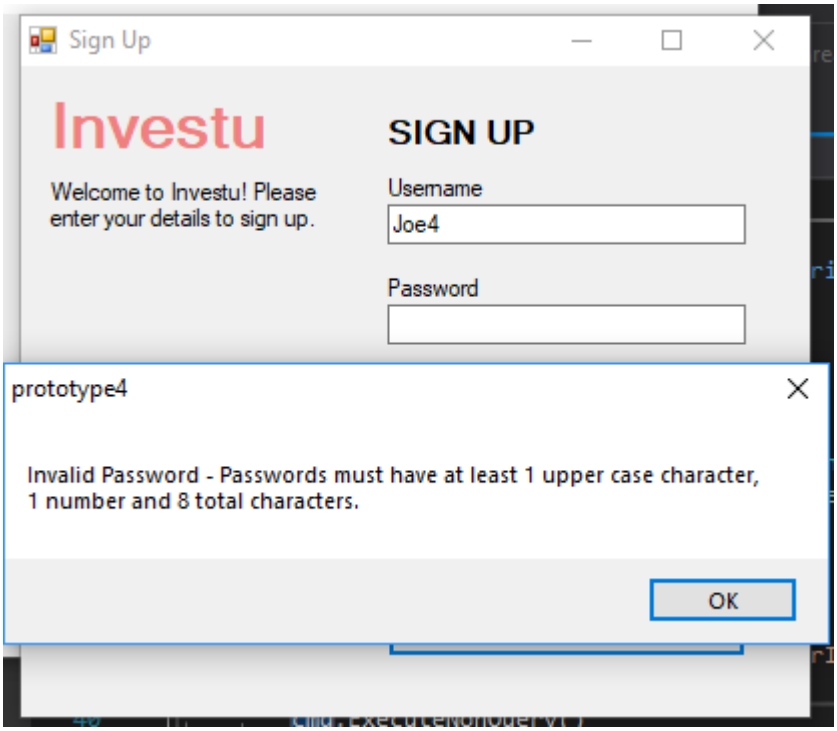


Although the first three password tests were successful, a problem arises when this string of random letters and characters is entered.

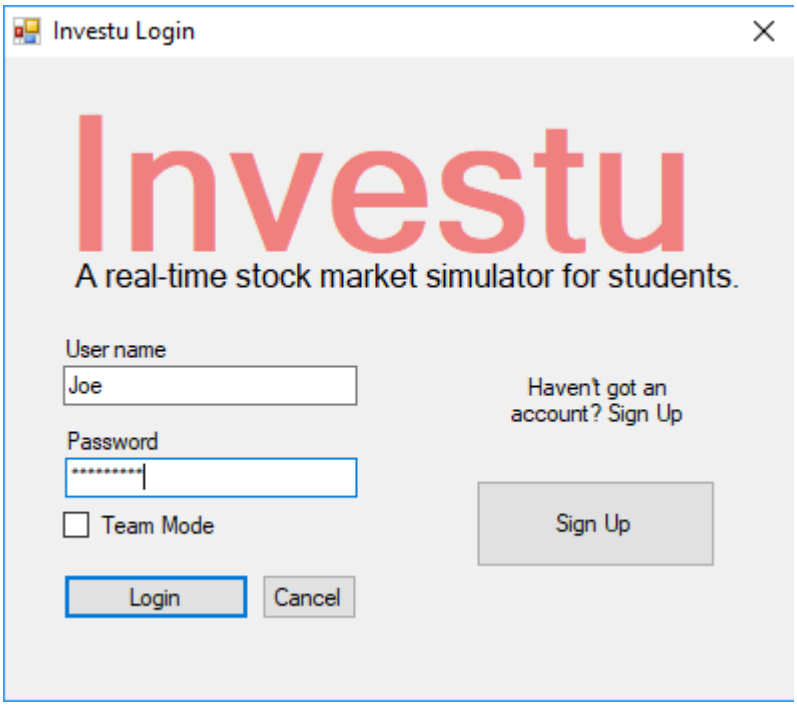
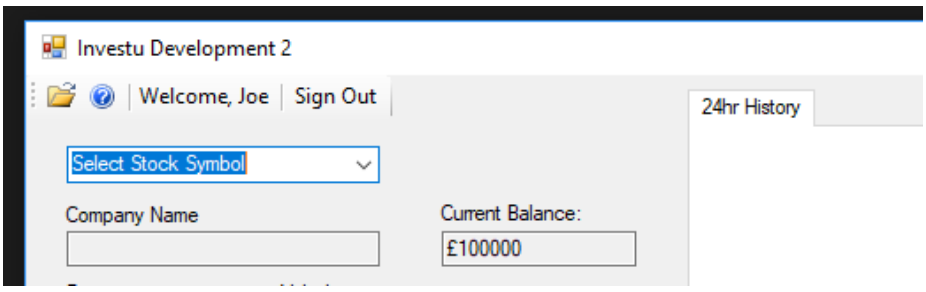


This is due to the fact the input contains apostrophes and brackets, which effectively alter the SQL statement. The regular expressions will need to be changed so that the user is not able to change the SQL, else the simulation will be vulnerable to SQL injections, allowing users to change information within the database.

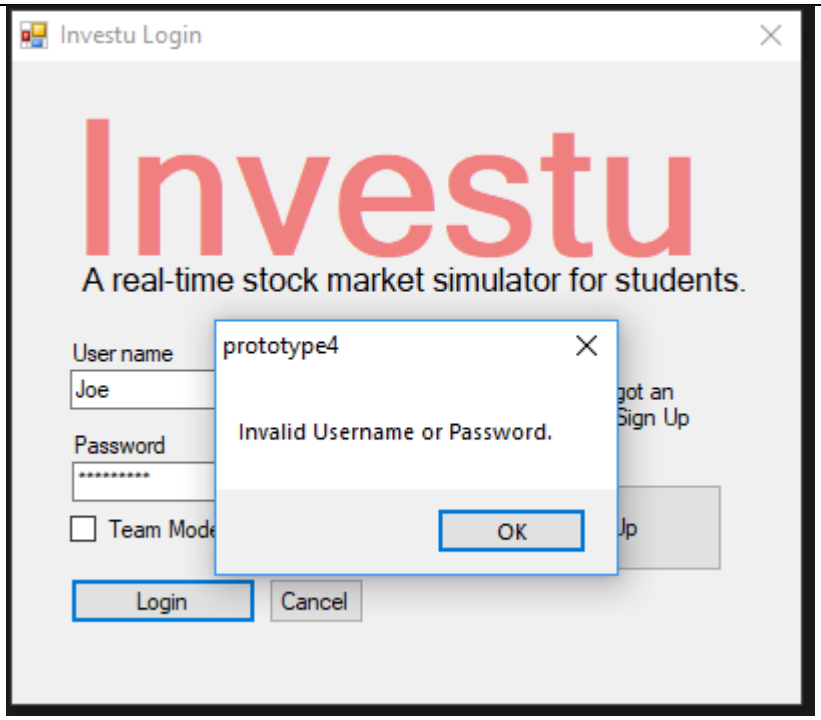
Show that accounts can only be created if they have a unique username	 Attempting to sign up with the username 'Joe' while there is already an account in the database with the same name, produces this error:  This indicates that the requirements set out in the test description have been met and the test has therefore been passed.	

Show that validation for blank entries in required fields works correctly	  This indicates that the requirements set out in the test description have been met and the test has therefore been passed.	

LoginForm Testing 1

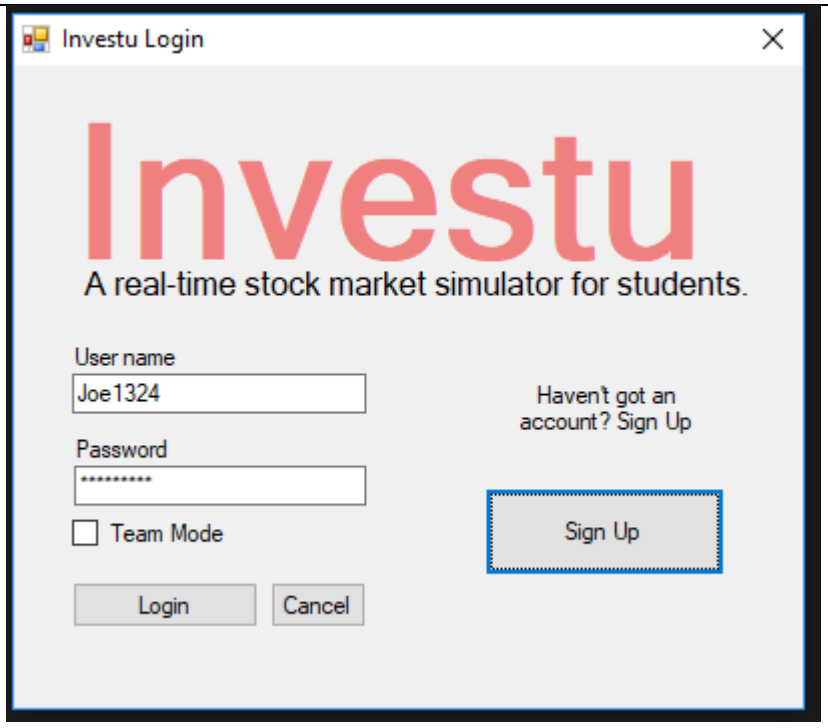
Test Objective	Evidence	Objectives met?
Show that accounts that exist in the database can be signed, resulting in the loading of the main form with the users information	From the testing of 'SignUpForm' we know that there exists an account in the database with the username 'Joe' and password 'Password1'.  Attempting to sign in with this information gives the following result:  The simulation loads, with the username displayed, and the correct balance	

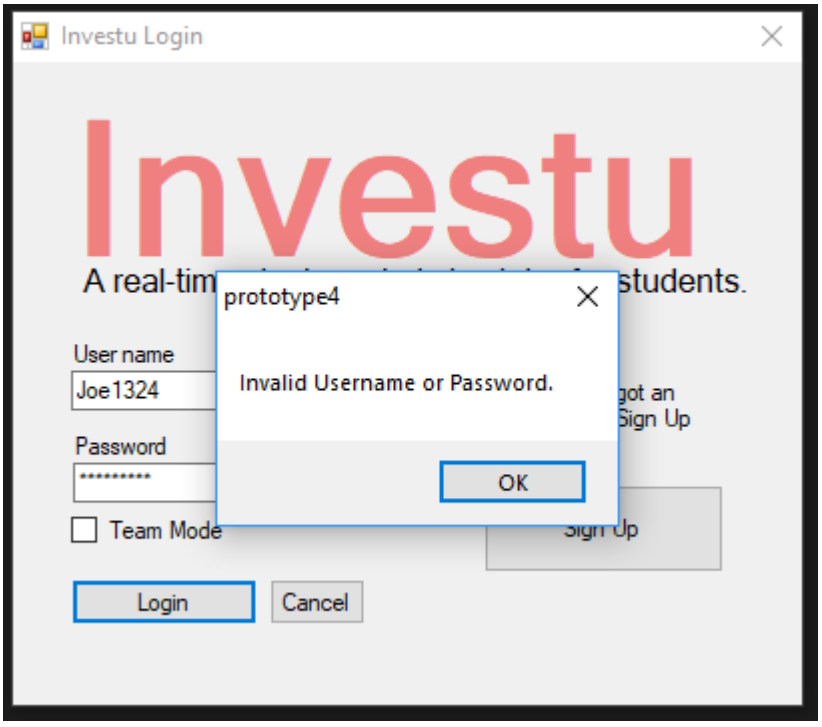
	in the ‘Current balance’ display box. This indicates that the requirements set out in the test description have been met and the test has therefore been passed.	
Show that correct username incorrect password combination results in a failed login attempt	 The password attempted here is ‘Password2’, whereas the correct password for the account name with ‘Joe’ is ‘Password1’.	

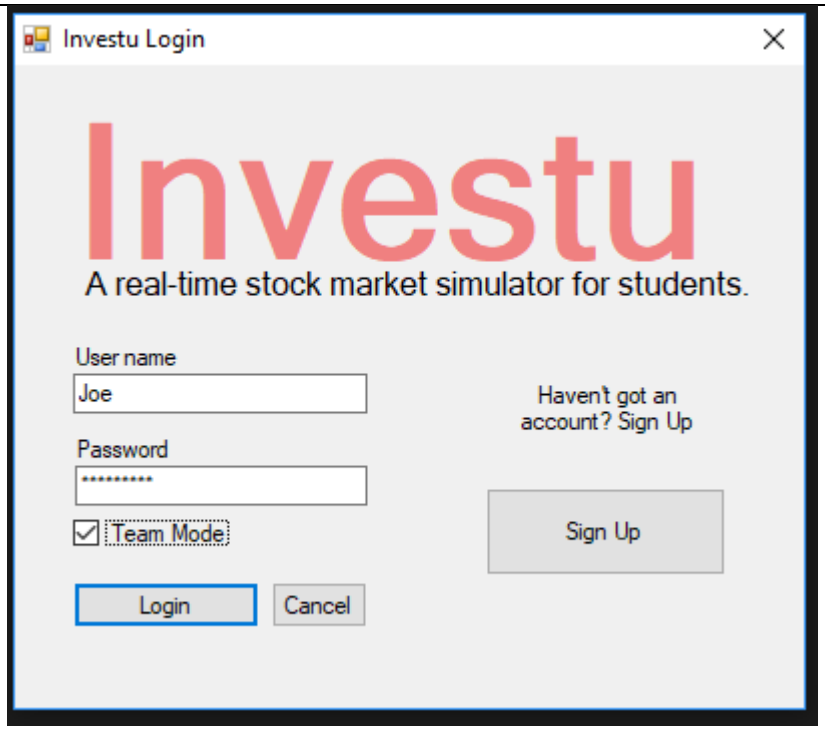
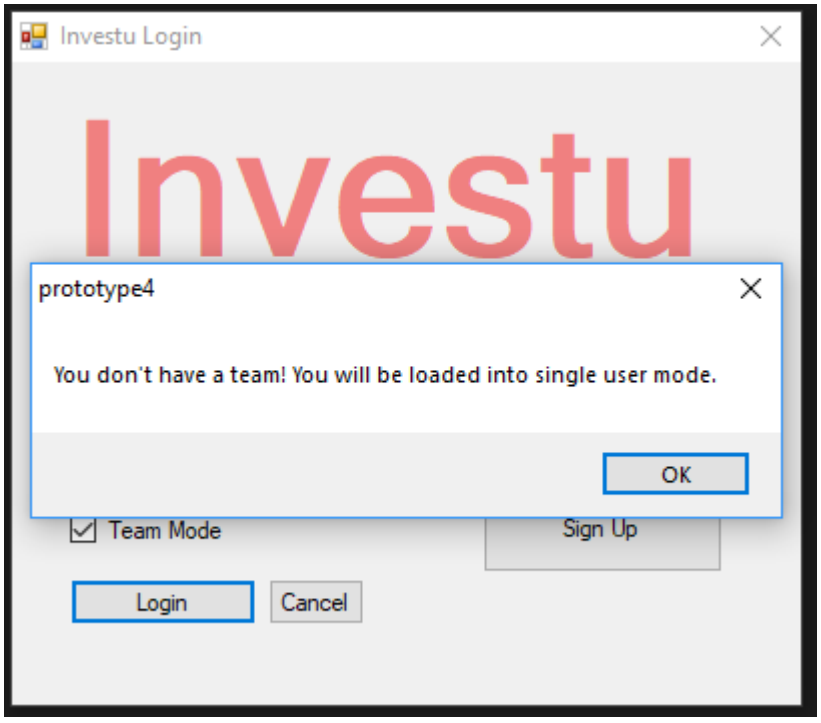


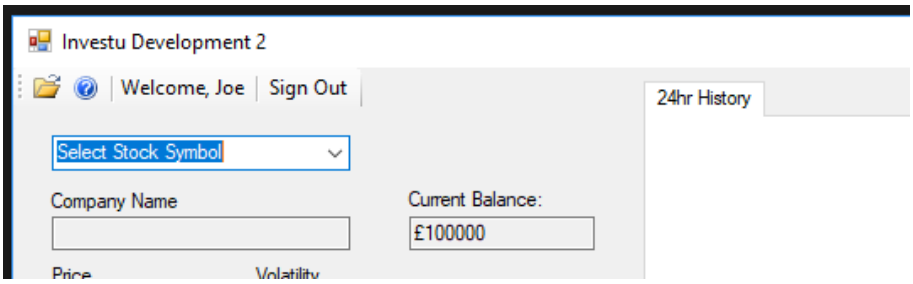
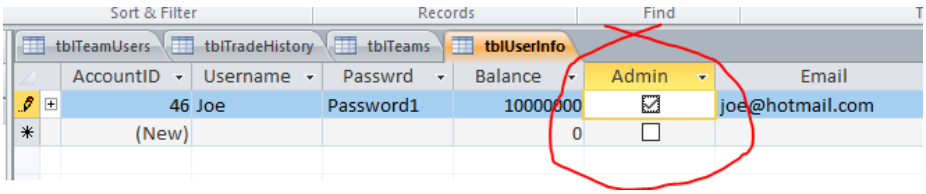
This produces this dialogue box.

A similar result is produced when the user tries to sign into an account with a username that does not exist in the database:


Investu Login
Investu
A real-time stock market simulator for students.
User name
Joe1324
Password
Team Mode
Login
Cancel
Haven't got an account? Sign Up
Sign Up


Investu Login
Investu
prototype4
Invalid Username or Password.
OK
User name
Joe1324
Password
Team Mode
Login
Cancel
Sign Up

Show that users who are not in a team cannot log into their account in team mode	 To load the simulation in team mode, the 'Team Mode' check box is checked.  Because the user is not in a team, they are prompted with a dialogue box.	

	 The user is then signed into their personal account. This indicates that the requirements set out in the test description have been met and the test has therefore been passed.	
Show that a designated admin account will be signed into admin mode.	The ability to log into an admin account exists in Development 2, however the functionality has not been added yet, as that will come later in the development. Therefore, it is expected that the admin login will result in the loading of a blank form. To create an admin, we will manually change the value of 'Admin' in the database.  Then, by loading signing into the account with the name 'Joe', the following is displayed.	

	 In future developments this form can be developmed to act as an admin display page, with functionality allowing for the creation, manipulation and analysis of teams. This indicates that the requirements set out in the test description have been met and the test has therefore been passed.	
Show that users can sign into team accounts	Because the ability for teams to be created and users to join teams does not exist yet, we will manually edit the database to test this functionality. We will add this user to a team: tblTeams tblUserInfo AccountID Username Passwrd Balance Admin 46 Joe Password1 10000000 (New) 0 First, a team is created – notice the balance of '12345', as opposed to the balance of the account 'Joe', which is 1000000.	

tblTeams

TeamID	TeamName	Balance	TeamCode	Cl
23	TestTeam	12345	TEST1	
(New)		0		

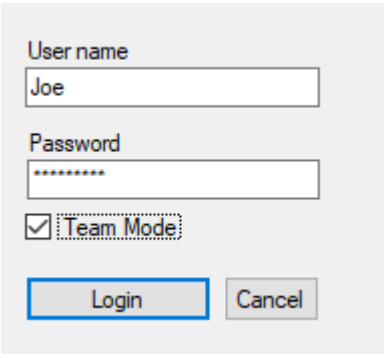
Next, the user ‘Joe’ is added to the team by creating a new entry in ‘tblTeamUsers’

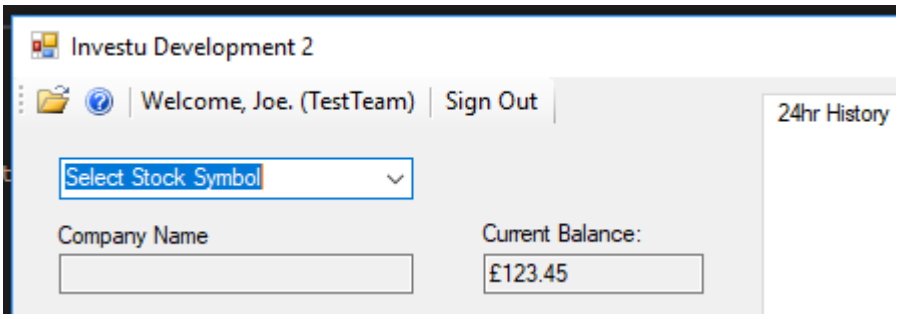
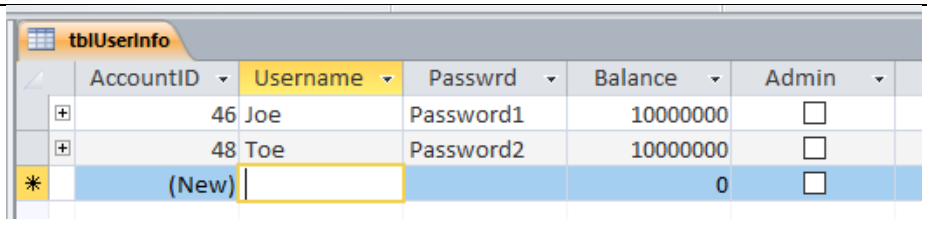
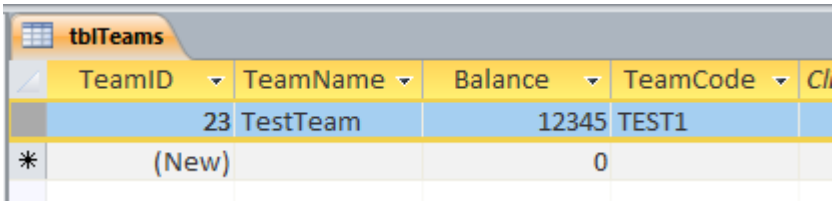
tblUserInfo | tblTeams | tblTeamUsers

TeamUsersI	AccountID	TeamID	Clic
14	46	23	
(New)			

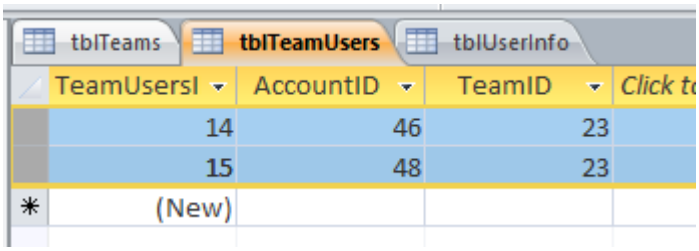
This is a link table, linking the team to the user.

Now that the user is in a team, the user should be able to log into their team account.

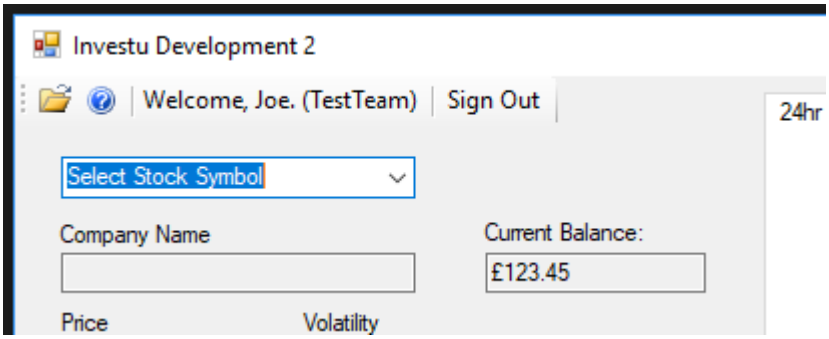


	This produces the following display:  This time, the user's team name is displayed along side their name, as well as the team balance – notice the balance is taken from 'tblTeams' instead of 'tblUserInfo', as this is the team account. This indicates that the requirements set out in the test description have been met and the test has therefore been passed.	
Show that users of the same team can log into the same team account	 Two users have been created, 'Joe' and 'Toe'. They will be added to the following team:  The ID's for the team and users are added to the link table to associate the	

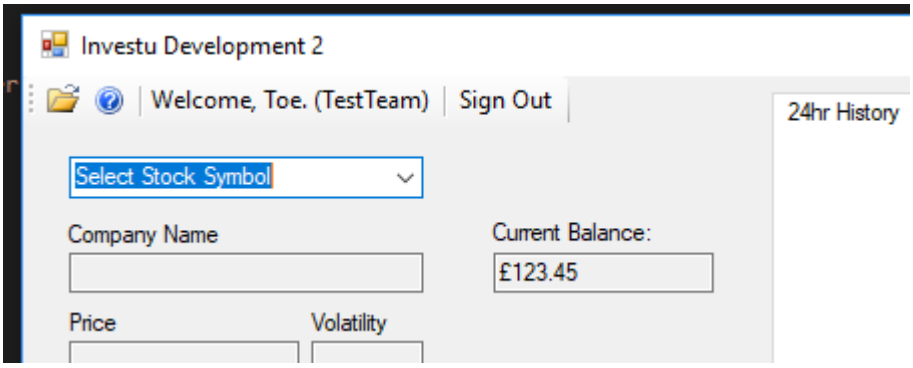
accounts with the team:



When signing into the account with the account named ‘Joe’, the following dialogue appears:



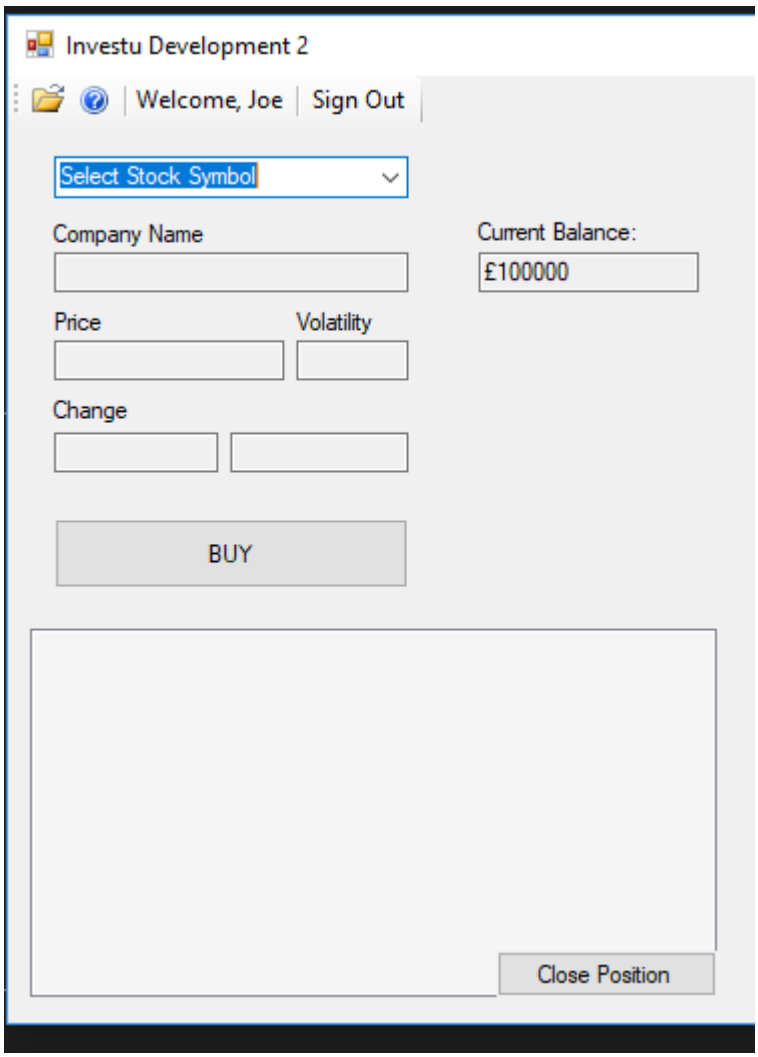
When signing into the account named ‘Toe’ with team mode set to true, the following dialogue appears:

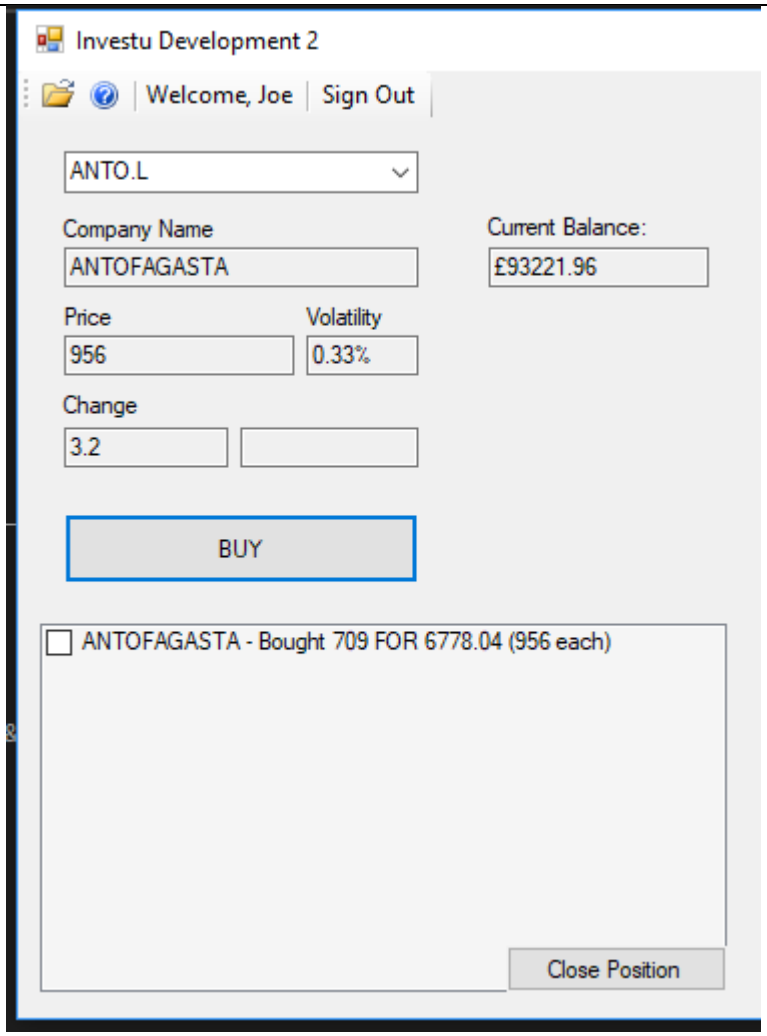


Notice that both accounts are have ‘(TeamTest)’ – the name of their team, after their name. Furthermore, the balance value for both accounts reflects that of the team.

I believe this is sufficient evidence to conclude that the requirements set out in the test description have been met and the test has therefore been passed.

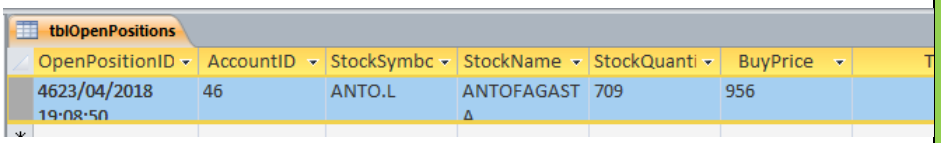
MainForm Testing 1

Test Objective	Evidence	Objectives met?
Prove that a user can sign in to their account, execute a trade, and then log out, and their progress will be saved	Initially, the balance of the account 'Joe' is £100,000 and the portfolio box is empty:  Then, a trade is executed on the account:	



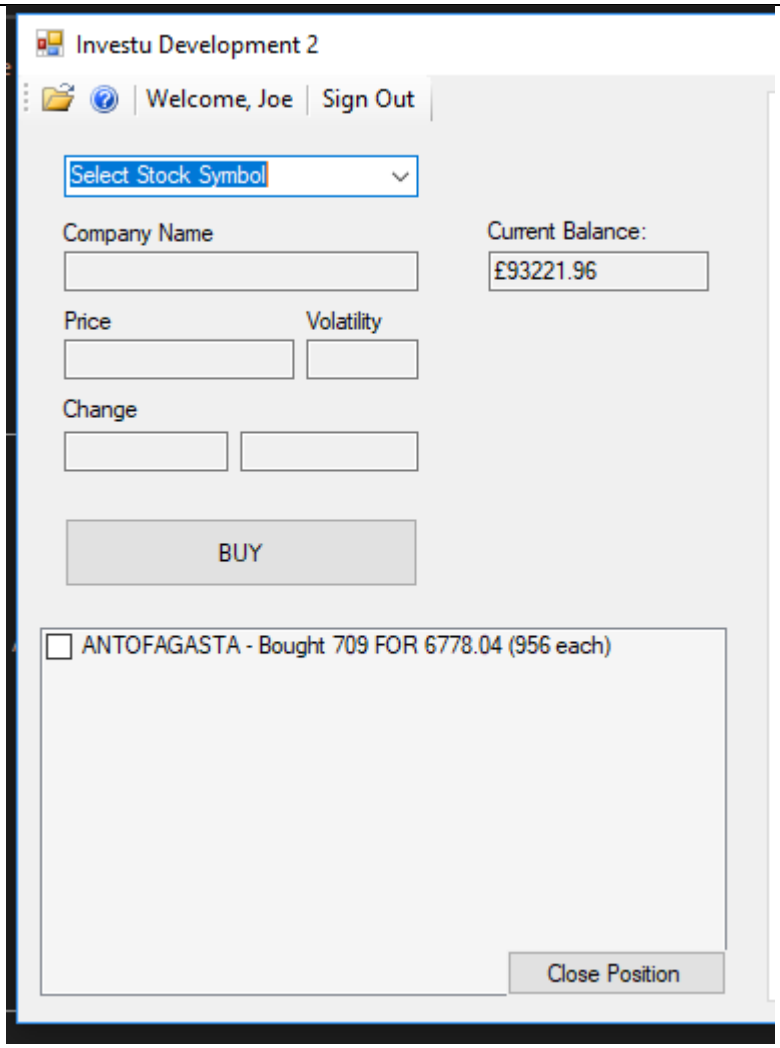
Now, the simulation is closed

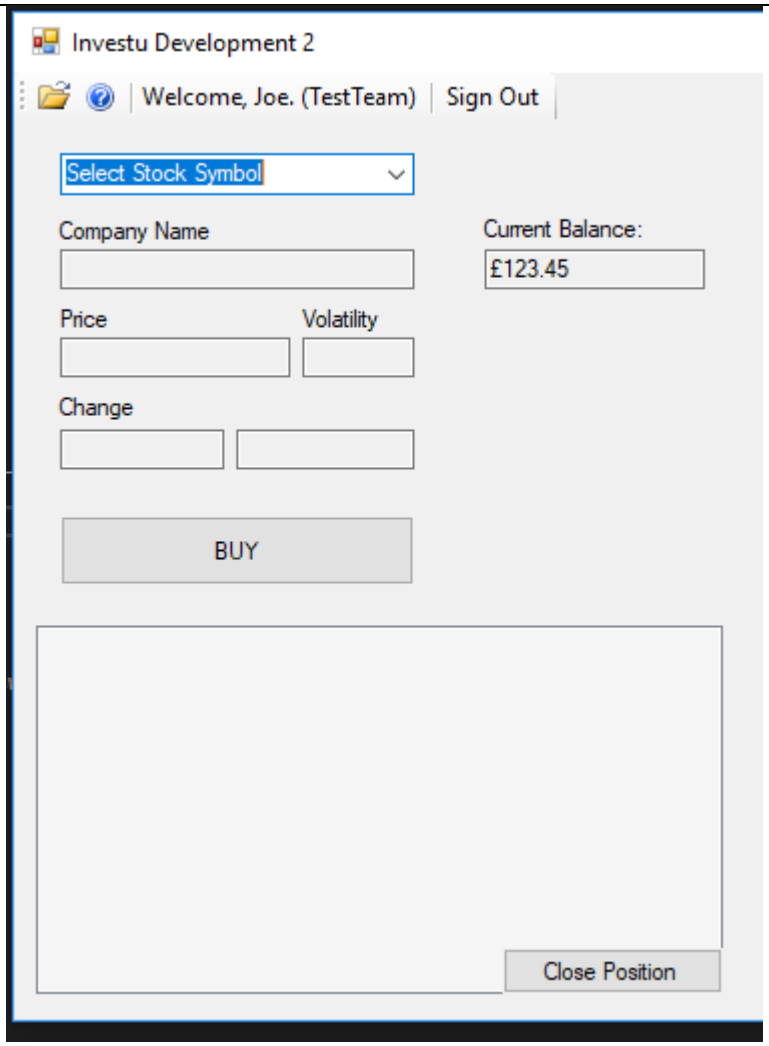
In the database, we can see that a position has been opened, with the account ID 46, meaning that this trade was made by the account 'Joe'.



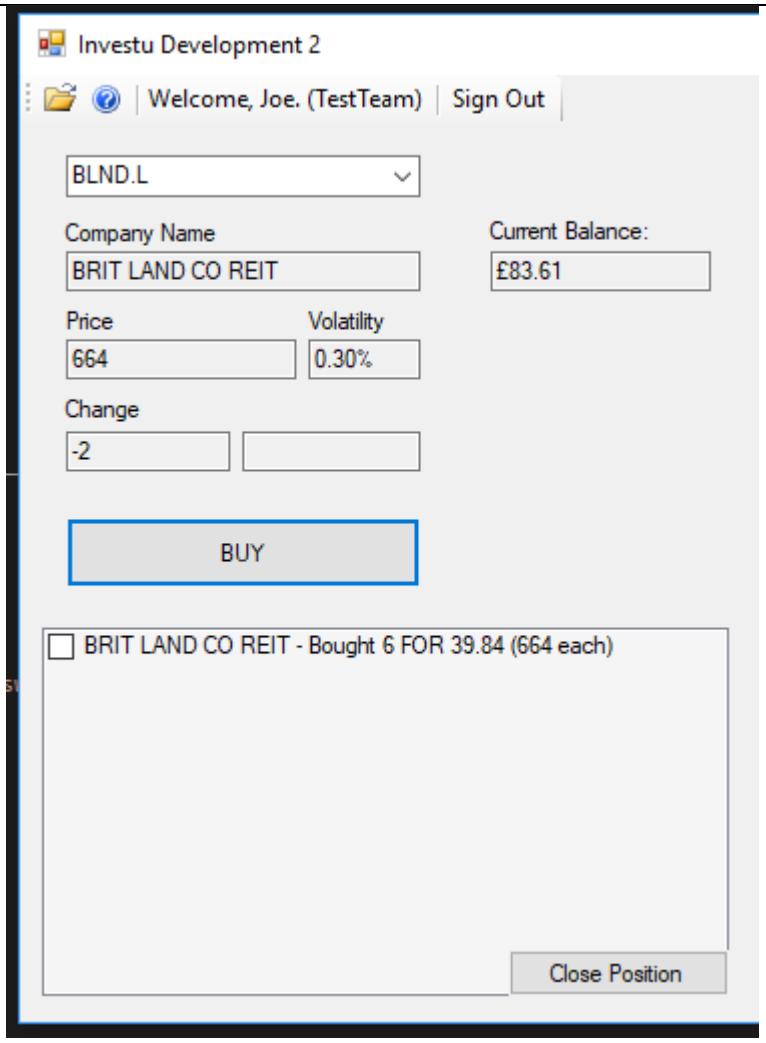
This means that the trade has been successfully written into the database.

Upon loading the simulation again with the same account, the following is displayed:

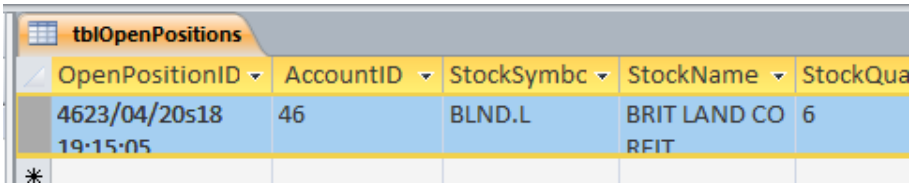
	 This indicates that the requirements set out in the test description have been met and the test has therefore been passed.	
Prove that a user can log into a team account, execute a trade, and then log out, and their progress will be saved on their team account	As shown in a previous test, the account 'Joe' has been made a member of the team 'TestTeam'. Signing into this account shows the following display:	



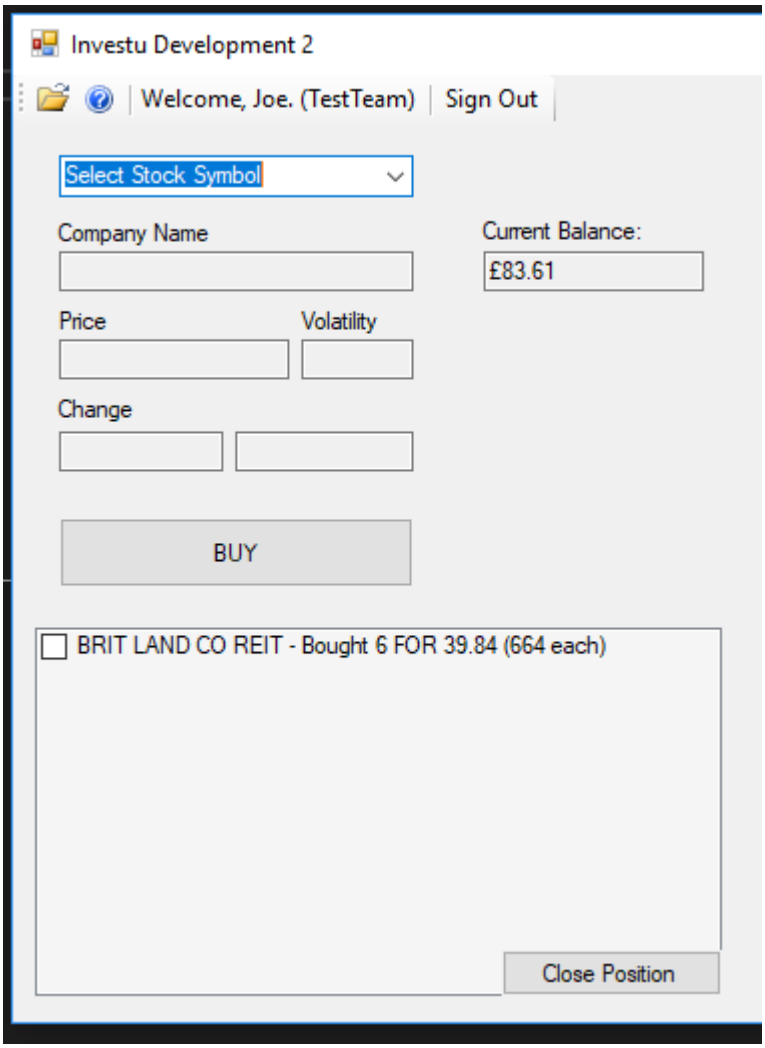
After executing a trade, the following display is shown:

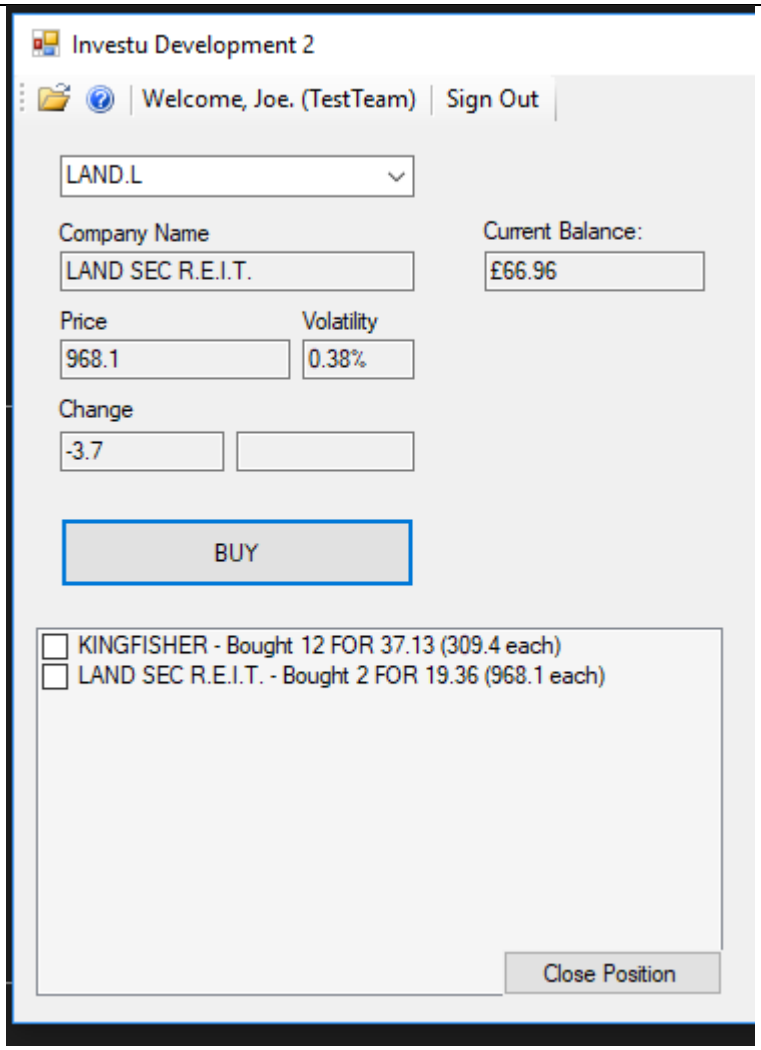


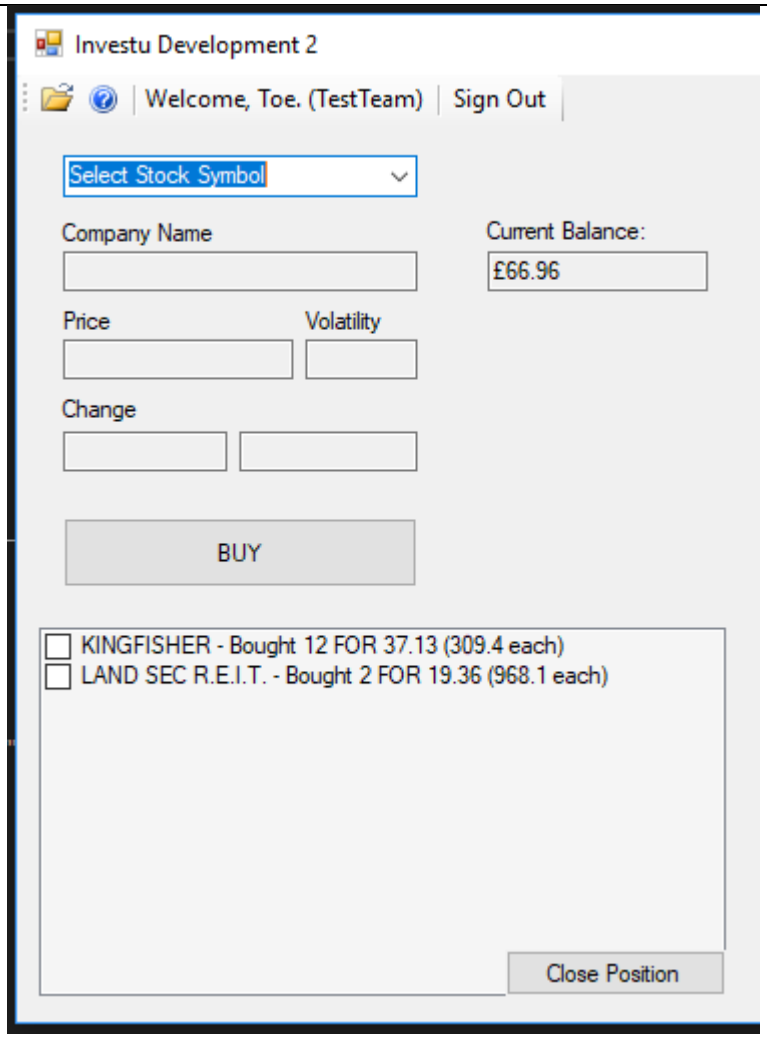
The simulation is now closed. The database shows as follows:



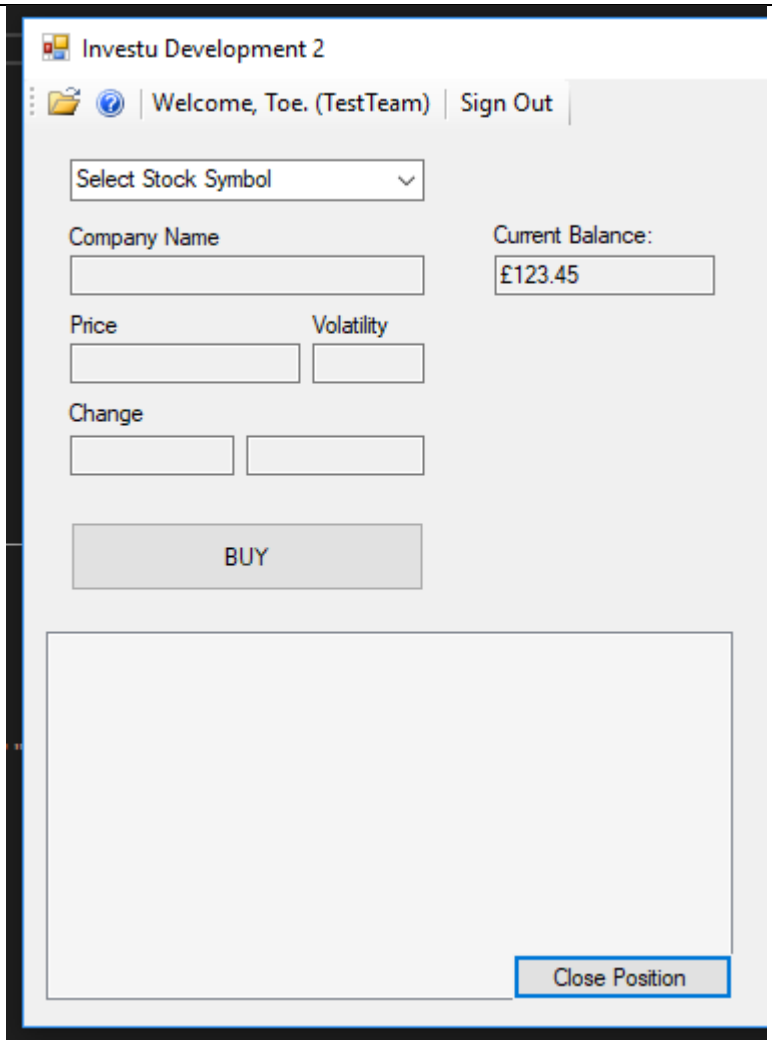
Upon reloading the account ‘Joe’ in team mode, the following display is shown:

	 This indicates that the requirements set out in the test description have been met and the test has therefore been passed.	
Show that members of the same team can both open and close positions, and the changes will be visible to	The two accounts used for this test will be 'Joe' and 'Toe', which were added to the team 'TestTeam' in a previous test. 'Joe' is loaded into team mode and two trades are executed:	

other users who are logged into the team account	 Then, the simulation is closed and then reloaded. The account 'Toe' is signed into, with team mode set to true:	



The positions are then both closed by the account ‘Toe’:



Notice that the balance returns to £123.45, the same as before 'Joe' executed the two trades.

This indicates that the requirements set out in the test description have been met and the test has therefore been passed.

Testing Findings - Investu Development 2

From the tests carried out on Development 2 we can see that the code runs largely as expected – the goals determined at the start have been met, and the code appears to run cleanly. The errors that were encountered appear to be at the sign up phase.

Specifically, these errors are:

1) When creating an account, the user has unrestricted access to special characters, including the apostrophe and brackets. This leads to corruption of the SQL Insert statement responsible for inserting the account into the database. Furthermore, this vulnerability leaves the simulation sucesptible to SQL injections, which would allow the user to edit the database. To fix this, the regular expression responsible for special characters must be changed to stop users using these special characters.

Fixing Errors - Investu Development 2

```
Function ValidatePassword(ByVal Password As String, Optional ByVal MinLength As
Integer = 8, Optional ByVal NumUpper As Integer = 1, Optional ByVal NumLower As
Integer = 1, Optional ByVal NumNumbers As Integer = 1, Optional ByVal NumSpecial As
Integer = 0) As Boolean

        Dim UpperCase As New System.Text.RegularExpressions.Regex("\p{Lu}")
        Dim LowerCase As New System.Text.RegularExpressions.Regex("[a-z]")
        Dim Numbers As New System.Text.RegularExpressions.Regex("[0-9]")

        Dim Specials As New System.Text.RegularExpressions.Regex("[^a-zA-Z0-9]")

        If Len(Password) < MinLength Then Return False

        If UpperCase.Matches(Password).Count < NumUpper Then Return False
        If LowerCase.Matches(Password).Count < NumLower Then Return False
        If Numbers.Matches(Password).Count < NumNumbers Then Return False
        If Specials.Matches(Password).Count < NumSpecial Then Return Fals

        Return True
End Function
```

These regular expressions determine the composition of the password. By changing this line:

```
If Specials.Matches(Password).Count < NumSpecial Then Return False
```

To:

```
If Specials.Matches(Password).Count >= 1 Then Return False
```

Then we can successfully prevent users from using special characters in their password.

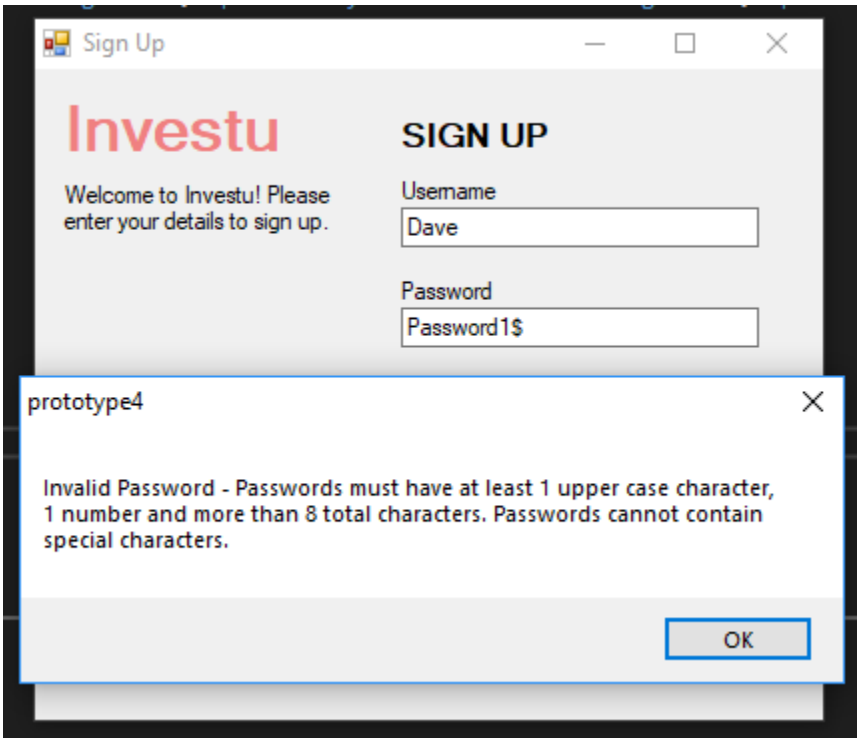
The same must also be applied to the username, for the same reasons. To do this, a new function is required, to check the number of special characters in the username.

```
Function ContainsSpecialChars(ByVal NewUsername As String)

        Dim Specials As New System.Text.RegularExpressions.Regex("[^a-zA-Z0-9]")

        If Specials.Matches(NewUsername).Count = 0 Then
            Return False
        End If

        Return True

    End Function
```

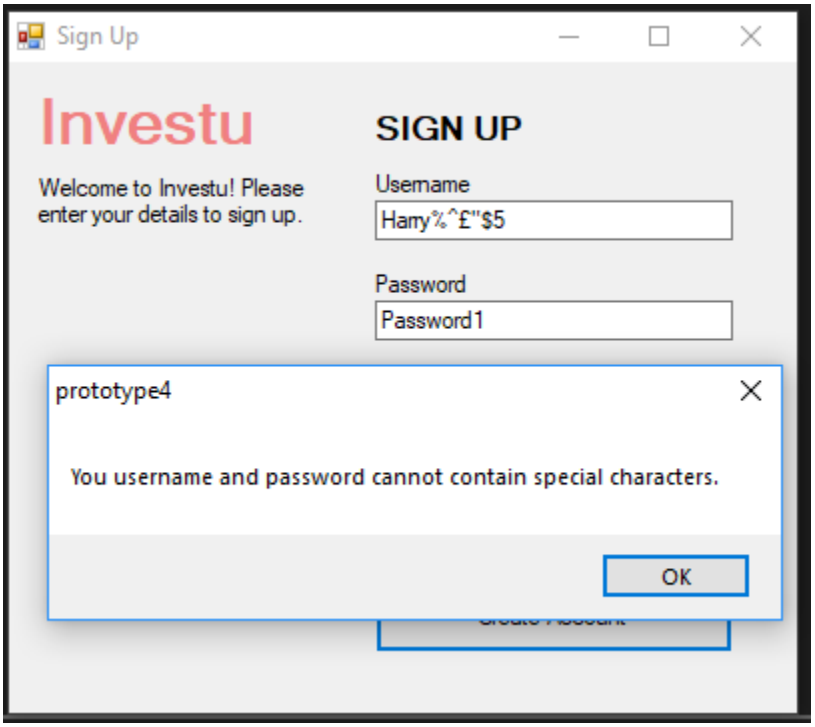
This function can then be called in ‘ValidateUsername’, using a conditional.

Testing 2 - Investu Development 2

Trying to create an account with a special character in the password now returns an error:



Trying a username with a special character, but a valid password, returns this error:



Feedback #4 – Client – Investu Development 2

The end of the second development represents a fairly significant milestone in the creation of Investu, and as such it seems appropriate to co-ordinate again with the client to ensure that the simulation is headed in the expected direction, and the developments made are along the correct lines.

The following is an email exchange with the client.

Me

“Mr Butterworth,
Attached is instructions for accessing the simulation. Could you please have a look at the program and let me know your thoughts regarding the progress so far? If you have any suggestions let me know.
Thanks,
Joe”

Client

“”Joe,
I’ve had a look at the program – very impressed! Looks really good so far. I made myself an account and had a play. I showed a class of mine during a lesson and they seemed very interested. I have been making some trades here and there and I’m very pleased that progress is saved now so that I can close it and then come back later at the same position I left. I’m going to open some high-risk high-reward positions and leave them for a few days – I’ll let you know how I get on.
Keep up the good work! Message me if you have any more questions or progress.
Thanks,
Gazza B.”

Final Conclusion – Investu Development 2

At the end of Development 1, a list of goals were laid out for Development 2. Those goals were as follows:

"In the next development of the program the aim is to successfully connect the database to the program, allowing for the storage of data related to the FTSE 100 and the user account. This will allow for a login system, as well as a teams system and the ability to save information relating to these two features. Furthermore, once the database is connected, it will be possible to begin collecting stock market data and storing it. This will require a small, additional program, that will operate on a server 24/7 in order to collect data."

Having finished Development 2, it is now clear that those goals have been successfully completed, as shown through the analysis of the code and the testing of the development previously.

Furthermore, in the analysis of the simulation, a feature list and an objectives list was created using feedback from the client and the user. Now that Development 2 has been implemented, we can see how many of these criteria have been met, and analyse the goals for the third and final development of the program.

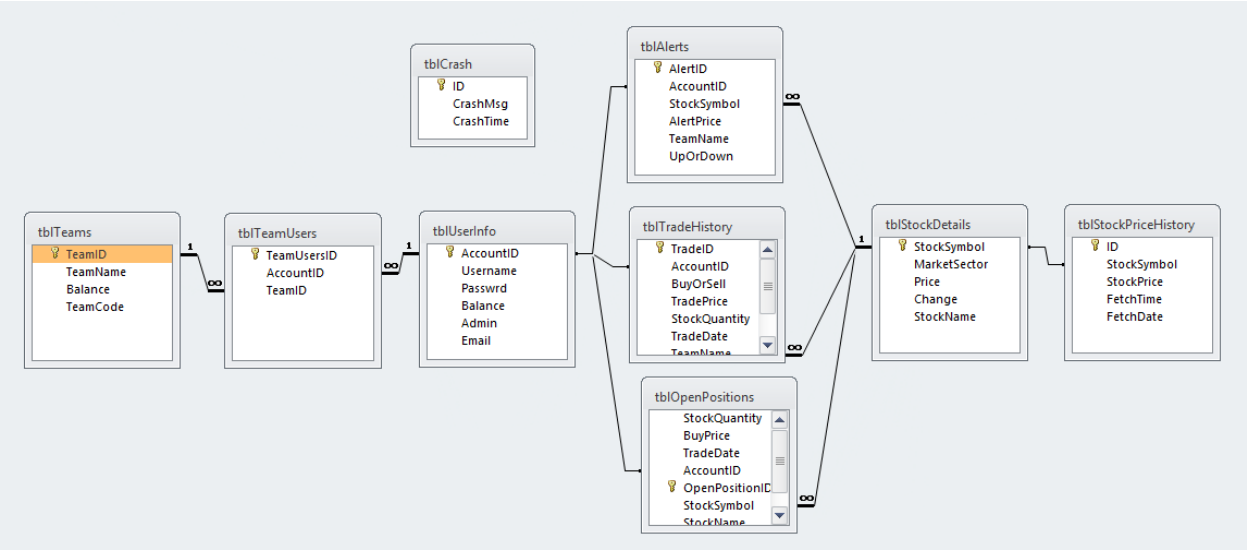
(The items highlighted in green have been successfully implemented into Development 1, as shown in the Development 1 testing phase in the previous section. The items highlighted in teal have been successfully implemented into Development 2, as shown in the development 1 testing phase in the previous section

- Ability to create and login to accounts (client)
- Ability to join and trade on a team account (client)
- Ability for users to be designated as admins (client)
- Ability for account progress on team and personal accounts to be saved between sessions (inferred from client and user)
- Ability for admins to view teams list (inferred from client)
- Ability for admins to view team details and progress (inferred from client)
- Ability to view real-time information for all FTSE 100 stocks (client and user)
- Ability to select an amount of stocks and buy that amount using an up to date virtual balance, at real current price (client/user)
- Display for all stocks currently held in portfolio (client/SIC)
- Ability to sell stocks in portfolio at real current price (client)
- Graphs to display current day price trends of all stocks (user)
- Graph to show all time price changes of all stocks (inferred from user)
- Ability to create price alerts and be notified when stock reaches current price (inferred from user)
- Interface allowing users to see all current alerts on their account (inferred from user)
- Interface allowing user to see entire trade history (SIC)
- Interface allowing user to see all stocks in the FTSE 100 in a single screen, with details such as price (SIC)
-
- Notes section displayed in trade history and portfolio with reasons for trade decision (user)

Development 3

Database 3 – Investu – Development 3

The final version of the database has a slightly more complex structure compared to the previous developments. This version incorporates all of the tables needed to create all of the features outlined in the analysis.



The final additions to the database are as follows:

- tblTradeHistory – A table that keeps a log of all executed trades made in the simulation; both buys and sales.
- tblAlerts – A table to keep track of Alerts that have been made
- tblStockDetails – A table consisting of all details of every stock in the FTSE100, including name, price, symbol etc.

AdminView - Investu – Development 3

A key feature of Investu is the AdminView form – admin accounts are for teachers, and will allow for the creation, monitoring, manipulation and analysis of teams by teachers. This will be an important aspect in making Investu successful, as it will allow for teachers to monitor students and hone their trading skills.

AdminView_Load – AdminView – Investu Development 3

```
Public Class AdminView

    Dim AccessDatabaseConnection As String = MainForm.AccessDatabaseConnection

    Private Sub AdminView_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load

        FetchTeams()
End Sub
```

On the loading of the admin view form, only one routine is called – 'FetchTeams'. This will access the database and fetch a list of every possible team.

FetchTeams – AdminView – Investu Development 3

```
    Sub FetchTeams()

        Dim TeamID As Integer
        Dim TeamInfo As String


        TeamInfoCheckedListbox.Items.Clear()
        TeamIdCheckedListBox.Items.Clear()

        Dim ConnectionDb As New OleDbConnection(AccessDatabaseConnection)
        If ConnectionDb.State = ConnectionState.Closed Then ConnectionDb.Open()
        Dim cmd As OleDbCommand = ConnectionDb.CreateCommand
        cmd.CommandText = "SELECT TeamID, TeamName, Balance, TeamCode FROM tblTeams"

        Dim SQLReply As OleDbDataReader = cmd.ExecuteReader

        For Each Record In SQLReply
            TeamID = Record.item("TeamID")
            TeamIdCheckedListBox.Items.Add(TeamID)

            TeamInfo = Record.item("TeamName") & " - " & Record.Item("TeamCode") & "
- " & Record.item("Balance")
            TeamInfoCheckedListbox.Items.Add(TeamInfo)
        Next

        ConnectionDb.Close()
    End Sub
```

FetchTeams works by querying the ‘tblTeams’ table in the database for the ID of each team and then displaying their information into display boxes.

```
        Dim TeamID As Integer
        Dim TeamInfo As String
```

The two variables declared are ‘TeamID’ and ‘TeamInfo’. ‘TeamID’ is for the ID of the team, and ‘TeamInfo’ is for all other information such as name and balance. ‘TeamID’ needs to be its own separate value with its own separate display box so that it can be used to query the database later. If it was concatenated with the other information, then it would need to be extracted from that string later when it was needed. Therefore, it has its own variable and display box.

```
        TeamInfoCheckedListbox.Items.Clear()
        TeamIdCheckedListBox.Items.Clear()
```

The visual display is cleared before the new information is written, to avoid duplication of data.

```
        Dim ConnectionDb As New OleDbConnection(AccessDatabaseConnection)
        If ConnectionDb.State = ConnectionState.Closed Then ConnectionDb.Open()
        Dim cmd As OleDbCommand = ConnectionDb.CreateCommand
```

A standard database connection is initiated, using the access database connection from ‘MainForm’. The connection to the database is opened by querying the connection state and then using ‘ConnectionDB.open’ if the state is found to be closed.

```
cmd.CommandText = "SELECT TeamID, TeamName, Balance, TeamCode FROM tblTeams"
```

The SQL statement is a simple Select command, that extracts 4 values from ‘tblTeams’.

```
For Each Record In SQLReply
            TeamID = Record.item("TeamID")
            TeamIdCheckedListBox.Items.Add(TeamID)

            TeamInfo = Record.item("TeamName") & " - " & Record.Item("TeamCode") & " - "
& Record.item("Balance")
            TeamInfoCheckedListbox.Items.Add(TeamInfo)
        Next
```

For every record returned by the SQL Select statement, a series of actions are carried out. First, the ID of the team is assigned to the variable ‘TeamID’, defined earlier. Then, this ID value is added to its own display box.

Then, the variable ‘TeamInfo’ is assigned the other information. This string is simply for display purposes and won’t be used further. This is then added to a display box located horizontally along from the ID display box, so that the information from the team relevant to the ID is displayed horizontally from the ID. This will allow the admin to clearly see which information relates to which teams.

AdminView_Load – AdminView – Investu Development 3

```
    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
        If ValidateInputs(TeamNameBox.Text, TeamCodeBox.Text) Then
            CreateNewTeam(TeamNameBox.Text, TeamCodeBox.Text)
        Else
            MsgBox("There was an error creating the new team. Your team name may
already be taken or you have entered invalid information.")
        End If
    End Sub
```

One of the capabilities of the AdminViewForm is creating new teams. This sequence is started when the admin clicks the 'Create Team' button. This takes the values of a series of input boxes and uses the data to create a new team.

```
If ValidateInputs(TeamNameBox.Text, TeamCodeBox.Text) Then

        Else
            MsgBox("There was an error creating the new team. Your team name may already
be taken or you have entered invalid information.")
        End If
```

'ValidateInputs' is a sub-routine that ensures the data for the new team is in the correct format and will not cause clashes in the database. If this check is failed, a message box prompt will appear informing the user.

```
CreateNewTeam(TeamNameBox.Text, TeamCodeBox.Text)
```

If the validation function in the conditional returns true, then the sub-routine 'CreateNewTeam' is called, which takes two arguments – the value of 'TeamNameBox' and 'TeamCodeBox'.

AdminView_Load – AdminView – Investu Development 3

```
Function ValidateInputs(ByVal NewTeamName As String, ByVal NewTeamCode As String)

        If TeamNameBox.Text = "" Or TeamCodeBox.Text = "" Or BalanceBox.Text = "" Then

            Return False
        Else

            Dim ConnectionDb As New OleDbConnection(AccessDatabaseConnection)
            If ConnectionDb.State = ConnectionState.Closed Then ConnectionDb.Open()

            Dim cmd As OleDbCommand = ConnectionDb.CreateCommand

            cmd.CommandText = "SELECT TeamName, TeamCode FROM tblTeams"

            Dim SQLReply As OleDbDataReader = cmd.ExecuteReader

            For Each Record In SQLReply

                If Record.item("TeamName") = NewTeamName Or Record.item("TeamCode") =
NewTeamCode Then
                    Return False
                End If
            Next

        End If
        Return True
    End Function
```

```
If TeamNameBox.Text = "" Or TeamCodeBox.Text = "" Or BalanceBox.Text = "" Then

            Return False
        Else
            ….
        End If
```

The first validation check ensures that no input boxes are blank. If one is found to be blank, then the function returns false.

```
Dim ConnectionDb As New OleDbConnection(AccessDatabaseConnection)
            If ConnectionDb.State = ConnectionState.Closed Then ConnectionDb.Open()

            Dim cmd As OleDbCommand = ConnectionDb.CreateCommand

            cmd.CommandText = "SELECT TeamName, TeamCode FROM tblTeams"
```

If all of the input the boxes is not empty, then a connection is initialised.

```
SELECT TeamName, TeamCode FROM tblTeams
```

The SQL statement simply selects every ‘TeamName’ and ‘TeamCode’ from the ‘tblTeams’ table.

```
For Each Record In SQLReply

                If Record.item("TeamName") = NewTeamName Or Record.item("TeamCode") =
NewTeamCode Then
                    Return False
                End If
Next
```

Then, a For-Loop loops through all of the team information returned and compares it to the data passed to the function. If any data is found to match, it means that there is a collision and the code or name has already been taken. This causes the function to return false.

CreateNewTeam – AdminView – Investu Development 3

```
Sub CreateNewTeam(ByVal NewTeamName As String, ByVal NewTeamCode As String)

        Dim Balance As Integer = BalanceBox.Text * 100

        Dim ConnectionDb As New OleDbConnection(AccessDatabaseConnection)
        If ConnectionDb.State = ConnectionState.Closed Then ConnectionDb.Open()

        Dim cmd As OleDbCommand = ConnectionDb.CreateCommand

        cmd.CommandText = "INSERT INTO tblTeams (TeamName, TeamCode, Balance) VALUES
('" & NewTeamName & "','" & NewTeamCode & "','" & Balance & "')"
        cmd.ExecuteNonQuery()

        ConnectionDb.Close()

        MsgBox("A new team with the name " & NewTeamName & " and team code " &
NewTeamCode & " has been created.")

        FetchTeams()

    End Sub
```

One of the most important features of the AdminView form is the ability for teachers to create teams for their students. This is done through the 'CreateNewTeam' sub-routine.

```
Dim Balance As Integer = BalanceBox.Text * 100
```

There is a section of the admin view form that has input boxes for the user to input information for a new team. One of these boxes will take the value for the new teams balance.

```
        Dim ConnectionDb As New OleDbConnection(AccessDatabaseConnection)
        If ConnectionDb.State = ConnectionState.Closed Then ConnectionDb.Open()

        Dim cmd As OleDbCommand = ConnectionDb.CreateCommand
```

A standard database connection is initialised in order to insert the new team information into the database.

```
cmd.CommandText = "INSERT INTO tblTeams (TeamName, TeamCode, Balance) VALUES ('" &
NewTeamName & "','" & NewTeamCode & "','" & Balance & "')"
```

The Insert statement takes 3 values and inserts them into 'tblTeams'. These values are:

- TeamName – The name of the team that will be displayed to users
- TeamCode – The code needed by students to join teams. This is a 5 digit code that is effectively a password, so that only the students with the code can join the team.
- Balance – The teams starting balance

```
MsgBox("A new team with the name " & NewTeamName & " and team code " & NewTeamCode & "
has been created.")
```

The admin is then informed that their team creation was successful.

Note that the input boxes used are the masked-input version of the textbox control. Each textbox has a mask, that will only permit certain data to be written. This is a similar validation method to regular expressions, without the need for more code. The masks for each box is as follows:

TeamNameBox

Mask – aaaaaaaaaaaaa

This mask means that only alphanumeric characters can be entered into the box, as defined in the VB.NET documentation below:

a	Alphanumeric, optional. If the AsciiOnly property is set to **true**, the only characters it will accept are the ASCII letters a-z and A-Z. This mask element behaves like the "A" element.

TeamCodeBox

Mask – LL000

This mask means that the first two characters must be capitalised letters, and the last three must be numbers. The mask also means that the code must be 5 or less characters long.

The VB.NET documentation for L and 0 is shown below:

L	Letter, required. Restricts input to the ASCII letters a-z and A-Z. This mask element is equivalent to [a-zA-Z] in regular expressions.
0	Digit, required. This element will accept any single digit between 0 and 9.

BalanceBox

Mask – 000000

This mask means that the balance box will only accept numbers between 0 and 9, and a maximum of 6.

0	Digit, required. This element will accept any single digit between 0 and 9.

TeamIDCheckedListBox_ItemCheck – AdminView – Investu Development 3

```
Private Sub TeamIdCheckedListBox_ItemCheck(ByVal sender As Object, ByVal box As
System.Windows.Forms.ItemCheckEventArgs) Handles TeamIdCheckedListBox.ItemCheck

        If box.NewValue = CheckState.Checked Then

            For index = 0 To TeamIdCheckedListBox.Items.Count - 1

                If index <> box.Index Then
                    Me.TeamIdCheckedListBox.SetItemChecked(index, False)
                    Me.TeamInfoCheckedListbox.SetItemChecked(index, False)
                Else
                    TeamInfoCheckedListbox.SetItemChecked(index, True)
                End If

            Next

        End If
```

This sub-routine is taken from Development 1 and modified slightly. The sub-routine makes it so that both list boxes are selected when one of them is selected. It also ensures that only one row can be selected at a time, by unchecking all boxes when a new box is checked.

FetchTeamInfo – AdminView – Investu Development 3

```
    Sub FetchTeamInfo()

        TeamDetailsListBox.Items.Clear()

        Dim TeamID As Integer = TeamIdCheckedListBox.Text
        Dim Balance As Integer
        Dim TeamName As String = ""


        Dim ConnectionDb As New OleDbConnection(AccessDatabaseConnection)
        If ConnectionDb.State = ConnectionState.Closed Then ConnectionDb.Open()

        Dim cmd As OleDbCommand
        Dim SQLReply As OleDbDataReader

        cmd = ConnectionDb.CreateCommand
        cmd.CommandText = "SELECT * FROM tblTeams WHERE TeamID=" & TeamID & ""

        SQLReply = cmd.ExecuteReader

        For Each Record In SQLReply

            TeamName = Record.item("TeamName")
            Balance = Record.item("Balance") / 100

            TeamDetailsListBox.Items.Add("You are viewing the details of " &
Record.item("TeamName") & " - Team Code: " & Record.Item("TeamCode"))

            TeamDetailsListBox.Items.Add("The team currently has £" & Balance)
            TeamDetailsListBox.Items.Add("")
            TeamDetailsListBox.Items.Add("The following are the members of this
team:")


            FetchUsersInTeam(TeamID)
        Next

        TeamDetailsListBox.Items.Add("")
        TeamDetailsListBox.Items.Add(TeamName & " has the following open positions;")

        cmd = ConnectionDb.CreateCommand

        cmd.CommandText = "SELECT * FROM tblOpenPositions WHERE TeamName='" & TeamName
& "'"

        SQLReply = cmd.ExecuteReader

        For Each Record In SQLReply
            TeamDetailsListBox.Items.Add(Record.item("StockSymbol") & "  -  " &
Record.item("StockQuantity") & "  -  " & Record.item("BuyPrice") & "  -  " &
Record.item("TradeDate"))
        Next

    End Sub
```

The sub-routine ‘FetchTeamInfo’ fetches the information regarding a team, when it is selected from the display box. This allows the admin to look at their list of teams, then select one and receive a more detailed breakdown of that team.

```
TeamDetailsListBox.Items.Clear()
```

The display box for team details is cleared, in case the information of another team is already displayed.

```
Dim TeamID As Integer = TeamIdCheckedListBox.Text
Dim Balance As Integer
Dim TeamName As String = ""
```

Three variables are declared. ‘TeamID’ is the value in the list box of ID’s. Because of the sub-routine ‘TeamIDCheckedListBox_ItemCheck’ we know that only one value in this box can ever be selected at once. This means that the selected value is the only value checked, and so that ID is that of the team that the user wants to query.

```
Dim ConnectionDb As New OleDbConnection(AccessDatabaseConnection)
        If ConnectionDb.State = ConnectionState.Closed Then ConnectionDb.Open()

        Dim cmd As OleDbCommand
        Dim SQLReply As OleDbDataReader

        cmd = ConnectionDb.CreateCommand
        cmd.CommandText = "SELECT * FROM tblTeams WHERE TeamID=" & TeamID & ""

        SQLReply = cmd.ExecuteReader
```

A standard database connection is initialised for a Select statement. The statement, which is as follows:

```
SELECT * FROM tblTeams WHERE TeamID=" & TeamID & "
```

Selects all of the information relating to the team in question, that is stored in the table ‘tblTeams’.

```
For Each Record In SQLReply

            TeamName = Record.item("TeamName")
            Balance = Record.item("Balance") / 100

            TeamDetailsListBox.Items.Add("You are viewing the details of " &
Record.item("TeamName") & " - Team Code: " & Record.Item("TeamCode"))

            TeamDetailsListBox.Items.Add("The team currently has £" & Balance)
            TeamDetailsListBox.Items.Add("")
            TeamDetailsListBox.Items.Add("The following are the members of this team:")


            FetchUsersInTeam(TeamID)
Next
```

The data is then displayed into the information display box, for the admin to view.
The sub-routine called at the end ‘FetchUserInTeam’ takes an argument containing the value of ‘TeamID’, which is used to list every member of the team into the display.

```
TeamDetailsListBox.Items.Add("")
TeamDetailsListBox.Items.Add(TeamName & " has the following open positions;")
```

After the first database connection finishes, a second one is opened. This time, the data relating to the team that is found in a different table, ‘tblOpenPositions’ is written to the display. This cannot be done in a single SQL statement as the team information For-Loop only loops once, whereas ‘tblOpenPositions’ will typically contain many entries per team. The queries have therefore been split into two sections.

The lines of code above separate the display visually.

```
cmd.CommandText = "SELECT * FROM tblOpenPositions WHERE TeamName='" & TeamName & "'"

        SQLReply = cmd.ExecuteReader

        For Each Record In SQLReply
            TeamDetailsListBox.Items.Add(Record.item("StockSymbol") & "  -  " &
Record.item("StockQuantity") & "  -  " & Record.item("BuyPrice") & "  -  " &
Record.item("TradeDate"))
        Next
```

The second SQL query extracts relevant information from the ‘tblOpenPositions’ table. The statement uses ‘SELECT *’ and so all information in the table is returned, where the condition is met. The condition is this case looks for a match between the values of the ‘TeamName’ attribute and the value of the variable ‘TeamName’. The reason the team ID is not used is because ‘tblOpenPositions’ writes a name not an ID, which is possibly a design mistake that can be corrected in the development of ‘BuyForm’.

After the open positions of the team have been written, the display box contains all of the relevant information for the team being queried.

FetchUsersInTeam – AdminView – Investu Development 3

```
     Sub FetchUsersInTeam(ByVal TeamID As Integer)

        Dim ConnectionDb As New OleDbConnection(AccessDatabaseConnection)
        If ConnectionDb.State = ConnectionState.Closed Then ConnectionDb.Open()

        Dim cmd As OleDbCommand
        Dim SQLReply As OleDbDataReader

        cmd = ConnectionDb.CreateCommand
        cmd.CommandText = "SELECT tblUserInfo.Username FROM tblUserInfo, tblTeamUsers,
tblTeams WHERE tblTeams.TeamID=" & TeamID & " AND tblTeamUsers.TeamID =
tblTeams.TeamID AND tblTeamUsers.AccountID = tblUserInfo.AccountID"

        SQLReply = cmd.ExecuteReader

        For Each Record In SQLReply
            TeamDetailsListBox.Items.Add(Record.item("Username"))
        Next

    End Sub
```

‘FetchUsersInTeam’ uses cross table parameterized SQL to fetch the members of a team. The users are connected to teams via a link table called ‘tblTeamUsers’.

```
SELECT tblUserInfo.Username FROM tblUserInfo, tblTeamUsers, tblTeams WHERE
tblTeams.TeamID=" & TeamID & " AND tblTeamUsers.TeamID = tblTeams.TeamID AND
tblTeamUsers.AccountID = tblUserInfo.AccountID
```

This SQL statement uses the link table to connect from ‘tblUserInfo’ to ‘tblTeams’. The value being selected is ‘tblUserInfo.Username’. Values are needed from both ‘tblUserInfo’, ‘tblTeamUsers’ and ‘tblTeamUser’, and so both of them are in the ‘FROM’ section of the query.

The ‘WHERE’ section of the query contains three conditions that are connected via ‘AND’. Firstly, the ‘TeamID’ attribute in ‘tblTeams’ is matched to the ‘TeamID’ variable. Secondly, ‘tblTeamUsers.TeamID’ is matched to that of ‘tblTeams.TeamID’. This will find all of the entries in the link table who have the ‘TeamID’ value of the required team. The ‘AccountID’ value of those accounts in the link table whose ‘TeamID’ matched, are then checked against the ‘AccountID’ attribute in the ‘tblUserInfo’ table.

The accounts who match all three of these checks have their username returned into the display box.

```
SQLReply = cmd.ExecuteReader

        For Each Record In SQLReply
            TeamDetailsListBox.Items.Add(Record.item("Username"))
        Next
```

This query is executed, and each reply is written to the display box.

MainForm - Investu – Development 3

MainForm_Load – MainForm – Investu Development 3

```
    Private Sub MainForm_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load

        AccountID = LoginForm.AccountID
        TeamMode = LoginForm.TeamMode

        If TeamMode = True Then
            TeamName = LoginForm.TeamName
            LoginLabel.Text = "Welcome, " & LoginForm.Username & ". (" & TeamName &
")"
        ElseIf TeamMode = False Then
            TeamName = "0"
            LoginLabel.Text = "Welcome, " & LoginForm.Username & ""
        End If

        Balance = Math.Round(FetchBalance(), 2)
        BalanceBox.Text = "£" & Math.Round((Balance / 100), 2)

        FetchOpenPositions()
        FetchWorldNews()
        FetchMarketNews()
        FetchTradeHistory()
        LoadDetailsGrid()
        FetchAlerts()
        CreateChart()
        GraphSettings()

        GraphScaleComboBox.SelectedItem = "2"

        PopulateSymbolArray("C:\Users\Joe\Documents\visual studio 2010\Investu
Writeup\Development 3\StockSymbols.csv")
        For L = 0 To Symbols.Count - 1
            SelectStockComboBox.Items.Add(Symbols(L))
        Next
    End Sub
```

The loading of the main form is similar to that of Development 2, with the addition of a few lines of code. These lines call various sub-routines for new features that are being added to enhance the user experience.

```
FetchWorldNews()
FetchMarketNews()
```

These two sub-routines will display news for the user, both market specific and global news

```
FetchTradeHistory()
```

The trade history of an account can be useful to see which trades were made when, and the result of each trade. This can help users by giving some context to their progress, and show them where they made strong or poor trading decisions.

`LoadDetailsGrid()`

This sub-routine will provide an easy-to-understand display of the information of all of the companies in the FTSE100.

`FetchAlerts()`

This sub-routine will provide a display with the alerts a user has.

PopulateSymbolArray – MainForm – Investu Development 3

```
    Public Sub PopulateSymbolArray(ByVal FilePath As String)
```

The code for this sub-routine can be found in Development 2 on page 135.

FetchAlerts – MainForm – Investu Development 3

```
    Sub FetchAlerts()

        Dim QueryString As String

        If TeamMode Then
            QueryString = "SELECT * FROM tblAlerts WHERE TeamName='" & TeamName &
"'"
        Else
            QueryString = "SELECT * FROM tblAlerts WHERE TeamName='0' AND
AccountID=" & AccountID & ""
        End If

        Dim ConnectionDb As New OleDbConnection(AccessDatabaseConnection)
        If ConnectionDb.State = ConnectionState.Closed Then ConnectionDb.Open()

        Dim cmd As OleDbCommand = ConnectionDb.CreateCommand
        cmd.CommandText = QueryString
        Dim SQLReply As OleDbDataReader = cmd.ExecuteReader

        For Each Record In SQLReply
            If TeamMode Then
                AlertsListBox.Items.Add(GetNameUsingID(Record.item("AccountID")) & "
- " & Record.item("StockSymbol") & " - " & Record.item("AlertPrice"))
            Else
                AlertsListBox.Items.Add(Record.item("StockSymbol") & " - " &
Record.item("AlertPrice"))
            End If
        Next
        ConnectionDb.Close()
    End Sub
```

Development 3 will allow for the creation of alerts. An alert can be set for a specific stock price, and the user will receive an email notification when the stock price is reached. This will allow users to be alerted when it's a good time to buy or sell a stock, so that they don't miss out on good trading opportunities.

```
If TeamMode Then
            QueryString = "SELECT * FROM tblAlerts WHERE TeamName='" & TeamName & "'"
        Else
            QueryString = "SELECT * FROM tblAlerts WHERE TeamName='0' AND AccountID=" &
AccountID & ""
        End If
```

It first needs to be clarified whether or not the user is in team mode or their personal account. If they are on team mode, then the SQL query needs to search for the alerts that were made on that team account. Otherwise, the simulation needs to only retrieve the alerts made on a that account when it was not in team mode. In this simulation, that is signified by a team name of ‘0’.

```
        Dim ConnectionDb As New OleDbConnection(AccessDatabaseConnection)
        If ConnectionDb.State = ConnectionState.Closed Then ConnectionDb.Open()

        Dim cmd As OleDbCommand = ConnectionDb.CreateCommand
        cmd.CommandText = QueryString
```

Timer1_Tick – MainForm – Investu Development 3

```
Private Sub Timer1_Tick(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles Timer1.Tick

        Dim StockInfoString

        Try
            Timer1.Interval = 60000

            StockInfoString =
FetchStockDetailsString(SelectStockComboBox.SelectedItem)

            NameBox.Text = SplitStockInfo(StockInfoString, "Name")
            PriceBox.Text = SplitStockInfo(StockInfoString, "Price")
            ChangeBox.Text = SplitStockInfo(StockInfoString, "Change")

            Series1.Points.Clear()
            Plot24hrData()
            LoadDetailsGrid()
            UpdatePortfolio()
            GraphSettings()

            VolatilityBox.Text = CalculateVolatility(PriceBox.Text, ChangeBox.Text)
& "%"
        Catch ex As Exception
            MsgBox(ex.ToString())
        End Try

    End Sub
```

This sub-routine is only slightly different to the same sub-routine in the second development.

`LoadDetailsGrid()`

This being added to the timer-tick simulation keeps the details grid updated and relevant. This means that the grid is kept up to date with the prices displayed in the details boxes on inspection of specific stocks.

FetchTradeHistory – MainForm – Investu Development 3

```
Sub FetchTradeHistory()

        Dim MyConnection As OleDbConnection
        Dim Adapter As OleDbDataAdapter
        Dim DataSet As DataSet
        Dim Tables As DataTableCollection
        Dim Source As New BindingSource

        MyConnection = New OleDbConnection
        MyConnection.ConnectionString = AccessDatabaseConnection
        DataSet = New DataSet
        Tables = DataSet.Tables

        If TeamMode Then
            Adapter = New OleDbDataAdapter("SELECT * FROM tblTradeHistory WHERE
TeamName='" & TeamName & "'", MyConnection)
        Else
            Adapter = New OleDbDataAdapter("SELECT * FROM tblTradeHistory WHERE
AccountID=" & AccountID & " AND TeamName='0'", MyConnection)
        End If

        Adapter.Fill(DataSet, "[tblTradeHistory]")
        Dim View As New DataView(Tables(0))
        Source.DataSource = View
        DataGridView1.DataSource = View

        DataGridView1.Columns(0).Width = 50
        DataGridView1.Columns(1).Width = 50
        DataGridView1.Columns(2).Width = 50
        DataGridView1.Columns(3).Width = 75
        DataGridView1.Columns(4).Width = 75
        DataGridView1.Columns(5).Width = 125
        DataGridView1.Columns(6).Width = 50
        DataGridView1.Columns(7).Width = 50
        DataGridView1.Columns(8).Width = 80

    End Sub
```

‘FetchTradeHistory’ takes the entire history of every trade made of the current account and displays them in a ‘DataGridView’ display.

```
Dim MyConnection As OleDbConnection
```

The data will be read directly from the database. To do this a series of variables are declared. The first of which is a standard database connection, declared as ‘OleDbConnection’ type from the ‘OleDb’ namespace.

```
Dim DataSet As DataSet
```

The dataset extracted from the database will be stored in a variable called ‘DataSet’ of the ‘DataSet’ data type.

```
Dim Adapter As OleDbDataAdapter
```

The variable ‘Adapter’ of the adapter data type. Adapters are used to interact with existing data sources.[vii]

```
Dim Tables As DataTableCollection
```

When the adapter is used to convert the data from the database into the simulation, it will need somewhere to put the data. This will be a table represented by the variable ‘Tables’ of the ‘DataTableCollection’ data type.

```
Dim Source As New BindingSource
```

The ‘BindingSource’ class encapsulates the data source of a form. The variable ‘Source’ will be used as a data source for the data in the table.

```
MyConnection = New OleDbConnection
MyConnection.ConnectionString = AccessDatabaseConnection
```

A new connection is created to the database. This connection uses a new instance of ‘MyConnection’ and then ‘ConnectionString’ together.

```
DataSet = New DataSet
Tables = DataSet.Tables
```

A new dataset is created. The variable ‘Tables’ is assigned the value ‘Dataset.Tables’. ‘Dataset.Tables’ represents the collection of tables contained within the dataset.

```
        If TeamMode Then
            Adapter = New OleDbDataAdapter("SELECT * FROM tblTradeHistory WHERE
TeamName='" & TeamName & "'", MyConnection)
        Else
            Adapter = New OleDbDataAdapter("SELECT * FROM tblTradeHistory WHERE
AccountID=" & AccountID & " AND TeamName='0'", MyConnection)
        End If
```

Similarly to in previous sub-routines. The value of the SQL query is dependent on whether or not the user is in team mode or not.
A new instance of an adapter takes two arguments: an SQL query and a connection string. In the case of team mode being true, the SQL query will select all of the information from the table ‘tblTradeHistory’ where the team name is that of the user. If it is not true, then only the trades from the user are selected.

```
Adapter.Fill(DataSet, "[tblTradeHistory]")
```

The '.Fill' method of the adapter takes two arguments: A dataset, and a data source. The data source is then mapped onto the dataset using the criteria of the SQL expression in the adapter.

```
Dim View As New DataView(Tables(0))
        Source.DataSource = View
        DataGridView1.DataSource = View
```

'View' is declared as a new instance of the class 'DataView'. The 'adapter.Fill' command fills data into tables, beginning at 0. Because the query would only return a single dataset, the data that is required will be in table 0. This data is assigned to 'View'. 'View' is then made the data source of the data grid view, and hence the data in the database is displayed into the data grid view within the simulation.

```
....
DataGridView1.Columns(3).Width = 75
DataGridView1.Columns(4).Width = 75
DataGridView1.Columns(5).Width = 125
DataGridView1.Columns(6).Width = 50
....
```

The display of the data grid view is then changed to best fit the simulation, for ease of viewing.

LoadDetailsGrid – MainForm – Investu Development 3

```
    Sub LoadDetailsGrid()

        Dim MyConnection As OleDbConnection
        Dim Adapter As OleDbDataAdapter
        Dim DataSet As DataSet
        Dim Tables As DataTableCollection
        Dim Source As New BindingSource

        MyConnection = New OleDbConnection
        MyConnection.ConnectionString = AccessDatabaseConnection
        DataSet = New DataSet
        Tables = DataSet.Tables
        Adapter = New OleDbDataAdapter("SELECT * FROM [tblStockDetails]",
MyConnection)
        Adapter.Fill(DataSet, "tblStockDetails")
        Dim View As New DataView(Tables(0))
        Source.DataSource = view
        StockDetailsGrid.DataSource = view


        StockDetailsGrid.Columns(4).Width = 187
    End Sub
```

The purpose of the 'DetailsGrid' is to display the details of all of the stock symbols in the 'StockSymbols.csv' file. This is done in the same way as the previous sub-routine, however using a different query.

```
Adapter = New OleDbDataAdapter("SELECT * FROM [tblStockDetails]", MyConnection)
```

This time, the adapter query is a simple select-all statement, that takes all of the information from the 'tblStockDetails' table. This is not dependent on any criteria – all of the data is always pulls every time the sub-routine is called. For optimisation in later developments, it could be made to only pull information that has changed. For example, it is unlikely that the symbols themselves have changed, and so pulling the entire contents of the 'symbol' attribute in the table seems like a waste of processing power, when the symbols themselves will not have changed since the last time the sub-routine was called. This, however, will be a development for a later version of the simulation.

```
Adapter.Fill(DataSet, "tblStockDetails")
```

The adapter is filled with data from the table 'tblStockDetails' and then the information is displayed in a data grid view in a tab inside the simulation.

CalculateVolatility – MainForm – Investu Development 3

```
Function CalculateVolatility(ByVal Price As Decimal, ByVal Change As Decimal)
```

The code for this sub-routine can be found in Development 2.

FetchBalance – MainForm – Investu Development 3

```
Function FetchBalance()
```

The code for this sub-routine can be found in Development 2.

FetchStockDetailsString – MainForm – Investu Development 3

```
Function FetchStockDetailsString(ByVal StockSymbol As String)
```

The code for this sub-routine can be found in Development 2.

FetchMarketNews – MainForm – Investu Development 3

```
Sub FetchMarketNews()

        WebBrowser2.DocumentText = ""

        Dim StockNews As String = ""

        Try
            Dim Document As XmlDocument
            Dim DescriptionNodes As XmlNodeList

            Document = New XmlDocument()

document.Load("https://finance.google.com/finance/company_news?q=INDEXFTSE:UKX&ei=Hy
nOWZC1MpKKUunwl-gF&output=rss")

            DescriptionNodes = Document.GetElementsByTagName("description")

            For L = 1 To DescriptionNodes.Count - 1
                Stocknews += DescriptionNodes.Item(L).InnerText

            Next

            WebBrowser2.DocumentText = StockNews

        Catch ErrorVariable As Exception

            MsgBox(ErrorVariable.ToString())
        End Try

    End Sub
```

‘FetchMarketNews’ employs the ‘System.XML’ import again to extract information from an XML newsfeed about the FTSE100 and display it in the simulation.

```
WebBrowser2.DocumentText = ""
```

The sub-routine will use a web browser, which is an object in VB that can display web information. Instead of simply giving a URL for the web browser, we can extract HTML data within the XML and then write it directly into the document text of the browser, which will give only the relevant stock information, instead of loading the whole webpage, which is a slow process in VB, especially when loading pages heavily laden with ads.

```
Dim StockNews As String = ""
```

‘StockNews’ is a variable which will be given the data that needs to be written to the web browser.

```
Try

      …

Catch ErrorVariable As Exception

    MsgBox(ErrorVariable.ToString())
End Try
```

This section of code may be particularly prone to errors as it relies on data extracted from an external source. The XML data that is parsed cannot be controlled or guaranteed to be parsed correctly, which could cause errors when it is decoded in the simulation. A try-catch eliminates the chances of this causing problems for the users of the simulation.

```
Dim Document As XmlDocument
Dim DescriptionNodes As XmlNodeList
Document = New XmlDocument()
```

Within the try-catch, two variables are declared and a new instance of the class ‘XmlDocument’ is created. ‘DescriptionNodes’ will be a list of nodes that contain the news that will be written to the web browser.

```
document.Load("https://finance.google.com/finance/company_news?q=INDEXFTSE:UKX&ei=HynOWZC
1MpKKUunwl-gF&output=rss")
```

The document loaded is from an RSS feed on finance.google.com.

```
DescriptionNodes = Document.GetElementsByTagName("description")
```

From the document that was just loaded, the ‘GetElementsByTagName’ method is called. This takes a single argument: The name of the elements to be indexed. This list of elements is stored in the variable ‘DescriptionNodes’.

```
For L = 1 To DescriptionNodes.Count - 1
      Stocknews += DescriptionNodes.Item(L).InnerText
Next
```

Once the list of nodes is retrieved, they are looped through using a For-Loop. This loop goes through all of the elements and appends them to the variable ‘StockNew’, by using the ‘InnerText’ method on the element. This method is used to get the text contained within the element. In this case, this text is the actual story in the news article.

```
WebBrowser2.DocumentText = StockNews
```

The document text of the web browser is then set to the variable ‘StockNews’ which now contains all of the news that has been extracted from the XML.

FetchWorldNews – MainForm – Investu Development 3

```
Sub FetchWorldNews()

        WebBrowser1.DocumentText = ""
        Dim StockNews As String = ""

        Try
            Dim document As XmlDocument
            Dim TitleNodes, DescriptionNodes, LinkNodes, ArticleNodes As XmlNodeList

            document = New XmlDocument()
            document.Load("http://feeds.bbci.co.uk/news/world/rss.xml")

            TitleNodes = document.GetElementsByTagName("title")
            DescriptionNodes = document.GetElementsByTagName("description")
            LinkNodes = document.GetElementsByTagName("link")
            ArticleNodes = document.GetElementsByTagName("pubDate")

            For L = 0 To 25

                stocknews += "<font size=" & "+1" & ">" & "<b>" & TitleNodes.Item(L
+ 2).InnerText & "    " & "</b>" & "</font>"
                stocknews += "<font size=" & "-2" & ">" &
ArticleNodes.Item(L).InnerText & "</font>" & "<br>"
                stocknews += DescriptionNodes.Item(L + 1).InnerText & "<Br>"
                stocknews += "<font size=" & "-1" & ">" & "Read more at " &
LinkNodes.Item(L + 2).InnerText & "<font>" & "<Br><Br><Br>"
            Next

            WebBrowser1.DocumentText += StockNews
        Catch errorVariable As Exception

            MsgBox(errorVariable.ToString())
        End Try

    End Sub
```

'FetchWorldNews' is a sub-routine with the same purpose – to display news to the user. However, this sub-routine fetches world news, instead of market specific news. The purpose of displaying this for the user is to enhance the trading experience and allow users to gauge the current world climate and then extrapolate its impact on the market.

The sub-routine works in much the same way, with a few differences.

```
Dim TitleNodes, DescriptionNodes, LinkNodes, ArticleNodes As XmlNodeList
```

In the previous sub-routine, the news was packaged helpfully into description nodes, that could be simply extracted. This news source has their data parsed into different XML nodes, which means more nodes need to be indexed and searched for data. These nodes are: 'TitleNodes' for the title of the articles, 'DescriptionNodes' for the description of the article, 'LinkNodes' for the URL of the article to allow the users to follow a link and read more on a certain story, and 'ArticleNodes', which contain the publish date of the article.

```
TitleNodes = document.GetElementsByTagName("title")
DescriptionNodes = document.GetElementsByTagName("description")
LinkNodes = document.GetElementsByTagName("link")
ArticleNodes = document.GetElementsByTagName("pubDate")
```

Similarly to the previous sub-routine, the nodes are then fetched from the XML using a tag name, which is the name inside the XML used to identify each section of data.

```
For L = 0 To 24

                stocknews += "<font size=" & "+1" & ">" & "<b>" & TitleNodes.Item(L +
2).InnerText & "    " & "</b>" & "</font>"
                stocknews += "<font size=" & "-2" & ">" & ArticleNodes.Item(L).InnerText
& "</font>" & "<br>"
                stocknews += DescriptionNodes.Item(L + 1).InnerText & "<Br>"
                stocknews += "<font size=" & "-1" & ">" & "Read more at " &
LinkNodes.Item(L + 2).InnerText & "<font>" & "<Br><Br><Br>"
            Next
```

This particular news site has many articles. Instead of listing all of them, only the top 24 will be displayed. Therefore the For-Loop loops 25 times. Inside the loop, the information from each of the node lists is output. The information is formatted using HTML, because the information will be inserted into the document text of the web browser.

```
WebBrowser1.DocumentText += StockNews
```

The information in the new ‘StockNews’ string, containing the information for all 25 news stories, is added to the document text of the web browser, and therefore displayed for the user to view in the simulation.

SplitStockInfo – MainForm – Investu Development 3

```
Function SplitStockInfo(ByVal StringToSplit As String, ByVal DetailsToExtract As
String)
```

The code for this sub-routine can be found in Development 1 on page 66

BuyButton_Click – MainForm – Investu Development 3

```
Private Sub BuyButton_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs)
```

The code for this sub-routine can be found in Development 2 on page 140.

SelectStockComboBox_SelectedIndexChang – MainForm – Investu Development 3

```
Private Sub SelectStockComboBox_SelectedIndexChanged(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles SelectStockComboBox.SelectedIndexChanged
```

The code for this sub-routine can be found in Development 2 on page 142.

Plot24hrData – MainForm – Investu Development 3

```
    Public Sub Plot24hrData()
```

The code for this sub-routine can be found in Development 2 on page 142.

PlotNewPoint – MainForm – Investu Development 3

```
    Sub PlotNewPoint(ByVal XValue As Decimal, ByVal YValue As Decimal)
```

The code for this sub-routine can be found in Development 2 on page 143.

GetStockChange – MainForm – Investu Development 3

```
    Function GetStockChange(ByVal StockSymbol As String)
```

The code for this sub-routine can be found in Development 2 on page 115.

GetStockPrice – MainForm – Investu Development 3

```
    Function GetStockPrice(ByVal StockSymbol As String)
```

The code for this sub-routine can be found in Development 2 on page 117.

GetStockName – MainForm – Investu Development 3

```
    Function GetStockName(ByVal StockSymbol As String)
```

The code for this sub-routine can be found in Development 2 on page 116.

ClosePositionsButton_Click – MainForm – Investu Development 3

```
Private Sub ClosePositionsButton_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles ClosePositionsButton.Click
```

The code for this sub-routine can be found in Development 2 on page 144.

InfoButton_Click – MainForm – Investu Development 3

```
Private Sub InfoButton_Click(sender As System.Object, e As System.EventArgs) Handles
InfoButton.Click

        Dim SelectedStock As String = OpenPositionsListBox.SelectedIndex

        Dim Value As Decimal = OpenPositions(SelectedStock).StockValue
        Dim StockName As String = OpenPositions(SelectedStock).StockName
        Dim Quantity As String = OpenPositions(SelectedStock).StockQuantity
        Dim StockSymbol As String = OpenPositions(SelectedStock).StockSymbol


        Dim CurrentTotalPrice As Decimal = Math.Round(((GetStockPrice(StockSymbol) *
Quantity) / 100), 2)
        Dim TotalTradePrice As Decimal = Math.Round(((Value * Quantity) / 100), 2)

        MsgBox("You bought" & Quantity & " " & StockName & " shares for a price of "
& Value & " each, costing a total of £" & TotalTradePrice & ". " & vbNewLine &
StockName & " shares are now worth " & GetStockPrice(StockSymbol) & " each, making
your shares worth a total of £" & CurrentTotalPrice & "." & vbNewLine & "Your net
gain from this trade is £" & TotalTradePrice - CurrentTotalPrice & ".")
    End Sub
```

As a way of providing more information to the user regarding their current investments, ‘InfoButton’ allows users to inspect a position and look at some information regarding it.

```
Dim SelectedStock As String = OpenPositionsListBox.SelectedIndex

Dim Value As Decimal = OpenPositions(SelectedStock).StockValue
Dim StockName As String = OpenPositions(SelectedStock).StockName
Dim Quantity As String = OpenPositions(SelectedStock).StockQuantity
Dim StockSymbol As String = OpenPositions(SelectedStock).StockSymbol
```

The variables here represent some information about a position. The information is fetched from the ‘OpenPositions’ list, which contains information of all of the currently open positions.

```
Dim CurrentTotalPrice As Decimal = Math.Round(((GetStockPrice(StockSymbol) * Quantity) /
100), 2)
Dim TotalTradePrice As Decimal = Math.Round(((Value * Quantity) / 100), 2)
```

Some calculations are performed to get the total value of the trade when it was made and then the current total price.

```
        MsgBox("You bought" & Quantity & " " & StockName & " shares for a price of " &
Value & " each, costing a total of £" & TotalTradePrice & ". " & vbNewLine & StockName &
" shares are now worth " & GetStockPrice(StockSymbol) & " each, making your shares worth
a total of £" & CurrentTotalPrice & "." & vbNewLine & "Your net gain from this trade is
£" & TotalTradePrice - CurrentTotalPrice & ".")
```

The information is then displayed to the user.

StoreNewTrade_Click – MainForm – Investu Development 3

```
Sub StoreNewTrade(ByVal OpenPositionID As String, ByVal CurrentPrice As Integer)

        Dim InsertString As String = ""

        Dim ConnectionDb As New OleDbConnection(AccessDatabaseConnection)
        If ConnectionDb.State = ConnectionState.Closed Then ConnectionDb.Open()

        Dim cmd As OleDbCommand = ConnectionDb.CreateCommand
        cmd.CommandText = "SELECT StockSymbol, StockQuantity FROM tblOpenPositions
WHERE OpenPositionID='" & OpenPositionID & "'"
        Dim SQLReply As OleDbDataReader = cmd.ExecuteReader

        For Each Record In SQLReply
            InsertString = "INSERT INTO tblTradeHistory (AccountID, StockSymbol,
StockQuantity, BuyOrSell, TradePrice, TradeDate, TeamName) VALUES ('" & AccountID &
"','" & Record.item("StockSymbol") & "','" & Record.item("StockQuantity") &
"','Sell','" & CurrentPrice & "','" & DateTime.Now & "','" & TeamName & "')"
        Next

        SQLReply.Close()

        cmd.CommandText = InsertString
        cmd.ExecuteNonQuery()
        ConnectionDb.Close()
    End Sub
```

One of the new features added to Development 3 is the trade history of an account. This is done by storing the details of a buy or sell each time it is made. The information is stored into a table called ‘tblTradeHistory’. It could be argued that this will make ‘tblOpenPositions’ redundant as there will now be duplication of data, e.g. the same trade information is ‘tblTradeHistory’ as well as ‘tblOpenPositions’. This may be the case, however time constraints mean merging the tables is impractical at this stage.

```
Dim ConnectionDb As New OleDbConnection(AccessDatabaseConnection)
```

```
If ConnectionDb.State = ConnectionState.Closed Then ConnectionDb.Open()
```

A standard connection is initiated to the database. From here, data will be retrieved that can then be inserted into the database. This data needs to be extracted from the database first because only some of the data is available in the simulation – but we can use the information we have to get the rest of the information from the database.

```
Dim cmd As OleDbCommand = ConnectionDb.CreateCommand
        cmd.CommandText = "SELECT StockSymbol, StockQuantity FROM tblOpenPositions WHERE
OpenPositionID='" & OpenPositionID & "'"
```

The Select statement fetches the symbol and quantity from the database using the primary key for that table.

```
Dim SQLReply As OleDbDataReader = cmd.ExecuteReader

For Each Record In SQLReply
            InsertString = "INSERT INTO tblTradeHistory (AccountID, StockSymbol,
StockQuantity, BuyOrSell, TradePrice, TradeDate, TeamName) VALUES ('" & AccountID & "','"
& Record.item("StockSymbol") & "','" & Record.item("StockQuantity") & "','Sell','" &
CurrentPrice & "','" & DateTime.Now & "','" & TeamName & "')"
Next
```

The query is executed. From the reply, a new string is created. This string will be a new SQL statement, for an insert in ‘tblTradeHistory’. The values returned from the previous query and used as inputs for this new query. The sub-routine is called on closing the position, and so the value for ‘BuyOrSell’ can be hard wired as ‘Sell’. The other data is retrieved from the arguments passed to the sub-routine and from information returned from the query.

```
SQLReply.Close()

cmd.CommandText = InsertString
cmd.ExecuteNonQuery()
ConnectionDb.Close()
```

Once the ‘InsertString’ value has been set, the reply is closed. The new value of the command text is now ‘InsertString’. This is not a query, and so can simply be executed with the ‘ExecuteNonQuery’ method. The connection is then closed and the process of storing new trade information into the table ‘tblTradeHistory’ is complete.

CreateAlertButton_Click – MainForm – Investu Development 3

```
Private Sub CreateAlertButton_Click(sender As System.Object, e As System.EventArgs)
Handles CreateAlertButton.Click

        Dim UpOrDown As String

        If SelectStockComboBox.Text <> "Select Stock Symbol" Then
            If AlertPriceBox.Text > PriceBox.Text Then
                UpOrDown = "UP"
            Else
                UpOrDown = "DOWN"
            End If


            CreateNewAlert(UpOrDown)
            AlertsListBox.Items.Clear()
            FetchAlerts()
        Else
            MsgBox("You need to select a stock from the drop down menu first.")
        End If
    End Sub
```

The alerts system will work by providing a small interface in which the user can put a price. There will be a button, which when activated, will call the above sub-routine.

```
Dim UpOrDown As String
```

The variable ‘UpOrDown’ is used as an identifier, so that the simulation knows whether the user has set a price above the current price – meaning they want to be alerted when the stock price goes above that price, or a price below the current price, meaning they want to be alerted when the stock price goes below that price.

For example, if the current stock price is 100, and the user sets an alert price of 105, then it is clear that the user wants to be alerted when the stock price reaches 105 or above. This is necessary so that the simulation knows which comparison to make on the current price and the alert price, e.g. knows when to compare the data with a less than symbol or greater than symbol.

```
If SelectStockComboBox.Text <> "Select Stock Symbol" Then
     ….
Else
      MsgBox("You need to select a stock from the drop down menu first.")
End If
```

A conditional is used to ensure that there is a stock symbol selected – otherwise the simulation would not know which symbol to apply the alert to.

```
If AlertPriceBox.Text > PriceBox.Text Then
      UpOrDown = "UP"
Else
      UpOrDown = "DOWN"
End If
```

Within the conditional, another conditional works out the value of ‘UpOrDown’. If the price of the alert is above the current price, then ‘UpOrDown’ is set to ‘Up’ – the user is waiting for the price to go up to that price. If the value of the alert price is less than the current price, then ‘UpOrDown’ is set to ‘Down’ – the user is waiting for the price to go down to that price.

```
CreateNewAlert(UpOrDown)
```

Once the value of ‘UpOrDown’ is set, it is passed as an argument to the sub-routine ‘CreateNewAlert’. This will add the alert to the database.

```
AlertsListBox.Items.Clear()
FetchAlerts()
```

The visual display for the alerts is cleared and then ‘FetchAlerts’ is called to re-populate it, in order to refresh the display with the newly added alert.

CreateNewAlert – MainForm – Investu Development 3

```
Sub CreateNewAlert(ByVal UpOrDown As String)

        If ValidateAlertPrice(AlertPriceBox.Text) Then

            Dim ConnectionDb As New OleDbConnection(AccessDatabaseConnection)
            If ConnectionDb.State = ConnectionState.Closed Then ConnectionDb.Open()

            Dim cmd As OleDbCommand = ConnectionDb.CreateCommand
            cmd.CommandText = "INSERT INTO tblAlerts (AccountID, StockSymbol,
AlertPrice, TeamName, UpOrDown) VALUES (" & AccountID & ",'" &
SelectStockComboBox.SelectedItem & "','" & AlertPriceBox.Text & "','" & TeamName &
"','" & UpOrDown & "')"
            cmd.ExecuteNonQuery()
            ConnectionDb.Close()

            MsgBox("A new alert for " & NameBox.Text & " at " & AlertPriceBox.Text &
" has been set.")
        Else
            MsgBox("Please enter a valid alert price.")
        End If

    End Sub
```

```
If ValidateAlertPrice(AlertPriceBox.Text) Then
   …..
Else
      MsgBox("Please enter a valid alert price.")
End If
```

The price that will be used as the alert price is first validated to ensure it is a valid value.

```
Dim ConnectionDb As New OleDbConnection(AccessDatabaseConnection)
If ConnectionDb.State = ConnectionState.Closed Then ConnectionDb.Open()
```

A new connection to the database is initiated.

```
Dim cmd As OleDbCommand = ConnectionDb.CreateCommand
            cmd.CommandText = "INSERT INTO tblAlerts (AccountID, StockSymbol, AlertPrice,
TeamName, UpOrDown) VALUES (" & AccountID & ",'" & SelectStockComboBox.SelectedItem &
"','" & AlertPriceBox.Text & "','" & TeamName & "','" & UpOrDown & "')"
```

The command text, or SQL statement, is given a value. In this sub-routine, an insert statement is used to insert the alert into the database. All of the attributes of the table are given a value.

```
cmd.ExecuteNonQuery()
ConnectionDb.Close()
```

The insert is executed, and the connection to the database is closed.

```
MsgBox("A new alert for " & NameBox.Text & " at " & AlertPriceBox.Text & " has been
set.")
```

The user is informed that their alert creation was successful.

ValidateAlertPrice – MainForm – Investu Development 3

```
    Function ValidateAlertPrice(ByVal Price As String)

        Dim Numbers As New System.Text.RegularExpressions.Regex("[0-9]")
        Dim NotNumbers As New System.Text.RegularExpressions.Regex("[^0-9]")

        If Numbers.Matches(Price).Count < 2 Then Return False
        If NotNumbers.Matches(Price).Count > 0 Then Return False

        Return True
    End Function
```

The validation for the alert price consists of two regular expressions that check for the number of numbers and the number of non-numeric characters. If the number of numbers is 0 or 1 then the validation fails. If the number of non-numeric characters is greater than 0 then the check also fails.

OpenToolStripButton_Click – MainForm – Investu Development 3

```
Private Sub OpenToolStripButton_Click(sender As System.Object, e As
System.EventArgs) Handles OpenToolStripButton.Click

        Dim TeamCode As String
        TeamCode = InputBox("Please input the 5 character Team Code here, issued to
you by your teacher", "Join Team", "")

        If ValidTeamCode(TeamCode) Then
            If UserAlreadyInTeam(AccountID) Then
                DeleteUserFromTeam(AccountID)
                AddNewPlayerToTeam(AccountID, TeamCode)
            Else
                AddNewPlayerToTeam(AccountID, TeamCode)
            End If
            LoginForm.Show()
            Me.Close()
        Else
            MsgBox("The team code you entered was not valid.")
        End If
    End Sub
```

The team system has been implemented into the simulation, however there is still no way for users to join teams, other than being written into the database manually. This sub-routine gives the users the ability to join a team easily and within the simulation. The sub-routine is called on click of a button in the ‘OpenToolStrip’ control.

```
Dim TeamCode As String
TeamCode = InputBox("Please input the 5 character Team Code here, issued to you by your
teacher", "Join Team", "")
```

The variable ‘TeamCode’ is defined, and given a value from an input box shown to the user.

```
If ValidTeamCode(TeamCode) Then
     …
Else
     MsgBox("The team code you entered was not valid.")
End If
```

A conditional is used that ensures the team code entered by the user is valid, by using the function ‘ValidTeamCode’. If the code isn’t valid, then the user is shown an error message.

```
If UserAlreadyInTeam(AccountID) Then
      DeleteUserFromTeam(AccountID)
      AddNewPlayerToTeam(AccountID, TeamCode)
Else
      AddNewPlayerToTeam(AccountID, TeamCode)
End If
```

Inside the first validation conditional is another conditional, that this time checks whether or not the user is already in a team. If they are found to already be in a team, then an extra sub-routine is called, that removes them from their current team first, then adds them to their desired team. If they are not in a team, then there is no need to remove them first, so they are added straight to their desired team.

```
LoginForm.Show()
Me.Close()
```

After joining a new team, the user is signed out, so that the program can be reloaded in either team mode or personal account mode.

DeleteUserFromTeam – MainForm – Investu Development 3

```
Sub DeleteUserFromTeam(ByVal AccountID As Integer)

        Dim ConnectionDb As New OleDbConnection(AccessDatabaseConnection)
        If ConnectionDb.State = ConnectionState.Closed Then ConnectionDb.Open()
        Dim cmd As OleDbCommand = ConnectionDb.CreateCommand

        cmd.CommandText = "DELETE * FROM tblTeamUsers WHERE AccountID=" & AccountID
& ""
        cmd.ExecuteNonQuery()
        MsgBox("You have been removed from your current team.")
        ConnectionDb.Close()
End Sub
```

```
DELETE * FROM tblTeamUsers WHERE AccountID=" & AccountID & "
```

This sub-routine opens a connection to the database, then executes a Delete SQL statement, which removes the connection between the user and the team, therefore removing the user from being a member of the team.

UserAlreadyInteam – MainForm – Investu Development 3

```
Function UserAlreadyInTeam(ByVal AccountID As Integer)


        Dim ConnectionDb As New OleDbConnection(AccessDatabaseConnection)
        If ConnectionDb.State = ConnectionState.Closed Then ConnectionDb.Open()

        Dim cmd As OleDbCommand
        Dim SQLReply As OleDbDataReader

        cmd = ConnectionDb.CreateCommand
        cmd.CommandText = "SELECT AccountID FROM tblTeamUsers WHERE AccountID=" &
AccountID & ""
        SQLReply = cmd.ExecuteReader

        For Each Record In SQLReply
            Return True
        Next
        ConnectionDb.Close()

        Return False
    End Function
```

Another of the validation checks that need to be done when adding a user to a team is a check to ensure the user is not already in a team.

```
"SELECT AccountID FROM tblTeamUsers WHERE AccountID=" & AccountID & ""
```

This is done through the standard database connection, and then setting the command text to the above SQL statement. This will return every connection between the account and a team. This will always return either 1 or 0 replies – 1 if the user is in a team, or 0 if the user is not in a team.

```
For Each Record In SQLReply
      AlreadyInTeam = True
Next
```

Hence, if a record exists, it can be assumed that the user is already in a team, and true can be returned.

ValidTeamCode – MainForm – Investu Development 3

```
Function ValidTeamCode(ByVal TeamCode As String)

        If CheckTeamCodeExists(TeamCode) Then
            If EmptySpaceInTeam(TeamCode) Then
                Return True
            Else
                ErrorMsg = "The team you are trying to join is already full."
            End If
        Else
            ErrorMsg = "The Team Code you entered does not exist."
        End If

        Return False
    End Function
```

'ValidTeamCode' is another function used to validate the users attempt at joining a team. The function will only return true once two other functions also return true.

```
If CheckTeamCodeExists(TeamCode) Then
      …
Else
      Msgbox("The Team Code you entered does not exist.")
End If
```

The first conditional checks that the actual team code exists. If it does not, then a relevant error message is returned.

```
If EmptySpaceInTeam(TeamCode) Then
      Return True
Else
      ErrorMsg = "The team you are trying to join is already full."
End If
```

Inside the first conditional, the next conditional checks that the team the user is trying to join is not full. If there is found to be space in the team then the function returns true.

```
Return False
```

If the code has not returned true already, then it means the validation failed and false is returned.

CheckTeamCodeExists – MainForm – Investu Development 3

```
    Function CheckTeamCodeExists(ByVal TeamCode As String)

        If TeamCode = "" Then
            Return False
        End If

        Dim ConnectionDb As New OleDbConnection(AccessDatabaseConnection)
        If ConnectionDb.State = ConnectionState.Closed Then ConnectionDb.Open()

        Dim cmd As OleDbCommand
        Dim SQLReply As OleDbDataReader

        cmd = ConnectionDb.CreateCommand
        cmd.CommandText = "SELECT TeamCode FROM tblTeams"
        SQLReply = cmd.ExecuteReader

        For Each Record In SQLReply
            If Record.item("TeamCode") = TeamCode Then Return True
        Next
        ConnectionDb.Close()
        Return False

    End Function
```

'CheckTeamCodeExists' queries the database to see if the team code that the user is trying to use to join a team is valid. This is done via a Select query that returns all of the possible team codes.

```
SELECT TeamCode FROM tblTeams
```

Once there is a list of all possible team codes, they can be iterated through, and compared to the users team code.

```
For Each Record In SQLReply
     If Record.item("TeamCode") = TeamCode Then Return True
Next
```

If one of the replies is equal to the user team code, then the team code has been proven to exist and the function can return true.

EmptySpaceInTeam – MainForm – Investu Development 3

```
    Function EmptySpaceInTeam(ByVal TeamCode As String)

        Dim UsersAlreadyInTeam As Integer = 0

        Dim ConnectionDb As New OleDbConnection(AccessDatabaseConnection)
        If ConnectionDb.State = ConnectionState.Closed Then ConnectionDb.Open()

        Dim cmd As OleDbCommand
        Dim SQLReply As OleDbDataReader

        cmd = ConnectionDb.CreateCommand
        cmd.CommandText = "SELECT * FROM tblTeamUsers WHERE
tblTeamUsers.TeamID=tblTeams.TeamID AND tblTeams.TeamCode='" & TeamCode & "'"

        SQLReply = cmd.ExecuteReader

        For Each Record In SQLReply
            UsersAlreadyInTeam += 1
        Next

        If UsersAlreadyInTeam < 4 Then
            Return True
        End If

        ConnectionDb.Close()

        Return False
    End Function
```

```
Dim UsersAlreadyInTeam As Integer = 0
```

First, the variable ‘UsersAlreadyInTeam’ is declared. ‘UsersAlreadyInTeam’ is a count of how many accounts are currently in the team. If this is less than 4, then there will be space in the team for the new user to join. If it is 4, then the team is full and the user can’t join.

```
cmd.CommandText = "SELECT * FROM tblTeamUsers WHERE tblTeamUsers.TeamID=tblTeams.TeamID
AND tblTeams.TeamCode='" & TeamCode & "'"
```

The simulation has the value of ‘TeamCode’, however in order to query the link table, the primary key identifying the team is needed. This is ‘TeamID’.

The SQL statement needs to use two clauses in the ‘Where’ section of the query. The first ensures that the data retrieved is that which has matching ‘TeamID’ values, and the second ensures that only the data of teams with a matching team code is retrieved.

This use of cross-table parametrized SQL reduces the need for two SQL statements, making the process of querying the database more efficient.

```
For Each Record In SQLReply
      UsersAlreadyInTeam += 1
Next
```

Every record that is returned from the query represents a single line returned from ‘tblTeamUsers’. ‘tblTeamUsers’ is a link table and so each line represents the connection of a user to a team. Therefore by counting the number of replies, we can work out how many users are in the team.

‘UsersAlreadyInTeam’ is incremented by 1 each time.

```
If UsersAlreadyInTeam < 4 Then
     Return True
End If

Return False
```

If the number of users in the team is less than 4, then the function returns true. If the function has not exited by the time it gets to the end, then it means that there is no space in the team, and so false is returned.

AddNewPlayToTeam – MainForm – Investu Development 3

```
Sub AddNewPlayerToTeam(ByVal AccountID As Integer, ByVal TeamCode As String)


        Dim TeamID As Integer
        Dim TeamName As String = ""

        Dim ConnectionDb As New OleDbConnection(AccessDatabaseConnection)
        If ConnectionDb.State = ConnectionState.Closed Then ConnectionDb.Open()

        Dim cmd As OleDbCommand
        Dim SQLReply As OleDbDataReader

        cmd = ConnectionDb.CreateCommand
        cmd.CommandText = "SELECT TeamID, TeamName FROM tblTeams WHERE TeamCode='" &
TeamCode & "'"

        SQLReply = cmd.ExecuteReader

        For Each Record In SQLReply
            TeamID = Record.item("TeamID")
            TeamName = Record.item("TeamName")
        Next

        SQLReply.Close()

        cmd.CommandText = "INSERT INTO tblTeamUsers (AccountID, TeamID) VALUES (" &
AccountID & ",'" & TeamID & "')"


        cmd.ExecuteNonQuery()
        MsgBox("You have joined " & TeamName & "")
        ConnectionDb.Close(
    End Sub
```

After all of the validation checks are passed, the user is finally allowed to join the team. This is done via the ‘AddNewPlayerToTeam’ sub-routine, that takes two arguments: ‘AccountID’ and ‘TeamCode’.

```
cmd.CommandText = "SELECT TeamID, TeamName FROM tblTeams WHERE TeamCode='" & TeamCode &
"'"
```

The value of ‘TeamCode’ is used in the first SQL statement of the sub-routine, which retrieves the ‘TeamID’ value from the database, as well as the name of the team.

```
For Each Record In SQLReply
     TeamID = Record.item("TeamID")
     TeamName = Record.item("TeamName")
Next
```

The results of the query gives the simulation a value for ‘TeamID’ and ‘TeamName’

```
SQLReply.Close()
```

```
        cmd.CommandText = "INSERT INTO tblTeamUsers (AccountID, TeamID) VALUES (" &
AccountID & ",'" & TeamID & "')"
```

The reply is closed, then the command text is set to a new SQL statement. This time, we are using the value we have retrieved to insert a new link into the link table ‘tblTeamUsers’. The values passed into ‘AccountID’ and ‘TeamID’ are the values of the account ID which was passed as an argument, and ‘TeamID’ which was retrieved in the previous SQL statement.

```
cmd.ExecuteNonQuery()
ConnectionDb.Close()
```

The SQL is executed, and the connection to the database is closed.

```
MsgBox("You have joined " & TeamName & "")
```

The user is informed that they have successfully joined the team.

BuyForm - Investu – Development 3

Development 3 adds only a single feature to 'BuyForm'. That is, the ability for users to add a description stating any relevant information relating to the buy that they are about to execute. This feature will be useful for keeping all team members up to date on the reasons for making a trade, as well as reminding the buyer of the buy and sell conditions of their position.

For the first version of 'BuyForm' code, refer to page 74.

Global Variables – BuyForm – Investu Development 3

```
Public Class BuyForm
```

The full code for this section can be found on page 74.

BuyForm_Load – BuyForm – Investu Development 3

```
    Private Sub BuyForm_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
```

```
        TeamMode = MainForm.TeamMode
        TeamName = MainForm.TeamName
```

This code is the same as in Development 1, with the addition of the 2 lines of code above. This information is important because it will be needed when adding new positions to the database.

The full code for this sub-routine can be found on page 75.

QuantitySlider_Scroll– BuyForm – Investu Development 3

```
Private Sub QuantitySlider_Scroll(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles QuantitySlider.Scroll
```

The full code for this sub-routine can be found on page 76.

BuyButton_Click – BuyForm – Investu Development 3

```
    Private Sub BuyButton_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles BuyButton.Click

        If NotesBox.TextLength > 255 Then
            MsgBox("Your note is too long.")
        Else
            For L = 0 To MainForm.Symbols.Count - 1

                If MainForm.Symbols(L) = MainForm.SelectStockComboBox.SelectedItem Then

                    If MainForm.Balance > (Quantity * StockPrice) Then
                        MainForm.OpenPositions.Add(New StockAttributes With
{.StockSymbol = MainForm.Symbols(L), .StockValue = StockPrice, .StockQuantity =
Quantity, .OpenPositionID = MainForm.AccountID & DateTime.Now, .BuyDate =
DateTime.Now, .StockName = Stockname})
                        StoreNewPosition(MainForm.AccountID, Stockname, Stocksymbol,
Quantity, StockPrice, DateTime.Now, TeamName, NotesBox.Text)
                        UpdateBalance(MainForm.AccountID)
                        MainForm.UpdatePortfolio()

                        Me.Close()
                    Else
                        MsgBox("You don't have enough money to buy that many " &
Stockname & " stocks.")
                    End If
                End If
            Next
            MainForm.FetchTradeHistory()
        End If
    End Sub
```

```
If NotesBox.TextLength > 255 Then
       MsgBox("Your note is too long.")
Else
```

The longest value for a string within the database is 255 characters. Therefore, this conditional is necessary to avoid the Insert statement from crashing the program.

```
For L = 0 To MainForm.Symbols.Count - 1
         If MainForm.Symbols(L) = MainForm.SelectStockComboBox.SelectedItem Then
```

Within the condition is a For-Loop and another conditional. These are used to find the correct stock symbol that the user is about to buy. Once the value of this symbol is determined, then it can be used to update ‘OpenPositions’ and the database.

```
If MainForm.Balance > (Quantity * StockPrice) Then
```

The final nested-conditional checks that the use has enough money to make the trade. If this final conditional returns true then the following code executes:

```
MainForm.OpenPositions.Add(New StockAttributes With {.StockSymbol =
MainForm.Symbols(L), .StockValue = StockPrice, .StockQuantity = Quantity, .OpenPositionID
= MainForm.AccountID & DateTime.Now, .BuyDate = DateTime.Now, .StockName = Stockname})
```

This code is responsible for updating the ‘OpenPositions’ list. The attributes of the list are set to the relevant values for the trade, such as quantity and price.

```
StoreNewPosition(MainForm.AccountID, Stockname, Stocksymbol, Quantity, StockPrice,
DateTime.Now, TeamName, NotesBox.Text)
```

Then, the new position is stored into the database using ‘StoreNewPosition’. This sub-routine takes 8 arguments to create entries into the database in the relevant tables, like ‘tblOpenPositions’ and ‘tblTradeHistory’.

```
UpdateBalance(MainForm.AccountID)
MainForm.UpdatePortfolio()
```

The sub-routine ‘UpdateBalance’ is called to store the new balance in the database, and ‘UpdatePortfolio’ is called in the main form in order to update the visual display with the new position.

```
MainForm.FetchTradeHistory()
```

‘FetchTradeHistory’ is called to update the trade-history display with the new position added.

UpdateBalance – BuyForm – Investu Development 3

```
Sub UpdateBalance(ByVal AccountID As Integer)

        MainForm.Balance = MainForm.Balance - ((Quantity * StockPrice))
        MainForm.BalanceBox.Text = "£" & Math.Round((MainForm.Balance / 100), 2)

        Dim CommandString As String
        If TeamMode Then
            CommandString = "UPDATE tblTeams SET tblTeams.Balance=" &
MainForm.Balance & " WHERE tblTeams.TeamName='" & TeamName & "';"

        Else
            CommandString = "UPDATE tblUserInfo SET tblUserInfo.Balance=" &
MainForm.Balance & " WHERE (((tblUserInfo.AccountID)=" & AccountID & "));"
        End If

        Dim ConnectionDb As New OleDbConnection(AccessDatabaseConnection)
        If ConnectionDb.State = ConnectionState.Closed Then ConnectionDb.Open()

        Dim cmd As OleDbCommand = ConnectionDb.CreateCommand

        cmd.CommandText = CommandString

        cmd.ExecuteNonQuery()
        ConnectionDb.Close()
    End Sub
```

```
MainForm.Balance = MainForm.Balance - ((Quantity * StockPrice))
MainForm.BalanceBox.Text = "£" & Math.Round((MainForm.Balance / 100), 2)
```

First, the balance is set within the simulation. As this is a buy trade, the price is taken away from the current balance. The balance textbox is then formatted to show the new change.

```
Dim CommandString As String
        If TeamMode Then
            CommandString = "UPDATE tblTeams SET tblTeams.Balance=" & MainForm.Balance &
" WHERE tblTeams.TeamName='" & TeamName & "';"

        Else
            CommandString = "UPDATE tblUserInfo SET tblUserInfo.Balance=" &
MainForm.Balance & " WHERE (((tblUserInfo.AccountID)=" & AccountID & "));"
        End If
```

The command string is set to one of two values. If the user is in ‘TeamMode’ then the balance is updated in the ‘tblTeams’ table. If the user is not in ‘TeamMode’, then the table ‘tblUserInfo’ is updated.

```
Dim ConnectionDb As New OleDbConnection(AccessDatabaseConnection)
If ConnectionDb.State = ConnectionState.Closed Then ConnectionDb.Open()
Dim cmd As OleDbCommand = ConnectionDb.CreateCommand
cmd.CommandText = CommandString
cmd.ExecuteNonQuery()
ConnectionDb.Close()
```

The connection to the database is initialised, and the SQL command is executed, updating the balance within the database.

StoreNewPosition – BuyForm – Investu Development 3

```
    Sub StoreNewPosition(ByVal ID As Integer, ByVal StockName As String, ByVal
StockSymbol As String, ByVal StockQuantity As Integer, ByVal StockValue As Decimal,
ByVal BuyDate As Date, ByVal TeamName As String, ByVal Notes As String)


        Dim ConnectionDb As New OleDbConnection(AccessDatabaseConnection)
        If ConnectionDb.State = ConnectionState.Closed Then ConnectionDb.Open()

        Dim cmd As OleDbCommand = ConnectionDb.CreateCommand
        cmd.CommandText = "INSERT INTO TblOpenPositions (AccountID, StockName,
StockSymbol, StockQuantity, BuyPrice, TradeDate, OpenPositionID, TeamName) VALUES ('"
& ID & "','" & StockName & "','" & StockSymbol & "','" & StockQuantity & "','" &
StockValue & "','" & BuyDate & "','" & ID & BuyDate & "','" & TeamName & "')"

        cmd.ExecuteNonQuery()

        cmd.CommandText = "INSERT INTO TblTradeHistory (AccountID, StockSymbol,
StockQuantity, BuyOrSell, TradePrice, TradeDate, TeamName, Notes) VALUES ('" & ID &
"','" & StockSymbol & "','" & StockQuantity & "','Buy','" & StockValue & "','" &
BuyDate & "','" & TeamName & "','" & Notes & "')"

        cmd.ExecuteNonQuery()
        ConnectionDb.Close()
    End Sub
End Class
```

The final sub-routine within 'BuyForm' is 'StoreNewPosition'. This is responsible for entering the information of a new trade into two tables, 'tblOpenPositions' and 'tblTradeHistory'.

```
Dim ConnectionDb As New OleDbConnection(AccessDatabaseConnection)
If ConnectionDb.State = ConnectionState.Closed Then ConnectionDb.Open()
```

The connection to the database is opened.

```
Dim cmd As OleDbCommand = ConnectionDb.CreateCommand

cmd.CommandText = "INSERT INTO TblOpenPositions (AccountID, StockName, StockSymbol,
StockQuantity, BuyPrice, TradeDate, OpenPositionID, TeamName) VALUES ('" & ID & "','" &
StockName & "','" & StockSymbol & "','" & StockQuantity & "','" & StockValue & "','" &
BuyDate & "','" & ID & BuyDate & "','" & TeamName & "')"
```

The first SQL command is set. This command adds a new entry to 'tblOpenPositions'. The table has 8 attributes, and therefore the SQL command takes 8 values, taken from the various variables in buy form.

```
cmd.ExecuteNonQuery()
```

The Insert command is executed.

```
cmd.CommandText = "INSERT INTO TblTradeHistory (AccountID, StockSymbol, StockQuantity,
BuyOrSell, TradePrice, TradeDate, TeamName, Notes) VALUES ('" & ID & "','" & StockSymbol
& "','" & StockQuantity & "','Buy','" & StockValue & "','" & BuyDate & "','" & TeamName &
"','" & Notes & "')"
```

Next, a new SQL command is set. This is an Insert statement that makes a new entry in ‘tblTradeHistory’. This takes much the same values as the previous SQL command, but enters the information into a different table.

```
cmd.ExecuteNonQuery()
ConnectionDb.Close()
```

The command is executed and the connection closed.

Investu Server Program – Version 2 – Development 3

The second version of Investu Server Program will peform the same function as in the previous version, but with an integrated alerts-sending system added. Users of the simulation will be able to set price alerts, and receive notifications telling them when the price is reached. In order for this to work when the user is offline, the server program will be constantly running and checking all of the currently active alerts, and then sending an email to the user, when their alert price is reached.

Namespaces/Imports – Investu Server Program – Investu Development 3

```
Imports System.Net.Mail
Imports System.Net
Imports System.IO
Imports System.Xml
Imports System.Data.OleDb
```

The imports for version 2 of Investu Server Program are the same as in the first version in Development 2, with the addition of the ‘System.Net.Mail’ import. This import will allow for the sending of mail from the program.

Global Variables – Investu Server Program – Investu Development 3

```
Public Class InvestuServerProgram

    Dim DBPath As String = "C:\Users\Joe\Documents\visual studio 2010\Investu
Writeup\Development 3\StockInfoDB.mdb"
    Public AccessDatabaseConnection As String = "Provider =
Microsoft.Jet.OLEDB.4.0;Data Source =" & DBPath

    Dim LoopCount As Integer = 0
    Public Symbols As New List(Of String)
```

The variables for the connection to the database are declared, as well as the ‘Symbols’ list which will be read in from a .CSV file containing a list of all FTSE100 stock symbols. A variable for counting the loops of the timer within the program is also declared.

InvestuServerProgram_Load – Investu Server Program – Investu Development 3

```
    Private Sub InvestuServerProgram_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
        RunningStoppedLabel.Text = "STOPPED"
        RunningStoppedLabel.ForeColor = Color.Red
End Sub
```

For this version, all that happens on load of the program is formatting. This formatting of the display lets the user know the current state of the program.

InvestuServerProgram_Load – Investu Server Program – Investu Development 3

```
Function GetFilePath()

        Dim FilePath As String
        Dim FileDialog As New OpenFileDialog
        FileDialog.InitialDirectory = Application.StartupPath
        If FileDialog.ShowDialog() = Windows.Forms.DialogResult.OK Then
            FilePath = FileDialog.FileName
        End If
        Return FilePath

End Function
```

The second version of the server program allows the user to select their own file paths for the database and stock symbol CSV file. This means that the program can be transported to other systems easily.

```
Dim FilePath As String
Dim FileDialog As New OpenFileDialog
```

Two varaibles are declared: a string for holding the resulting file path, and an ‘OpenFileDialog’ to allow the user to browse their system for files.

```
If FileDialog.ShowDialog() = Windows.Forms.DialogResult.OK Then
      FilePath = FileDialog.FileName
End If
```

This conditional opens a new file browser, and then waits for the user to press the ‘OK’ button. If they do, then the value of ‘FilePath’ is set to the currently selected path in the file browser.

Return FilePath

The value of the variable ‘FilePath’ is returned. This function can now be used anywhere that a file path is needed.

StartButton_Click – Investu Server Program – Investu Development 3

```
  Private Sub StartButton_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles StartButton.Click
        PopulateSymbolArray(GetFilePath())
        Timer1.Start()
        Timer2.Start()
        RunningStoppedLabel.Text = "RUNNING"
        RunningStoppedLabel.ForeColor = Color.Green
End Sub
```

```
PopulateSymbolArray(GetFilePath())
```

When the user opts to run the program, they must first select the list of stock symbols they want to us. ‘PopulateSymbolArray’ is a sub-routine that takes a single argument: a file path for a .CSV file containing stock symbols. This file is then processed, and all of the stock symbols are extracted into a list. Then, the program searches for the latest stock information regarding these symbols. It is therefore vital that this list is up to date and accurate, and so the ability to select a new list every time the program is run is an important feature.

```
Timer1.Start()
Timer2.Start()
```

Two timers are started that control different aspects of the program. These timers will be explained later on.

```
RunningStoppedLabel.Text = "RUNNING"
RunningStoppedLabel.ForeColor = Color.Green
```

The visual display is updated to inform the user that the program is running

FillDB_Load – Investu Server Program – Investu Development 3

```
Sub FillDB()

        PopulateSymbolArray(GetFilePath())

        Dim ConnectionDb As New OleDbConnection(AccessDatabaseConnection)
        If ConnectionDb.State = ConnectionState.Closed Then ConnectionDb.Open()

        Dim cmd As OleDbCommand = ConnectionDb.CreateCommand

        cmd.CommandText = "DELETE * FROM tblStockDetails"
        cmd.ExecuteNonQuery()

        For L = 0 To 99
            cmd.CommandText = "INSERT INTO tblStockDetails (StockSymbol, StockName,
Price, Change) VALUES ('" & Symbols(L) & "','" & GetStockName(Symbols(L)) & "','" &
GetStockPrice(Symbols(L)) & "','" & GetStockChange(Symbols(L)) & "')"
            cmd.ExecuteNonQuery()
        Next
        ConnectionDb.Close()
    End Sub
```

One of the new features in Development 3 of the main program is a display showing the data of all of the stocks in the program in a single tab, so that they can easily be viewed. This table is effectively a list of all stock symbols, and their latest price and change values. However, because the stock symbols list can change, there needs to be a way of updating this table. 'FillDB' is responsible for wiping the information from this table and then re-writing it, with the update list of stock symbols, and their respective price information.

Because this is an intensive process, it will only be run on the click of a button, and not automatically. Once it has been run once, the prices for each stock can be kept up to date by using an 'Update' SQL command every time new price data is retrieved for the table 'tblStockPriceHistory'

```
PopulateSymbolArray(GetFilePath())
```

The file pathway of the symbol list is retrieved, and the list 'Symbols' is updated.

```
Dim ConnectionDb As New OleDbConnection(AccessDatabaseConnection)
If ConnectionDb.State = ConnectionState.Closed Then ConnectionDb.Open()

Dim cmd As OleDbCommand = ConnectionDb.CreateCommand
```

The database is connected to, and a new command is created.

```
cmd.CommandText = "DELETE * FROM tblStockDetails"
cmd.ExecuteNonQuery()
```

This is responsible for first wiping the table before new stock details are written in. It is immediately executed.

```
        For L = 0 To 99
            cmd.CommandText = "INSERT INTO tblStockDetails (StockSymbol, StockName, Price,
Change) VALUES ('" & Symbols(L) & "','" & GetStockName(Symbols(L)) & "','" &
GetStockPrice(Symbols(L)) & "','" & GetStockChange(Symbols(L)) & "')"
            cmd.ExecuteNonQuery()
        Next
```

Then, a For-Loop is initiated, that loops 100 times, because there are 100 stock symbols in the FTSE100. For each loop of the For-Loop, a new line is written to the table. The values inserted are the symbol, and then the name, price and change values, retrieved from the 'GetStock' functions.

FillDBButton_Click – Investu Server Program – Investu Development 3

```
Private Sub FillDBButton_Click(sender As System.Object, e As System.EventArgs) Handles
FillDBButton.Click
        If Not BackgroundWorker.IsBusy Then
            BackgroundWorker.RunWorkerAsync()
        End If
    End Sub
```

There are a total of 300 internet queries made every time the database if filled: 3 for each of the 100 stock symbols. This is obviously highly costly in terms of computer resources and so can cause the program to slow down and stop the program from doing its main function: fetching stock price data and writing it to the database.

Therefore, when 'FillDB' is called, it is done via a background worker. A background worker is a VB control that performs tasks in the background – it allows the program to run an operation on a seperate, dedicated thread, which stops the user interface from appearing to stop responding.

BackgroundWorker_DoWork – Investu Server Program – Investu Development 3

```
Private Sub BackgroundWorker_DoWork(sender As System.Object, e As
System.ComponentModel.DoWorkEventArgs) Handles BackgroundWorker.DoWork
        FillDB()
    End Sub
```

When the background worker is called, the 'FillDB' sub-routine is called, and the process of filling the 'tblStockDetails' table with stock infromation begins.

PopulateSymbolArray – Investu Server Program – Investu Development 3

```
Public Sub PopulateSymbolArray(ByVal FilePath As String)
```

This sub-routine is the same as found in Development 2 on page 135.

Timer1_Tick – Investu Server Program – Investu Development 3

```
    Private Sub Timer1_Tick(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Timer1.Tick
```

This sub-routine is the same as found in Development 2 on page 109.

FetchLatestStockInfo – Investu Server Program – Investu Development 3

```
Sub FetchLatestStockInfo()
```

This sub-routine is the same as found in Development 2 on page 110.

FormatString – Investu Server Program – Investu Development 3

```
Function FormatString(ByVal A As Integer, B As String, C As String, D As String)
```

This sub-routine is the same as found in Development 2 on page 112.

UpdateDatabase – Investu Server Program – Investu Development 3

```
    Sub UpdateDatabase(ByVal StockSymbol As String, ByVal StockPrice As Decimal, ByVal
StockChange As Decimal)

        Dim ConnectionDb As New OleDbConnection(AccessDatabaseConnection)
        If ConnectionDb.State = ConnectionState.Closed Then ConnectionDb.Open()

        Dim cmd As OleDbCommand = ConnectionDb.CreateCommand

        cmd.CommandText = "INSERT INTO tblStockPriceHistory (StockSymbol, StockPrice,
FetchTime, FetchDate) VALUES ('" & StockSymbol & "','" & StockPrice & "','" &
TimeOfDay & "','" & Date.Now & "')"
        cmd.ExecuteNonQuery()

       cmd.CommandText = "UPDATE tblStockDetails SET Price=" & StockPrice & ",
Change=" & StockChange & " WHERE StockSymbol='" & StockSymbol & "'"
        cmd.ExecuteNonQuery()

        ConnectionDb.Close()

    End Sub
```

This sub-routine is the same as the first version of Investu Server Program, with the addition of this code:

```
      cmd.CommandText = "UPDATE tblStockDetails SET Price=" & StockPrice & ", Change=" &
StockChange & " WHERE StockSymbol='" & StockSymbol & "'"
        cmd.ExecuteNonQuery()
```

The sub-routine now not only adds a new entry to ‘tblStockPriceHistory’, but also updates the value of the current price within ‘tblStockDetails’. ‘tblStockDetails’ is responsible for populating a display with the information of all stocks in the FTSE100, and so by updating every time a new price is fetched, we have the most chance of keeping the data within that table accurate.

SplitStockInfo – Investu Server Program – Investu Development 3

```
Function SplitStockInfo(ByVal StringToSplit As String, ByVal DetailsToExtract As
String)
```

This sub-routine is the same as found in the Testing section of Development 1 on page 96.

CheckAlerts – Investu Server Program – Investu Development 3

```
    Sub CheckAlerts()
        Dim ConnectionDb As New OleDbConnection(AccessDatabaseConnection)
        If ConnectionDb.State = ConnectionState.Closed Then ConnectionDb.Open()

        Dim cmd As OleDbCommand = ConnectionDb.CreateCommand

        cmd.CommandText = "SELECT * FROM tblAlerts"

        Dim SQLReply As OleDbDataReader = cmd.ExecuteReader

        For Each Record In SQLReply

            If Record.item("UpOrDown") = "UP" And
GetStockPrice(Record.item("StockSymbol")) > Record.item("AlertPrice") Then

                SendAlert(Record.item("ID"), Record.item("AccountID"),
Record.item("StockSymbol"), Record.item("AlertPrice"), Record.item("TeamName"))

            ElseIf Record.item("UpOrDown") = "DOWN" And
GetStockPrice(Record.item("StockSymbol")) < Record.item("AlertPrice") Then

                SendAlert(Record.item("ID"), Record.item("AccountID"),
Record.item("StockSymbol"), Record.item("AlertPrice"), Record.item("TeamName"))
            End If
            FetchAlerts()
        Next
        ConnectionDb.Close()
    End Sub
```

The database will contain a list of alerts that have been set by users. These alerts need to activate and the user needs to be notified, when the price they specified is reached. This means that the alert prices and the actual prices of stocks need to be constantly checked to see if the alert price has been reached.

```
Dim ConnectionDb As New OleDbConnection(AccessDatabaseConnection)
If ConnectionDb.State = ConnectionState.Closed Then ConnectionDb.Open()

Dim cmd As OleDbCommand = ConnectionDb.CreateCommand
```

In order to do this, a new connection is initiated, and an SQL command is created.

```
SELECT * FROM tblAlerts
```

The SQL command selects every single alerts from the table.

```
For Each Record In SQLReply
   ....
Next
```

These alerts are then looped through.

```
            If Record.item("UpOrDown") = "UP" And
GetStockPrice(Record.item("StockSymbol")) > Record.item("AlertPrice") Then

                SendAlert(Record.item("ID"), Record.item("AccountID"),
Record.item("StockSymbol"), Record.item("AlertPrice"), Record.item("TeamName"))
```

Inside the For-Each loop, a conditional checks the status of the alerts.

Alerts can be ‘Up’ alerts, or ‘Down’ alerts. ‘Up’ are alerts such that the stock price is below the alert price, and the user wants a notification when the current price goes OVER their alert price. ‘Down’ alerts are alerts such that the stock price is above the alert price, and the user wants a notification when the current price goes BELOW their alert price. This is important as it determines which logical comparison is made.

If the alert is an ‘Up’ alert, then...

```
GetStockPrice(Record.item("StockSymbol")) > Record.item("AlertPrice")
```

... checks whether or not the current price, retrieved with ‘GetStockPrice’, is greater than the alert price. If this is the case, then the sub-routine ‘SendAlert’ is called, and passed 5 arguments.

```
ElseIf Record.item("UpOrDown") = "DOWN" And GetStockPrice(Record.item("StockSymbol")) <
Record.item("AlertPrice") Then

                SendAlert(Record.item("ID"), Record.item("AccountID"),
Record.item("StockSymbol"), Record.item("AlertPrice"), Record.item("TeamName"))
```

However, if the alert is ‘Down’ alert then the comparison...

```
GetStockPrice(Record.item("StockSymbol")) < Record.item("AlertPrice") Then
```

... checks whether or not the current price is below the alert price. If this is the case, then an alert is sent to the user, using the same ‘SendAlert’ sub-routine as before.

```
FetchAlerts()
```

Once the alert is called, it will be deleted from the list, so ‘FetchAlerts’ is called to update the visual display of currently active alerts.

SendAlert – Investu Server Program – Investu Development 3

```
Sub SendAlert(ByVal AlertID As Integer, ByVal AccountID As Integer, ByVal StockSymbol
As String, ByVal AlertPrice As Integer, ByVal TeamName As String)

        Dim Mail As New MailMessage()

        Mail.From = New MailAddress("InvestuAlerts@outlook.com")
        Mail.[To].Add(GetEmailUsingID(AccountID))

        Mail.Subject = "Investu Alert"
        Mail.Body = "This is an alert for " & GetStockName(StockSymbol) & ". The stock
has reached the price of £" & AlertPrice / 100 & " has been reached. Login to your
Investu account to take further action."

        Dim SMTP As New SmtpClient()
        SMTP.Host = "smtp.live.com"
        SMTP.Credentials = New NetworkCredential("investualerts@outlook.com",
"Alerts@Investu")
        SMTP.EnableSsl = True


        Try
            SMTP.Send(Mail)
        Catch exc As Exception
            StoreCrashInfo(exc.ToString(), DateTime.Now)
        End Try
        DeleteAlert(AlertID)
    End Sub
```

‘SendAlert’ sends the notification to the user telling them that their alert has been reached. The sub-routine makes use of the ‘System.Net.Mail’ import to send an email to the user.

```
Dim Mail As New MailMessage()
```

A new instance of the ‘MailMessage’ class is initiated.

```
Mail.From = New MailAddress("InvestuAlerts@outlook.com")
Mail.[To].Add(GetEmailUsingID(AccountID))
Mail.Subject = "Investu Alert for " & GetStockName(StockSymbol) & ""
Mail.Body = "This is an alert for " & GetStockName(StockSymbol) & ". The stock has
reached the price of £" & AlertPrice / 100 & " has been reached. Login to your Investu
account to take further action."
```

4 values are given to the new mail. The sender, receiver, subject, and body. The ‘To’ value is the result of a function called ‘GetEmailUsingID’ which uses the ID of the account which wrote the alert, and then uses it to find the email of that account.

```
        Dim SMTP As New SmtpClient()
        SMTP.Host = "smtp.live.com"
        SMTP.Credentials = New NetworkCredential("investualerts@outlook.com",
"Alerts@Investu")
        SMTP.EnableSsl = True
```

Once the message itself is composed, the sending details are set. ‘SMTP’ stands for ‘Simple Mail Transfer Protocol’, and is a set of rules for sending mail. A new SMTP client is initiated, through which the mail will be sent. The client is given a host, which is ‘smtp.live.com’ in this case, as the Investu email was made with Outlook, which uses the Live servers.

The credentials of the sender are then set. The ‘NetworkCredential’ class takes two arguments: an email and password. The email and password are for Investu email account.

```
SMTP.EnableSsl = True
```

SSL is enabled to ensure that the communication is encrypted.

```
        Try
            SMTP.Send(Mail)
            DeleteAlert(AlertID)
        Catch exc As Exception
            StoreCrashInfo(exc.ToString(), DateTime.Now)
        End Try
```

Once the mail is composed and the sending information is set, the sub-routine tries to send the mail. This is wrapped in a Try-Catch as it can be prone to errors. If the send is successful, then the user will receive an email notification with information regarding their alert.

If the send is not successful, the exeception that is thrown is stored in the database and the program continues.

The alert is then deleted using the sub-routine ‘DeleteAlert’.

DeleteAlert – Investu Server Program – Investu Development 3

```
Sub DeleteAlert(ByVal AlertID As Integer)

        Dim ConnectionDb As New OleDbConnection(AccessDatabaseConnection)
        If ConnectionDb.State = ConnectionState.Closed Then ConnectionDb.Open()
        Dim cmd As OleDbCommand = ConnectionDb.CreateCommand
        cmd.CommandText = "DELETE * FROM tblAlerts WHERE ID=" & AlertID & ""
        cmd.ExecuteNonQuery()
        ConnectionDb.Close()
    End Sub
```

```
DELETE * FROM tblAlerts WHERE ID=" & AlertID & "
```

The sub-routine connects to the database and uses the above SQL command to remove all trace of the alert.

FetchAlerts – Investu Server Program – Investu Development 3

```
Sub FetchAlerts()

        Dim MyConnection As OleDbConnection
        Dim Adapter As OleDbDataAdapter
        Dim DataSet As DataSet
        Dim Tables As DataTableCollection
        Dim Source As New BindingSource

        MyConnection = New OleDbConnection
        MyConnection.ConnectionString = AccessDatabaseConnection
        DataSet = New DataSet
        Tables = DataSet.Tables
        Adapter = New OleDbDataAdapter("SELECT * FROM tblAlerts", MyConnection)
        Adapter.Fill(DataSet, "tblAlerts")
        Dim View As New DataView(Tables(0))
        Source.DataSource = View
        AlertsGrid.DataSource = View

        AlertsGrid.Columns(0).Width = 60
        AlertsGrid.Columns(1).Width = 60
        AlertsGrid.Columns(2).Width = 90
        AlertsGrid.Columns(3).Width = 90
        AlertsGrid.Columns(4).Width = 60
        AlertsGrid.Columns(5).Width = 90

    End Sub
```

‘FetchAlerts’ is a purely cosmetic sub-routine, that creates a display with all alerts in the user interface for the server program. The sub-routine is not necessary for the function of the program, but can be useful for seeing which alerts are active, for debugging and testing purposes.

```
Adapter = New OleDbDataAdapter("SELECT * FROM tblAlerts", MyConnection)
Adapter.Fill(DataSet, "tblAlerts")
```

An instance of the ‘OleDbDataAdapter’ class is initiated, with a Select-all query passed as an argument. The data set is then filled with matching data from ‘tblAlerts’ which is written to a data grid view control within the program.

For a more detailed description of this process refer to page 212.

GetEmailUsingID – Investu Server Program – Investu Development 3

```
    Function GetEmailUsingID(ByVal AccountID)

        Dim Email As String = ""

        Dim ConnectionDb As New OleDbConnection(AccessDatabaseConnection)
        If ConnectionDb.State = ConnectionState.Closed Then ConnectionDb.Open()

        Dim cmd As OleDbCommand = ConnectionDb.CreateCommand
        cmd.CommandText = "SELECT Email FROM tblUserInfo WHERE AccountID=" & AccountID
& ""
        Dim SQLReply As OleDbDataReader = cmd.ExecuteReader

        For Each Record In SQLReply
            Email = Record.item("Email")
        Next

        ConnectionDb.Close()
        Return Email

    End Function
```

This sub-routine is a simple select SQL statement that assigns the variable ‘Email’ a value from the ‘Email’ column in the database, where the accountID attribute in the table, and the accountID variable within the program match.

GetStockChange – Investu Server Program – Investu Development 3

```
    Function GetStockChange(ByVal StockSymbol As String)
```

This sub-routine is from Developmemnt 2. A full description and the code can be found on page 115.

GetStockPrice – Investu Server Program – Investu Development 3

```
    Function GetStockPrice(ByVal StockSymbol As String)
```

This sub-routine is from Developmemnt 2. A full description and the code can be found on page 117.

GetStockName – Investu Server Program – Investu Development 3

```
Function GetStockName(ByVal StockSymbol As String)
```

This sub-routine is from Developmemnt 2. A full description and the code can be found on page 116.

StoreCrashInfo – Investu Server Program – Investu Development 3

```
    Sub StoreCrashInfo(ByVal CrashMsg, ByVal CrashTime)
```

This sub-routine is from Developmemnt 2. A full description and the code can be found on page 118.

Timer2_Tick – Investu Server Program – Investu Development 3

```
Private Sub Timer2_Tick(sender As System.Object, e As System.EventArgs) Handles
Timer2.Tick
        Timer2.Interval = 60000
        CheckAlerts()
End Sub
```

This sub-routine is called once every 60 seconds, and calls the sub-routine ‘CheckAlerts’.

DBPathButton_Click – Investu Server Program – Investu Development 3

```
Private Sub DBPathButton_Click(sender As Object, e As EventArgs) Handles Button4.Click
        DBPath = GetFilePath()
    End Sub
```

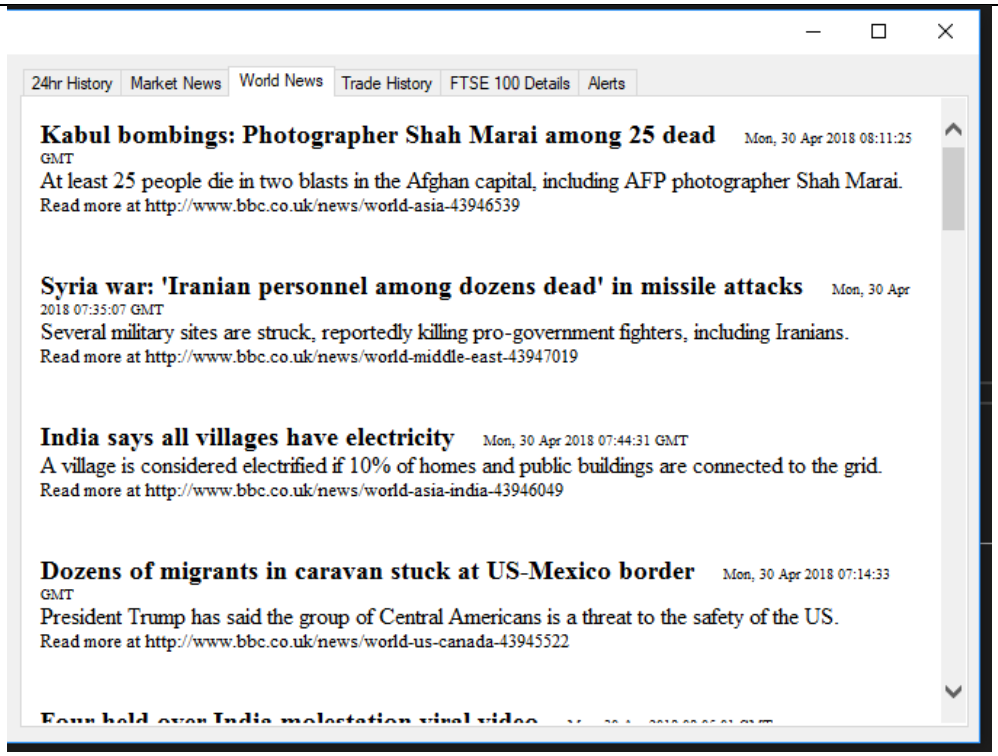
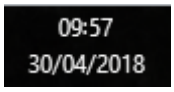
Calls the function ‘GetFilePath’ to allow the user to change the file pathway of the database.

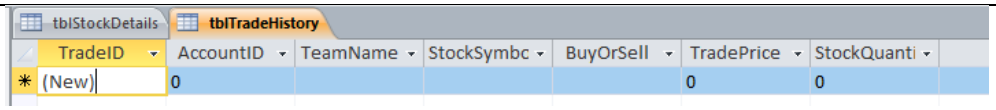
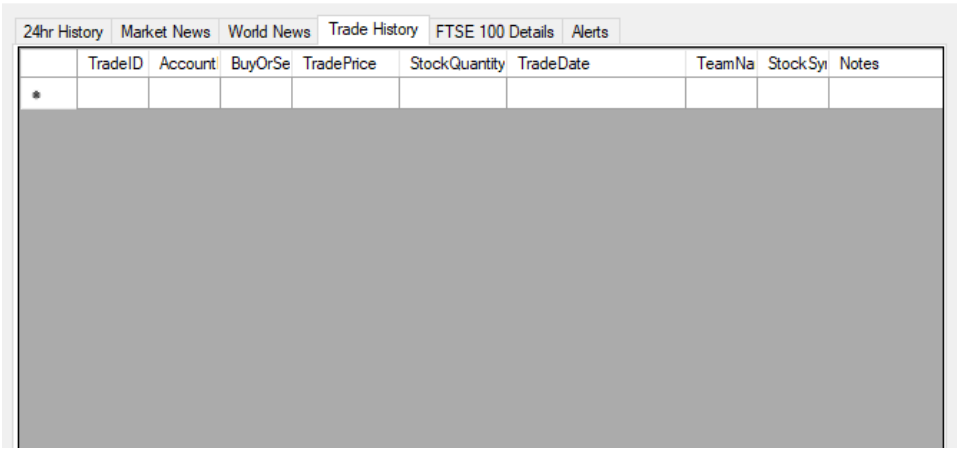
Testing 1 – Investu Simulation – Development 3

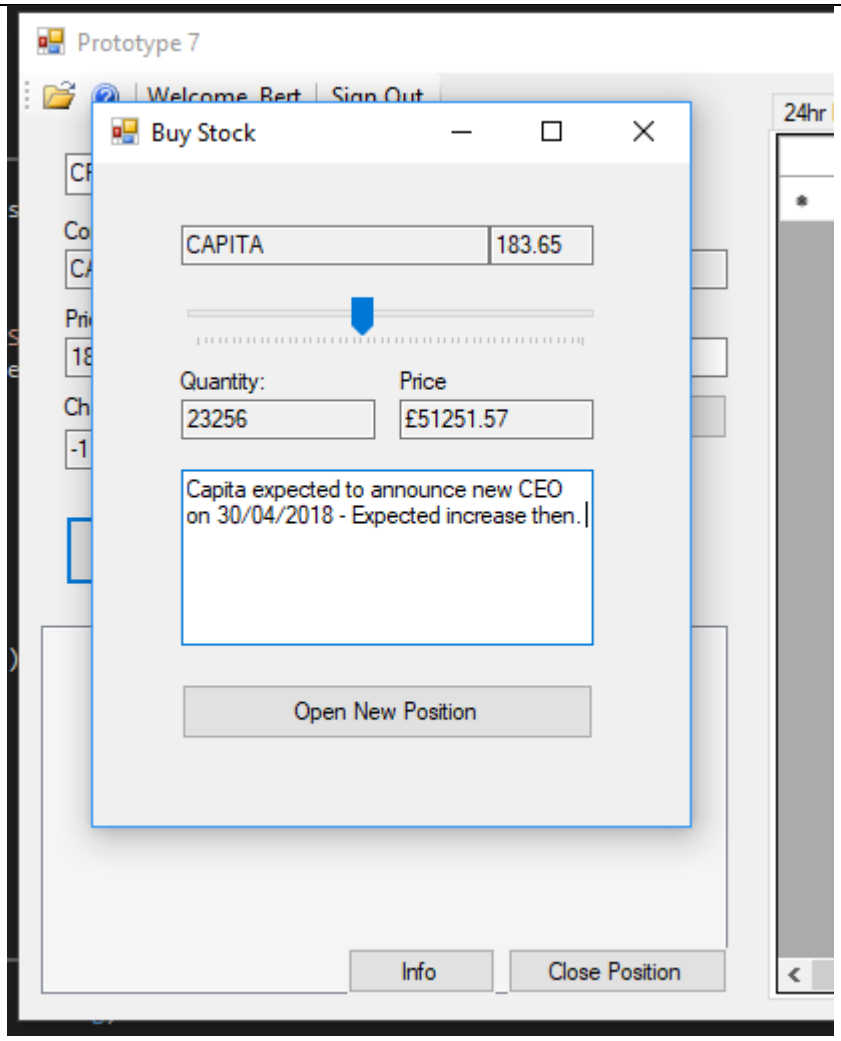
Every sub-routine has been tested individually to ensure it works independently, however these tests have not been shown. The tests displayed are tests that show multiple sub-routines and functions working together to produce the desired outcome.

Many of the features of the simulation that haven't been changed since Development 2 are not tested here. To see the tests that are not displayed here, refer to Testing in Development 2, on page 163.

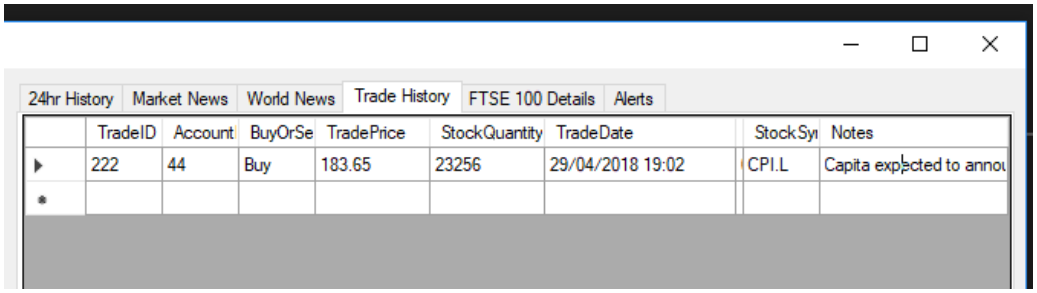
MainForm Testing 1

Show that the user can view world news and market news within the simulation.	 Loading the program and clicking the 'World News' tab displays the above information. Note that the current time is shown below:  The last story was posted at 8:11am (top right corner of first story). This shows that the news is fairly recent. The market news section uses the same code and such we can assume it works. This indicates that the requirements set out in the test description have been met and

	the test has therefore been passed.	
Show that when the user executes a trade, the database table 'tblTradeHistory' updates with a new entry, and the 'Trade History' tab in the tab control updates with the new trade.	 As shown above, the database is currently empty. There are no trades in 'tblTradeHistory'. Within the simulation, the tab containing trade history is also blank:  A new trade is made, as shown below:	

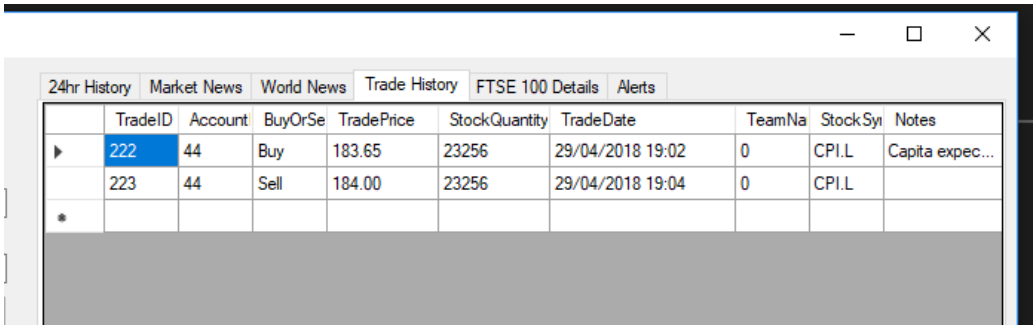


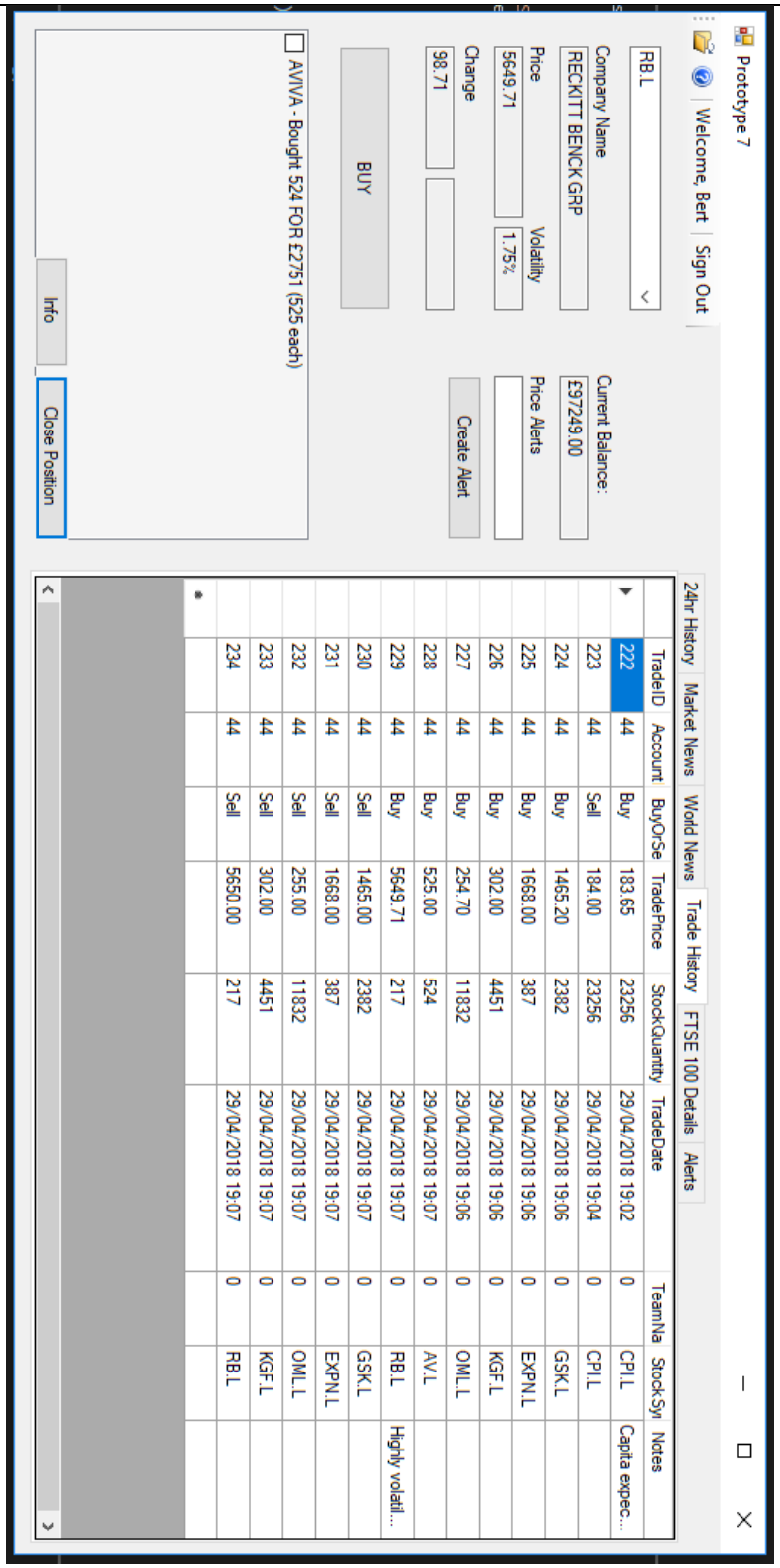
Making a new trade offer, with the details above, results in the results below:



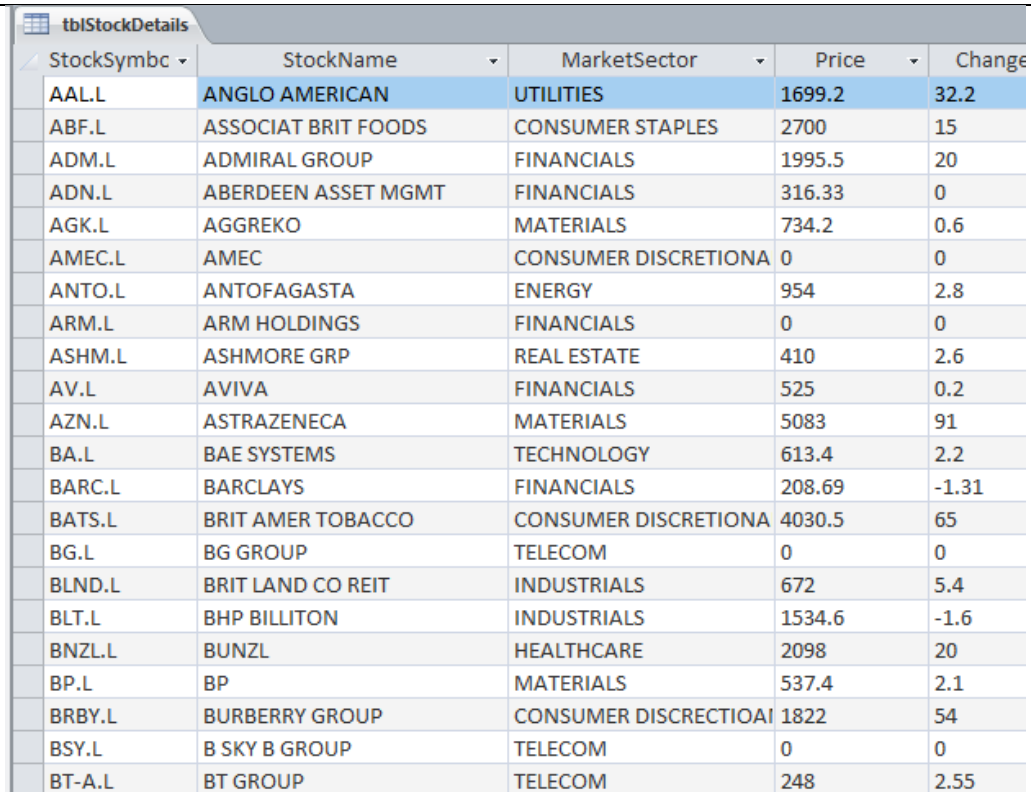
The trade has been written into the 'Trade History' display in the tab control.

The database is also updated, as shown below:

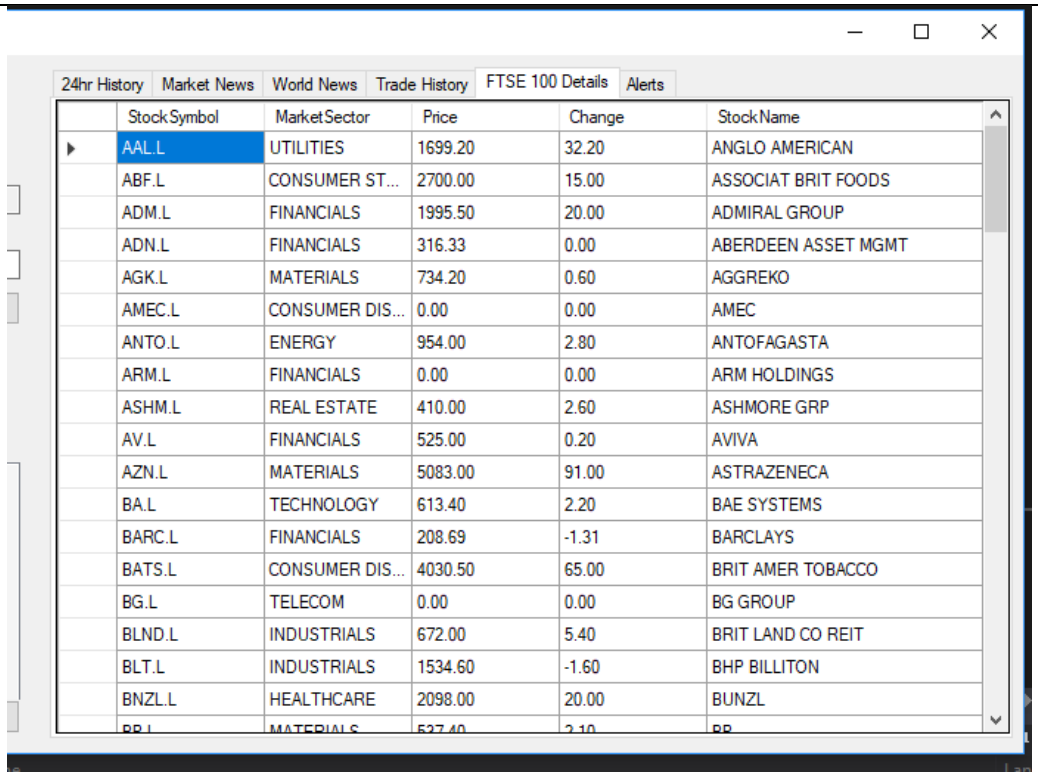
	tblStockDetails | tblTradeHistory TradeID | AccountID | TeamName | StockSymbc | BuyOrSell | TradePrice | StockQ 222 | 44 | 0 | CPI.L | Buy | 183.65 | 23256 Upon closing the position, the data grid view is again updated:  24hr History | Market News | World News | Trade History | FTSE 100 Details | Alerts TradeID | Account | BuyOrSe | TradePrice | StockQuantity | TradeDate | TeamNa | StockSyi | Notes 222 | 44 | Buy | 183.65 | 23256 | 29/04/2018 19:02 | 0 | CPI.L | Capita expec... 223 | 44 | Sell | 184.00 | 23256 | 29/04/2018 19:04 | 0 | CPI.L Reloading the program results in the same display as above – the trade history information is retained forever. This indicates that the requirements set out in the test description have been met and the test has therefore been passed.	
Show that the trade history display shows all items in the users trade history.	The account has been used for a few minutes, making buy offers and then selling the stock immediately. The following is the result:	



Notice how there is an entry for a buy offer for ‘AV.L’ but not sale offer – that is because the position is still open, as show on the left hand side of the simulation in the ‘Open Positions’ display.

	This indicates that the requirements set out in the test description have been met and the test has therefore been passed.	
Show that the details grid displays all of the information within 'tblStockDetails'.	tblStockDetails  Above is the contents of the table 'tblStockDetails'. This table is a visual display of the information for the details of all companies in the FTSE100.	

StockSymbc	StockName	MarketSector	Price	Change
AAL.L	ANGLO AMERICAN	UTILITIES	1699.2	32.2
ABF.L	ASSOCIAT BRIT FOODS	CONSUMER STAPLES	2700	15
ADM.L	ADMIRAL GROUP	FINANCIALS	1995.5	20
ADN.L	ABERDEEN ASSET MGMT	FINANCIALS	316.33	0
AGK.L	AGGREKO	MATERIALS	734.2	0.6
AMEC.L	AMEC	CONSUMER DISCRETIONA	0	0
ANTO.L	ANTOFAGASTA	ENERGY	954	2.8
ARM.L	ARM HOLDINGS	FINANCIALS	0	0
ASHM.L	ASHMORE GRP	REAL ESTATE	410	2.6
AV.L	AVIVA	FINANCIALS	525	0.2
AZN.L	ASTRAZENECA	MATERIALS	5083	91
BA.L	BAE SYSTEMS	TECHNOLOGY	613.4	2.2
BARC.L	BARCLAYS	FINANCIALS	208.69	-1.31
BATS.L	BRIT AMER TOBACCO	CONSUMER DISCRETIONA	4030.5	65
BG.L	BG GROUP	TELECOM	0	0
BLND.L	BRIT LAND CO REIT	INDUSTRIALS	672	5.4
BLT.L	BHP BILLITON	INDUSTRIALS	1534.6	-1.6
BNZL.L	BUNZL	HEALTHCARE	2098	20
BP.L	BP	MATERIALS	537.4	2.1
BRBY.L	BURBERRY GROUP	CONSUMER DISCRECTIOAI	1822	54
BSY.L	B SKY B GROUP	TELECOM	0	0
BT-A.L	BT GROUP	TELECOM	248	2.55

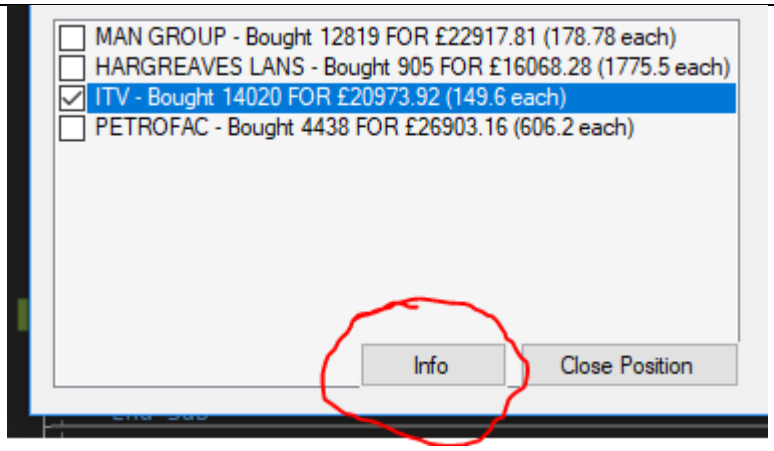


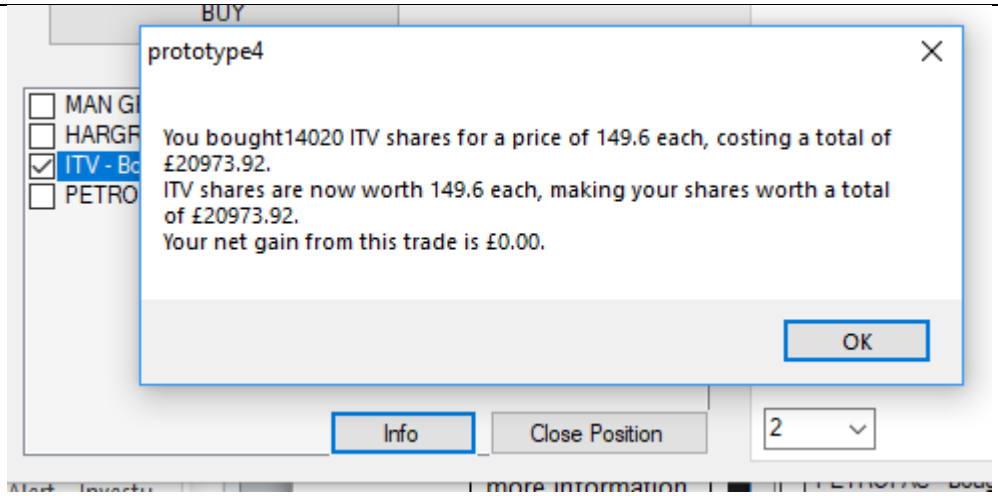
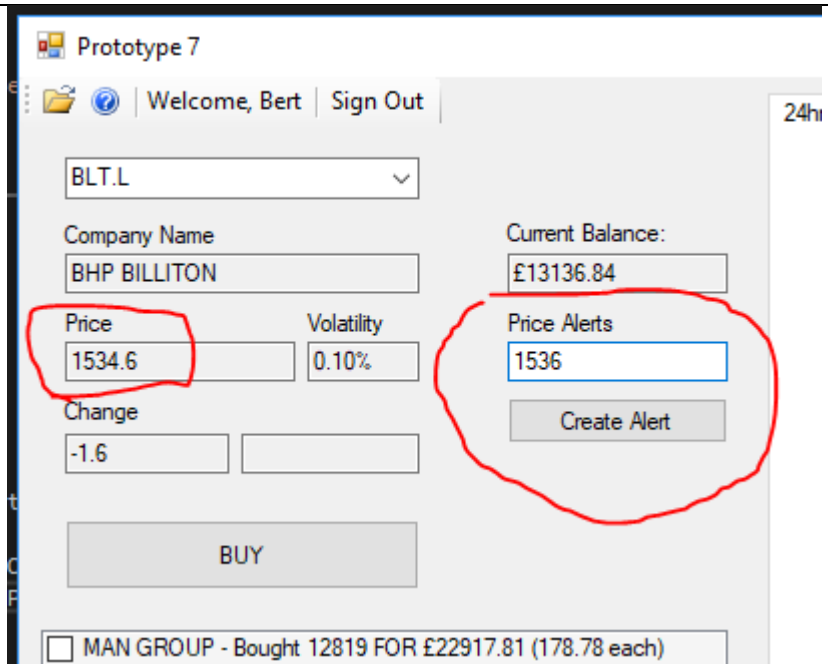
Stock Symbol	Market Sector	Price	Change	Stock Name
AAL.L	UTILITIES	1699.20	32.20	ANGLO AMERICAN
ABF.L	CONSUMER ST...	2700.00	15.00	ASSOCIAT BRIT FOODS
ADM.L	FINANCIALS	1995.50	20.00	ADMIRAL GROUP
ADN.L	FINANCIALS	316.33	0.00	ABERDEEN ASSET MGMT
AGK.L	MATERIALS	734.20	0.60	AGGREKO
AMEC.L	CONSUMER DIS...	0.00	0.00	AMEC
ANTO.L	ENERGY	954.00	2.80	ANTOFAGASTA
ARM.L	FINANCIALS	0.00	0.00	ARM HOLDINGS
ASHM.L	REAL ESTATE	410.00	2.60	ASHMORE GRP
AV.L	FINANCIALS	525.00	0.20	AVIVA
AZN.L	MATERIALS	5083.00	91.00	ASTRAZENECA
BA.L	TECHNOLOGY	613.40	2.20	BAE SYSTEMS
BARC.L	FINANCIALS	208.69	-1.31	BARCLAYS
BATS.L	CONSUMER DIS...	4030.50	65.00	BRIT AMER TOBACCO
BG.L	TELECOM	0.00	0.00	BG GROUP
BLND.L	INDUSTRIALS	672.00	5.40	BRIT LAND CO REIT
BLT.L	INDUSTRIALS	1534.60	-1.60	BHP BILLITON
BNZL.L	HEALTHCARE	2098.00	20.00	BUNZL

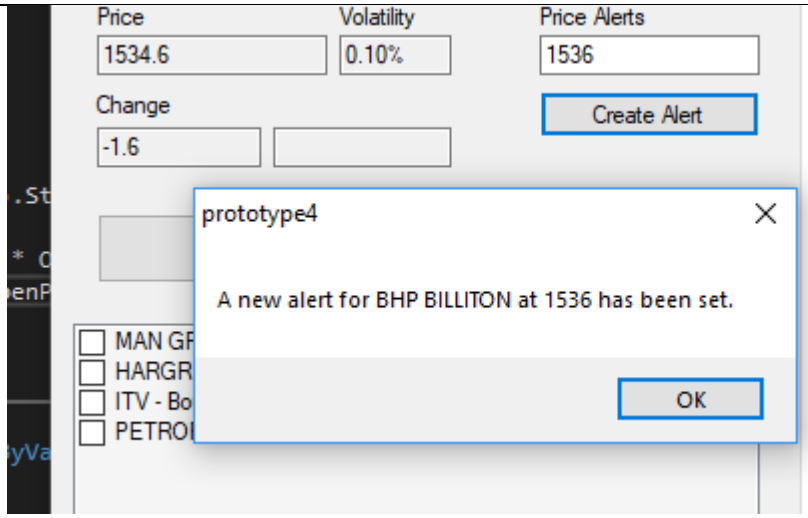
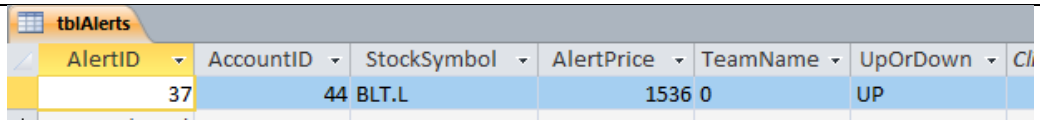
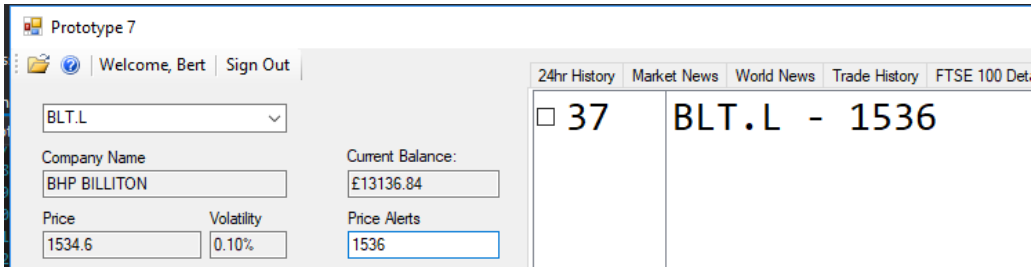
Within the simulation, under the tab ‘FTSE100 Details’, the data is written a gridview.

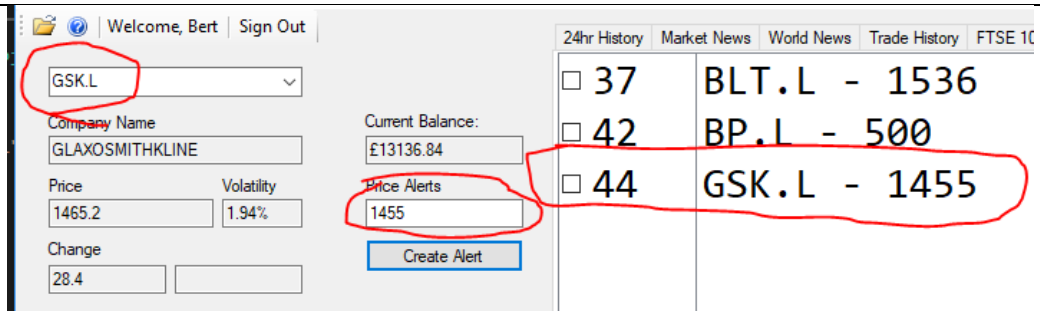
This indicates that the requirements set out in the test description have been met and the test has therefore been passed.

Show that the user can click ‘Info’ and see more information relating to their trade.



	 As the trade has only just been made, the net gain is quoted as £0. If this position was left open for a while during a weekday, it would state a positive or negative profit depending on whether the stock went up or down in price. This indicates that the requirements set out in the test description have been met and the test has therefore been passed.
Show that when the user uses valid data, they can make a new alert.	 The price of BHP BILLITON shares is 1534.6. A valid price alert is any integer not equal to the current price, and not 0. Therefore, 1536 is chosen, as shown in the red circle on the right.

	 Clicking 'Create Alert' results in the above dialogue. This indicates that the requirements set out in the test description have been met and the test has therefore been passed.
Show that the alert is written to the database, in the table 'tblAlerts', and that the alerts display updates to show the new alert.	 Above shows the table 'tblAlerts' within the database. The alert has been added to the table.  The alert has been added to the 'Alerts' tab in the tab control. Adding more alerts updates this display:



The database now has the following entries in 'tblAlerts':

tblAlerts

AlertID	AccountID	StockSymbol	AlertPrice	TeamName	UpOrDown
37	44	BLT.L	1536	0	UP
42	44	BP.L	500	0	DOWN
44	44	GSK.L	1455	0	DOWN
(New)					

This indicates that the requirements set out in the test description have been met and the test has therefore been passed.

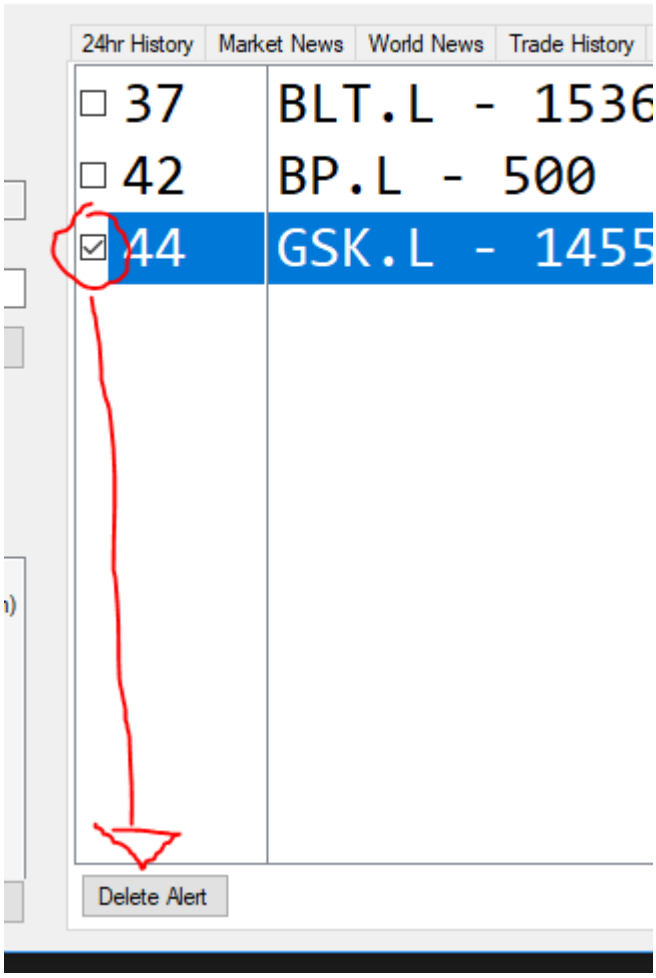
Show that the user can delete an alert, and that the database and display will update to show the change.

tblAlerts

AlertID	AccountID	StockSymbol	AlertPrice	TeamName	UpOrDown
37	44	BLT.L	1536	0	UP
42	44	BP.L	500	0	DOWN
44	44	GSK.L	1455	0	DOWN
(New)					

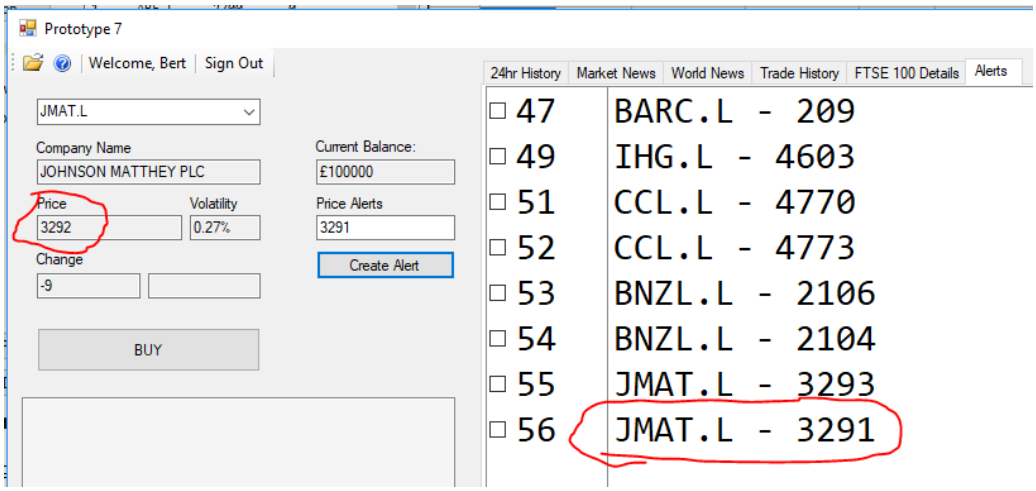
The database has the content shown in the above screenshot.

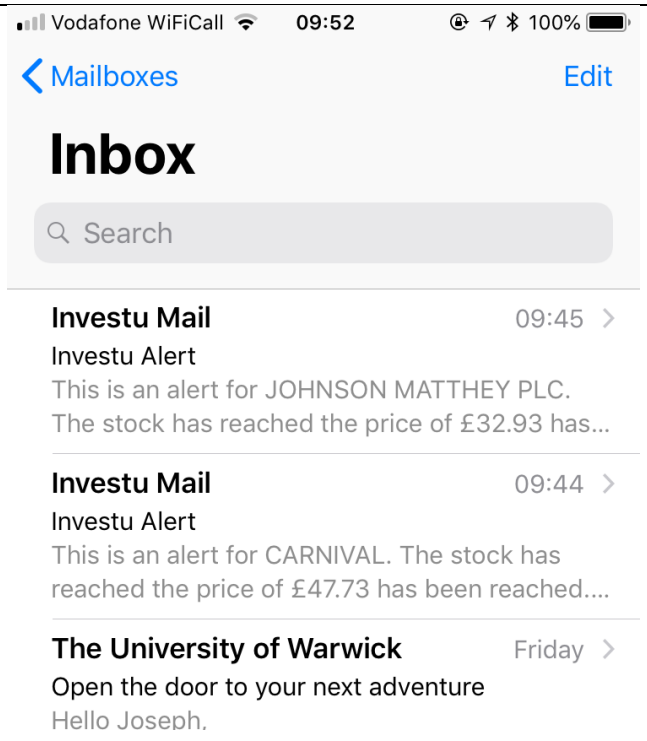
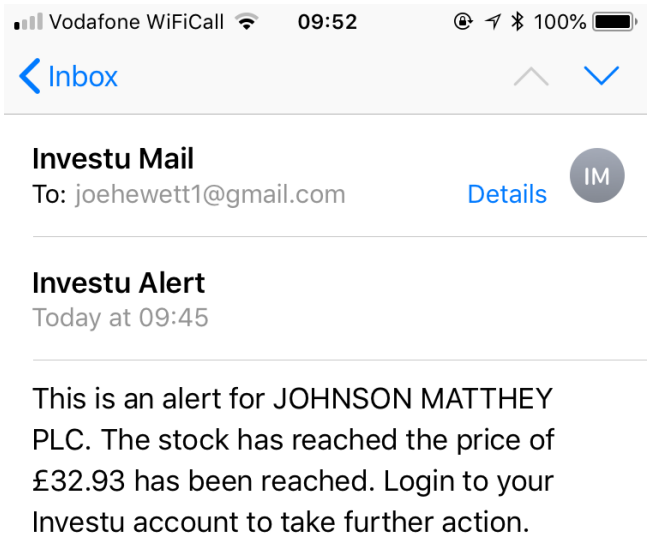
We will go into the simulation, select an alert and then click 'Delete'.



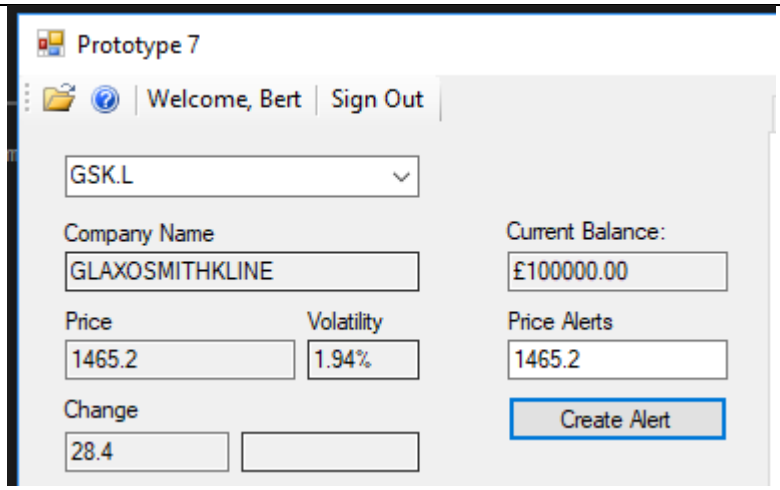
Clicking delete then causes the display to update, with the alert removed, as shown below:

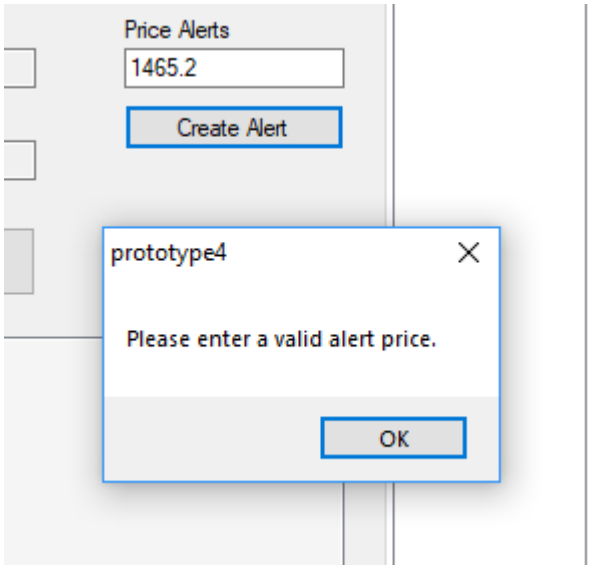
The database now only has two alerts in, as opposed to three at the start of the test:

	tblAlerts AlertID | AccountID | StockSymbol | AlertPrice | TeamName | UpOrDown 37 | 44 | BLT.L | 1536 | 0 | UP 42 | 44 | BP.L | 500 | 0 | DOWN #Deleted | #Deleted | #Deleted | #Deleted | #Deleted | #Deleted (New) This indicates that the requirements set out in the test description have been met and the test has therefore been passed.
Show that alerts execute when the price of the stock exceeds the alert price, and that the user is notified by email.	Some alerts are created on an account called 'Bert'. This account included an email when it was created. The account creates a series of alerts, with the alert price very close to the actual price. This will mean that even a small change in price should result in an alert. The following screenshot shows the alerts that have been set:  Note the price in the left red circle for the 'JMAT.L' stock is very close to the alert price in the right hand circle. The other alerts also have alert prices very close to the actual price. Now, we wait a while for price changes. After a while, the following email appears in my inbox:

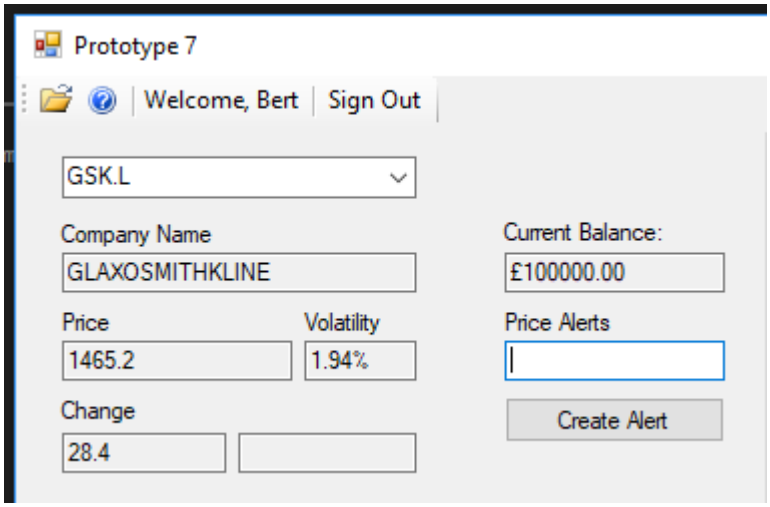
	 Clicking on one of the emails shows the following:  This indicates that the requirements set out in the test description have been met and the test has therefore been passed.	
Show that new alerts are rejected if the	Invalid alert data includes entering the current price of the stock as the alert price, trying to make an alert with blank entries, or using letters as the alert price.	

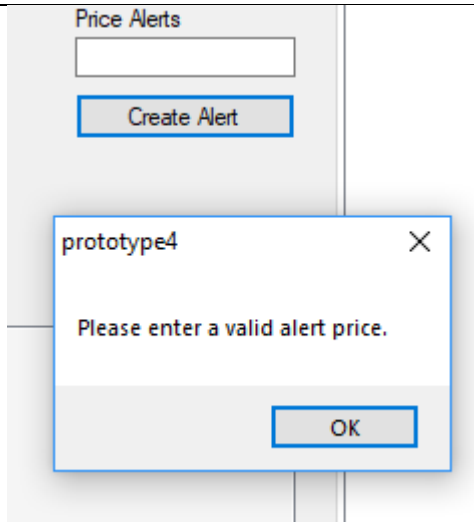
input data is invalid.



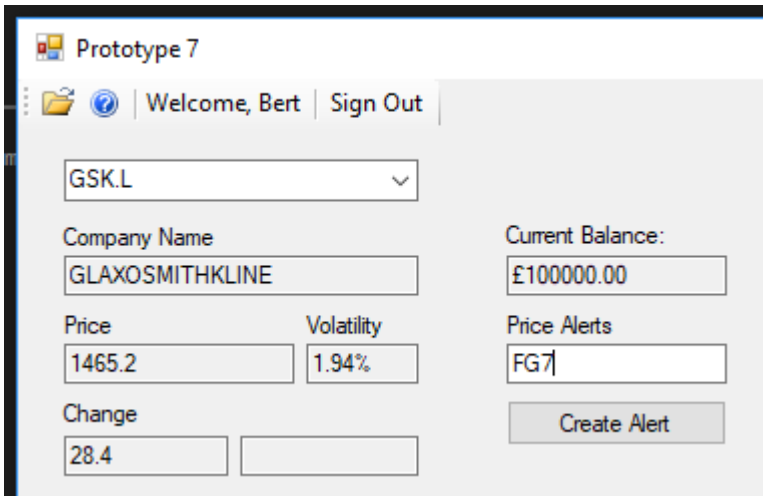


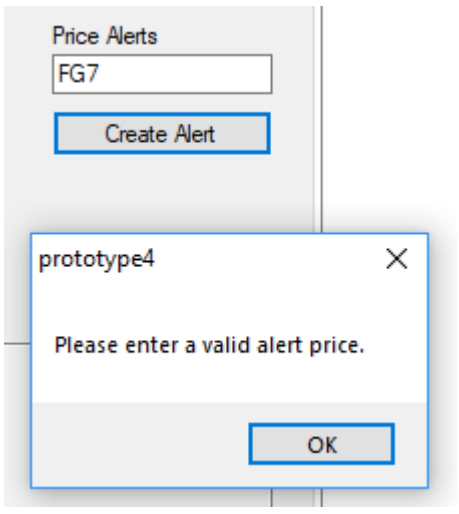
This shows that entering an alert price equal to the current price will not be accepted.



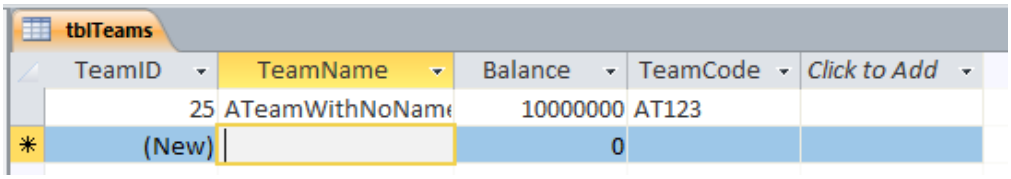
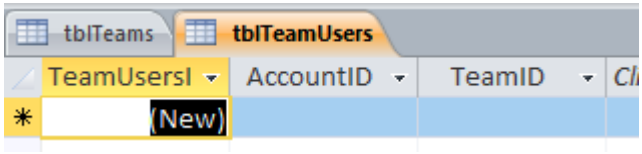
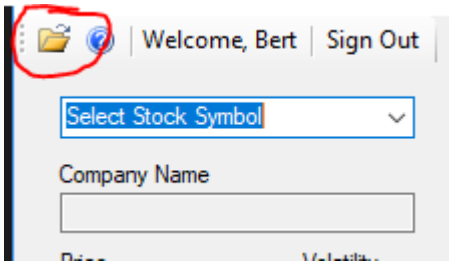
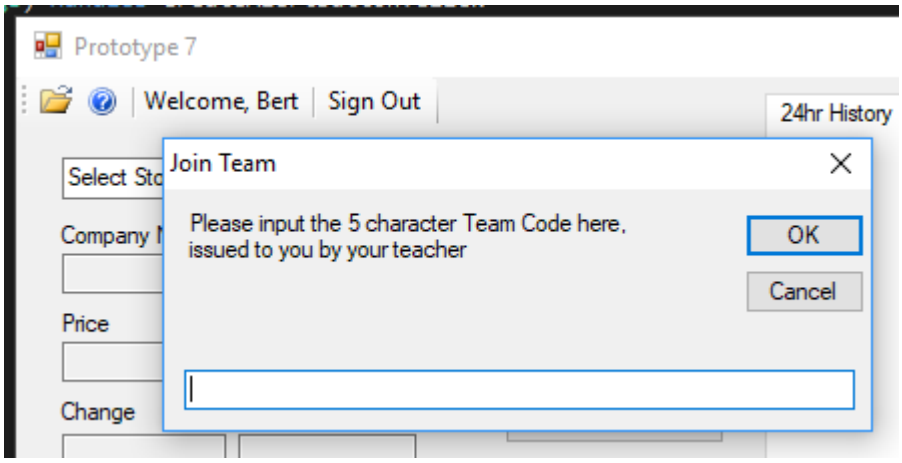


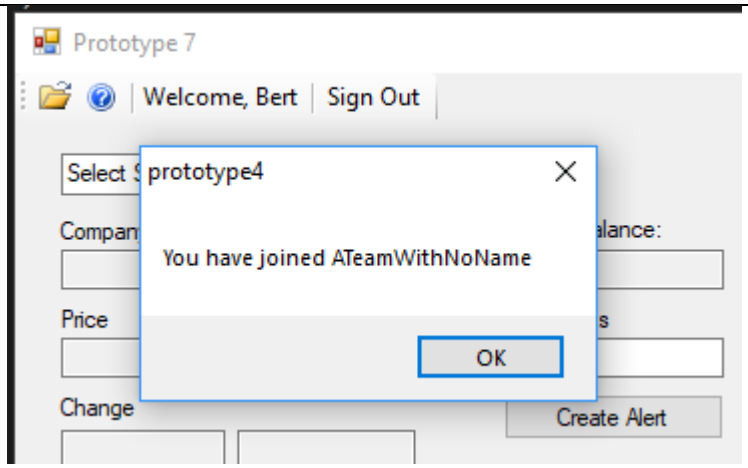
This shows that alerts with blank inputs will not be accepted.



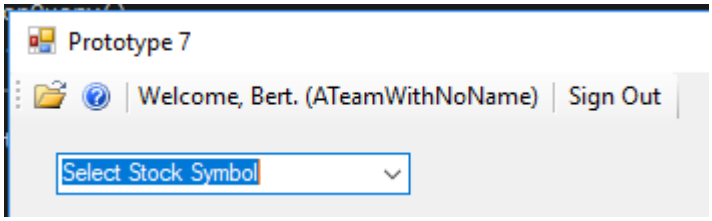


This shows that non-numeric inputs will not be accepted in the alert price box.

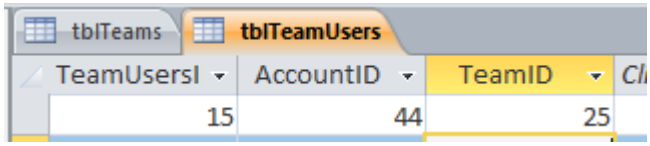
	This indicates that the requirements set out in the test description have been met and the test has therefore been passed.	
Show that the user can join a team from within the simulation.	A team is created in the database:  There is no members in this team, as shown by the blank link table 'tblTeamUsers':  Inside the simulation, this button is clicked:  This results in the following dialogue:  The code entered is 'AT123'. This results in the following dialogue:	



The account is then forced out of the simulation back to the sign-in form. Upon signing back in (with team mode selected), the following dialogue is shown at the top:



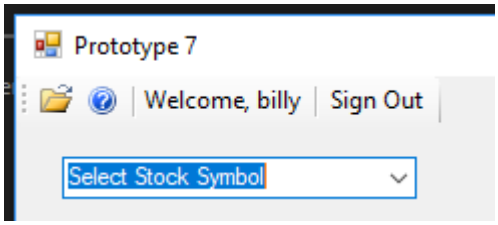
To prove further that the user is in the team, we can look at the link table 'tblTeamUsers':



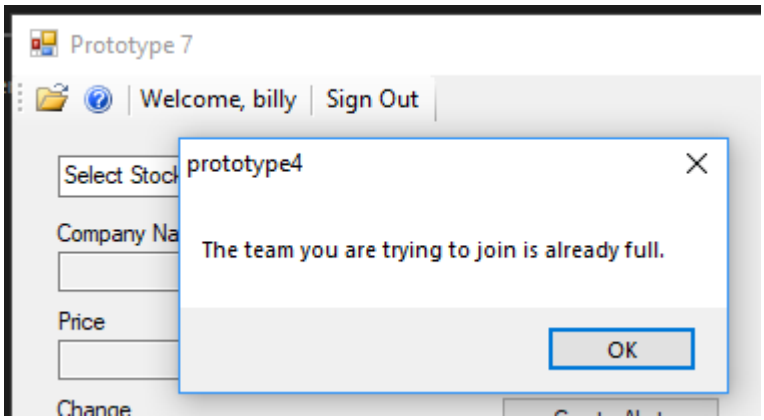
The account ID of the account 'Bert' is 44, and the team ID of 'AteamWithNoName' is 25. Therefore a link between the two exists, and the user is a member of the team.

This indicates that the requirements set out in the test description have been met and the test has therefore been passed.

Show that the user is rejected from the team if it is full or the team code is invalid.	A new account called Billy is created via the sign-up form.

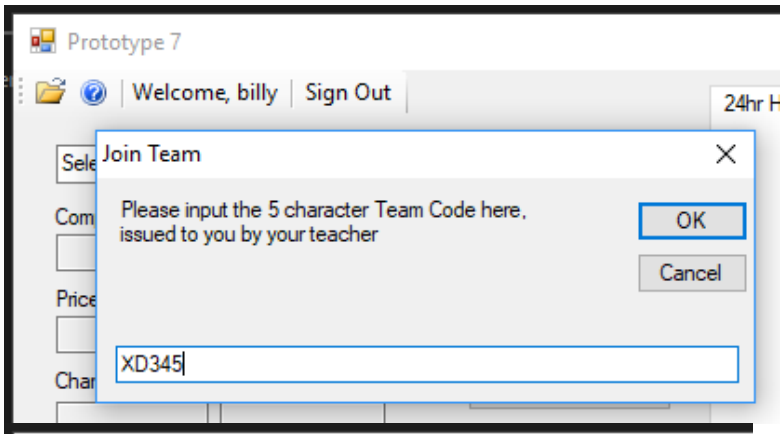


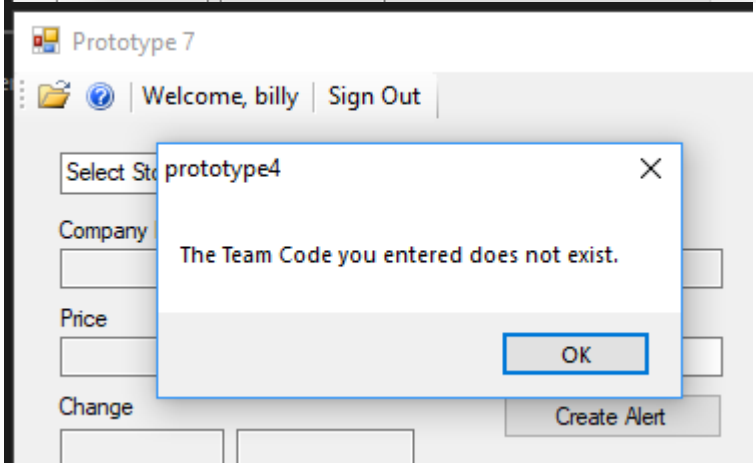
The link table ‘tblTeamUsers’ is filled with entries that link a total of 4 accounts to the team. The maximum number of accounts in a team is 4. Therefore, when Billy tries to join the team, there should be an error.



Billy is rejected from the team.

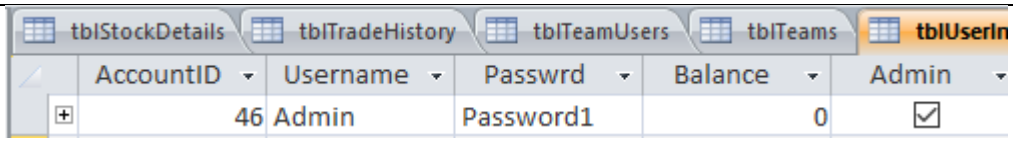
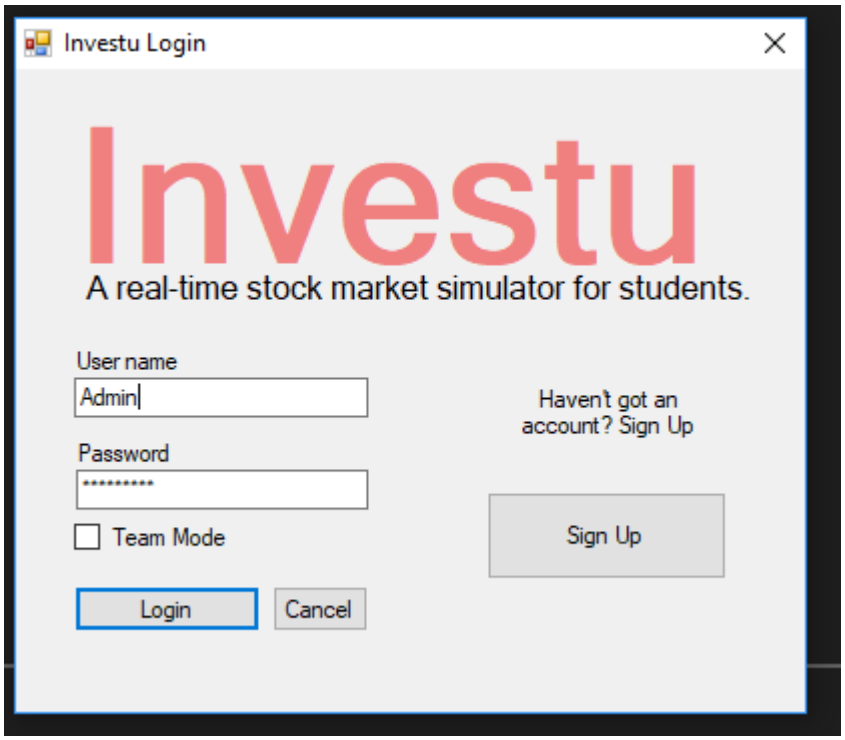
Billy tries to join another team with a team code that does not exists (Only 1 team exists in the database, the team code of which is ‘AT123’.)

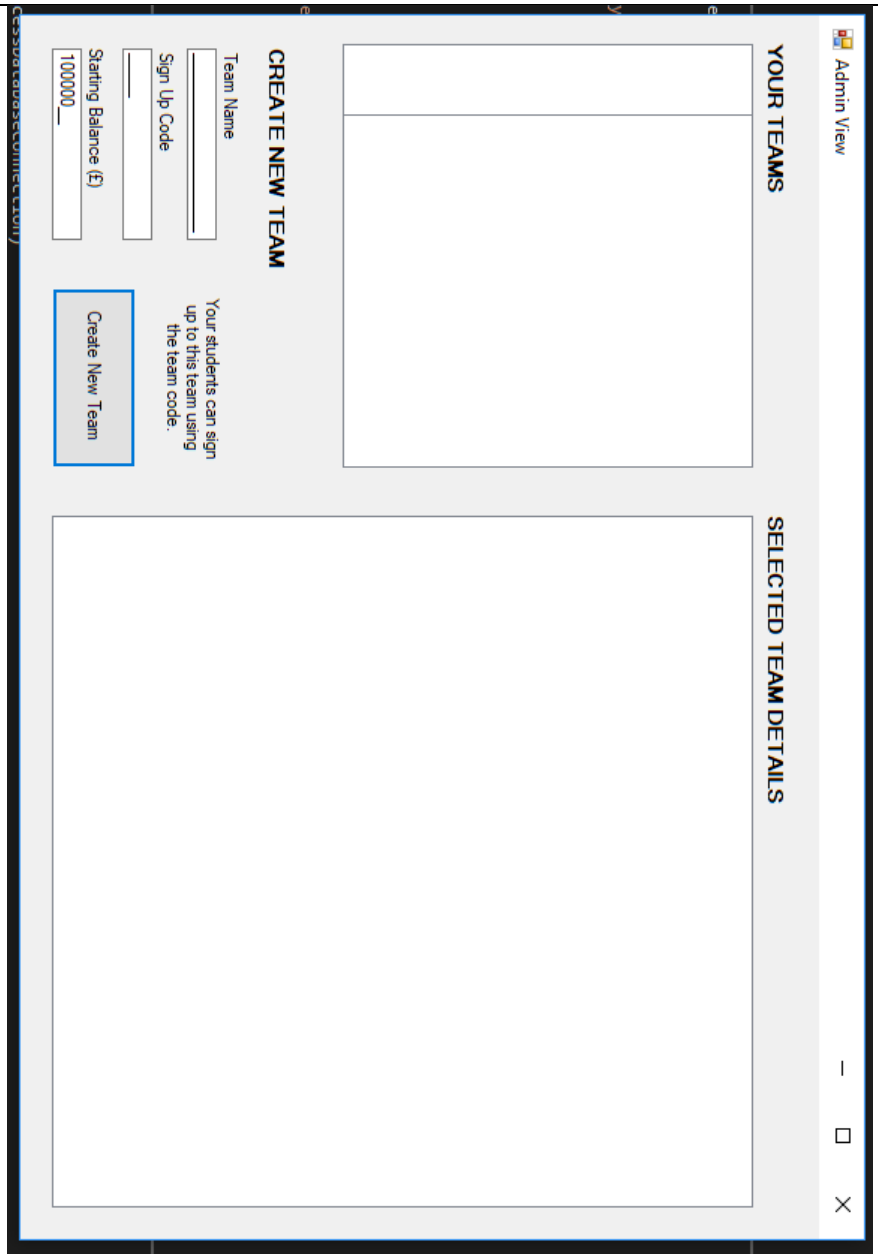




	Billy is not able to sign up to this team either, as the team code does not exist. This indicates that the requirements set out in the test description have been met and the test has therefore been passed.	

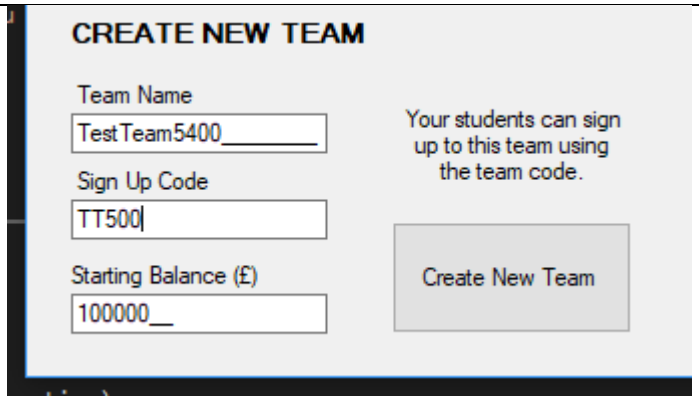
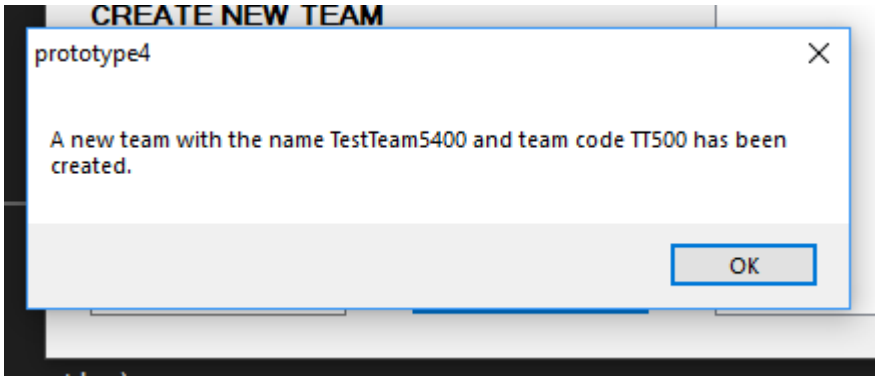
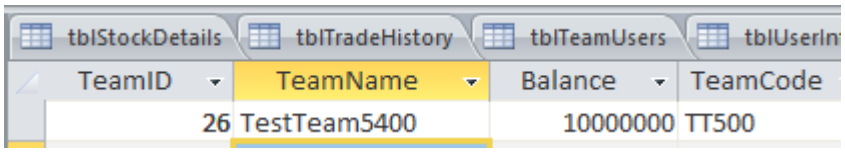
AdminView Testing 1

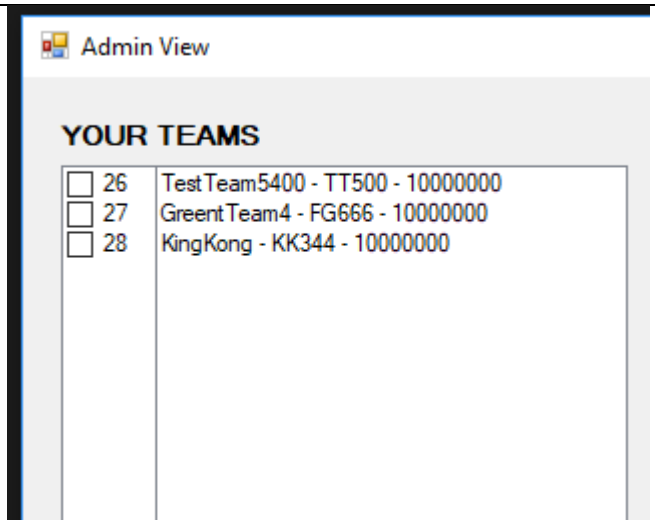
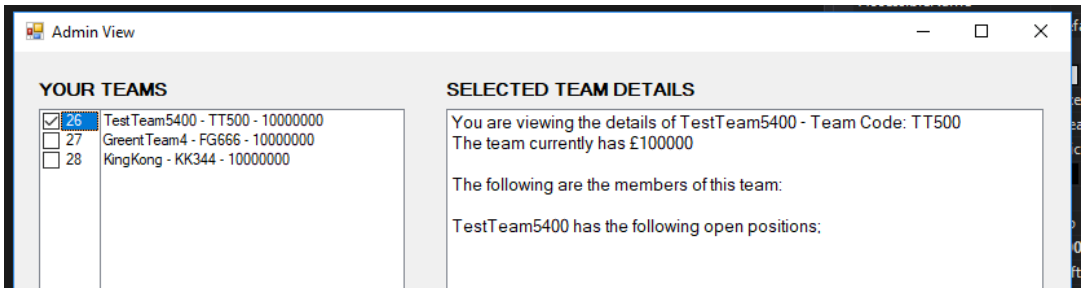
Show that admins load into AdminView when they sign in via the sign in form.	 A new admin is written into the database. Note the check in the 'Admin' attribute.  Upon clicking login with the admin details added, the following is displayed:



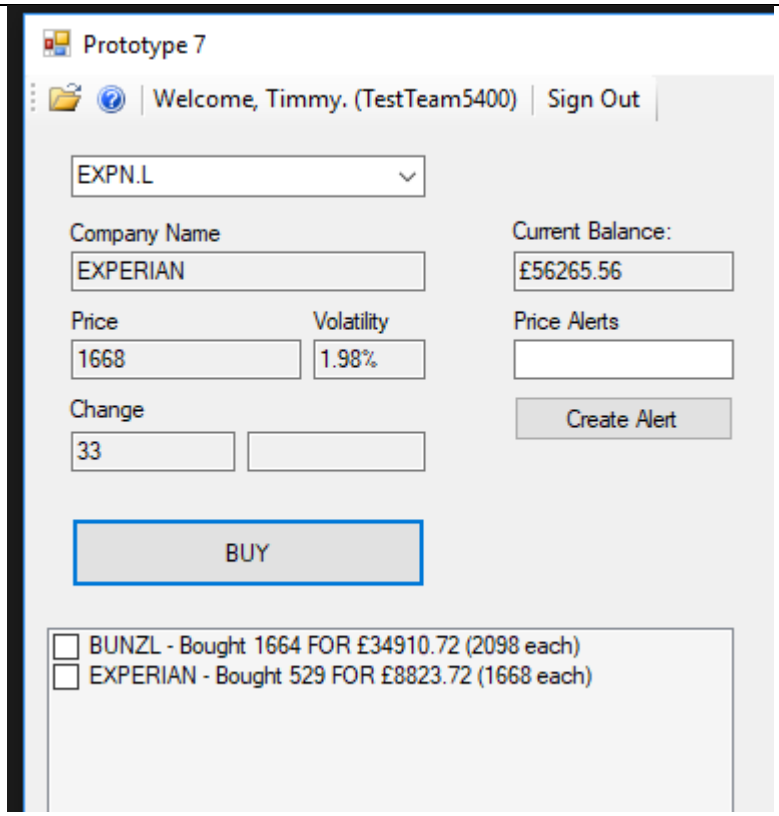
Note that there is no information displayed because there are currently no teams or accounts to view.

This indicates that the requirements set out in the test description have been met and the test has therefore been passed.

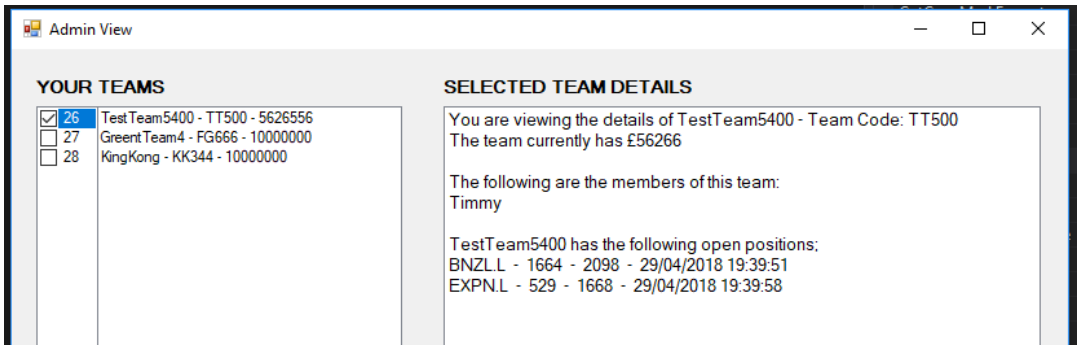
Show that admins can create a team, and the team is inserted into the database.	 Entering the above information into the boxes given, then pressing 'Create New Team', results in the following dialogue:  Going into the database, under the table 'tblTeams', the following is shown:  Note that the use of masked textboxes means it is not possible to enter invalid data into any of the boxes. See page 201 for more information. This indicates that the requirements set out in the test description have been met and the test has therefore been passed.	
Show that the list of created teams is displayed in the display.	Creating three teams results in the following list displaying in the display box:	

	 This indicates that the requirements set out in the test description have been met and the test has therefore been passed.
Show that these teams can be selected, and more information relating to the team will display.	Selecting one of the teams in the list box results in the following display in the display box on the right hand side of the form:  Note that there are no members or open positions, because the team has just been created. This indicates that the requirements set out in the test description have been met and the test has therefore been passed.
Show that adding members to the team and making trades shows within admin form when the	A new account is created called Timmy. Timmy joins the team ‘TestTeam5400’ via the sign up feature within the simulation, using the code’ TT500’. Timmy is then signed in, in team mode. Timmy makes 2 trades, as shown below:

team is selected.

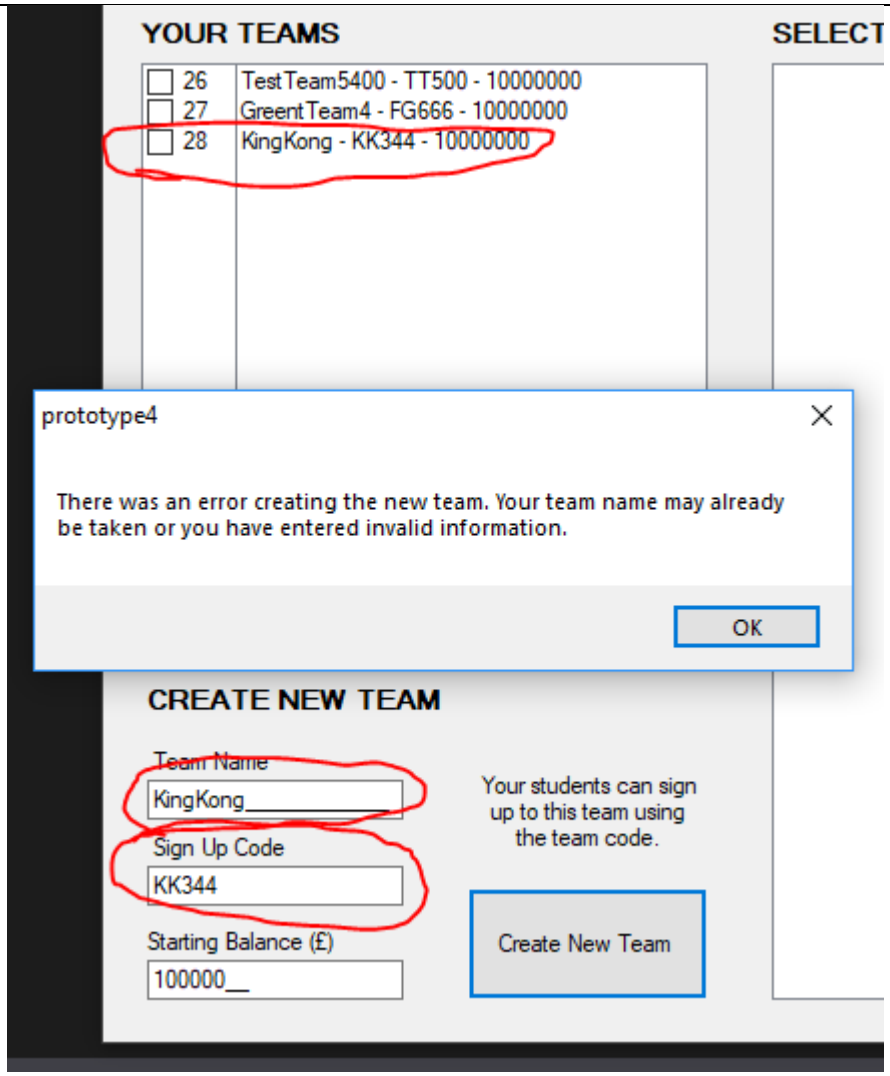


Loading up the simulation on an admin account now displays the following, when the team is selected:



Note how Timmy is now listed under ‘members of this team:’ and the trades made on that account also display below.

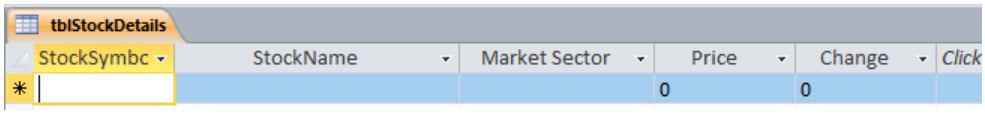
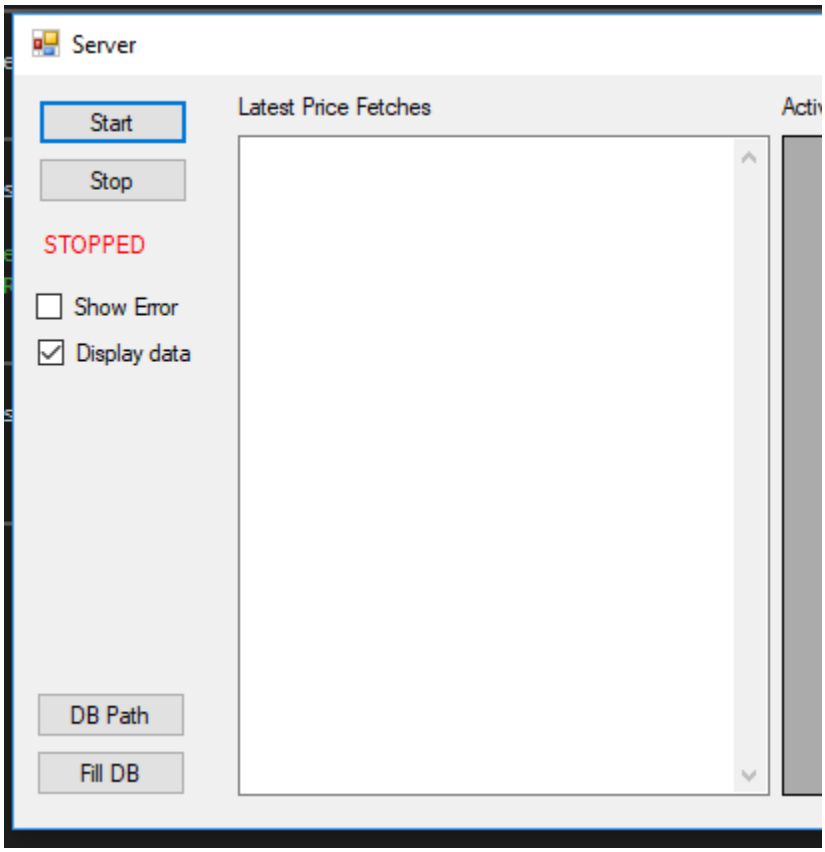
This indicates that the requirements set out in the test description have been met and the test has therefore been passed.

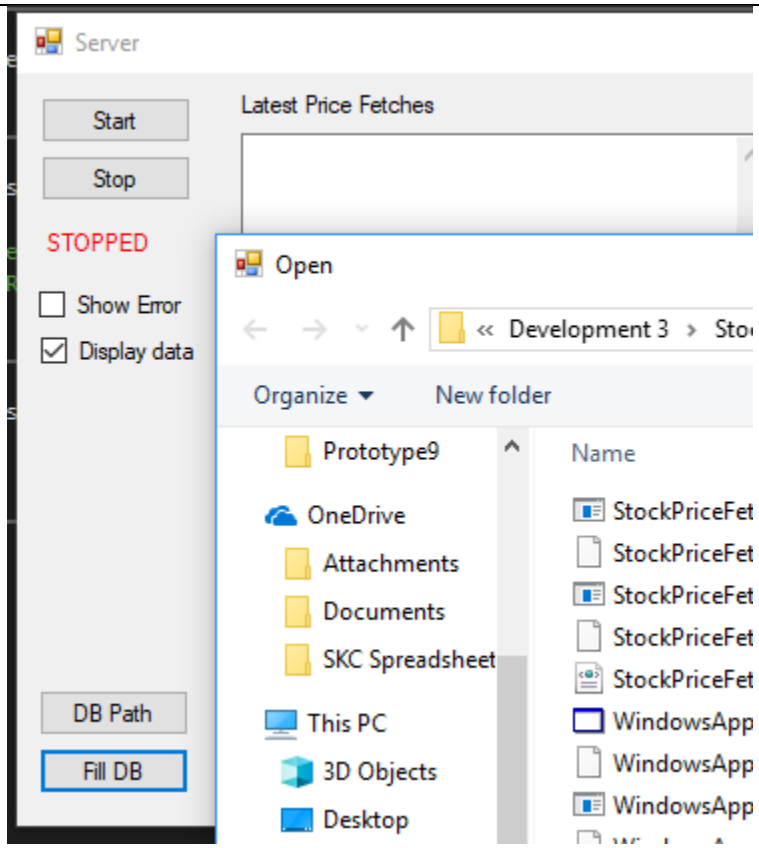
Show that teams with duplicate name or code cannot be created.	 Trying to make a new team with the same name or code results in the above error. Note how the details for the team in the 'Create New Team' area are the same as an already-existing team. Therefore, the creation is rejected. This indicates that the requirements set out in the test description have been met and the test has therefore been passed.

Testing 1 – Investu Server Program – Development 3

Every sub-routine has been tested individually to ensure it works independently, however these tests have not been shown. The tests displayed are tests that show multiple sub-routines and functions working together to produce the desired outcome.

Many of the features of the simulation that haven't been changed since Development 2 are not tested here. To see the tests that are not displayed here, refer to Testing in Development 2, on page 149.

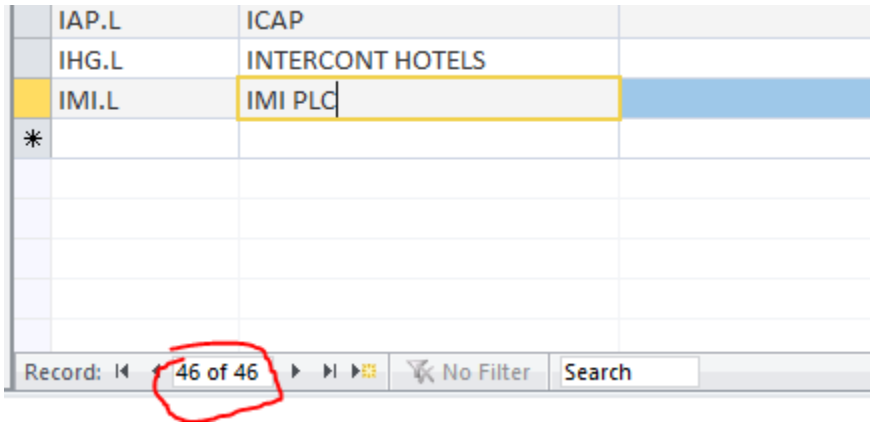
Show that the database table 'tblStockDetails' is filled with up-to-date information upon clicking the 'FillDB' button.	The table 'tblStockDetails' is currently empty, as shown below:  After loading the server program, the following display is presented:  Clicking the 'FillDB' button results in the following:



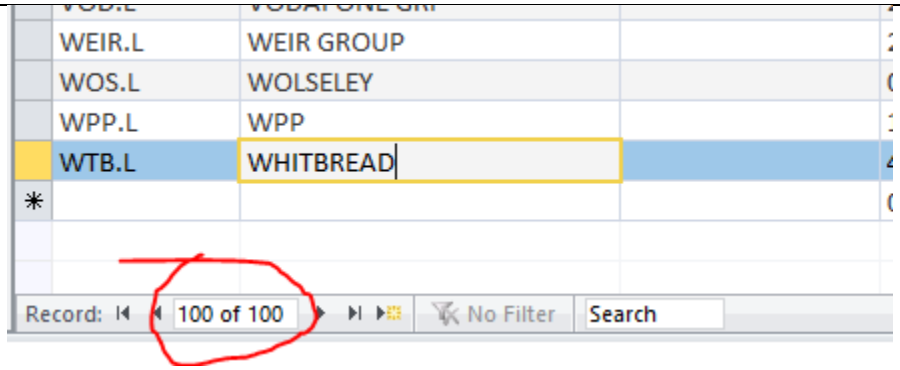
A file finder appears. This appears to allow the user to select the stock symbol list.

Selecting the CSV file with the symbols in closes the file finder.

The database now appears as follows:



Note there are 46 entries, whereas the expected number is 100 (there are 100 stock symbols in the FTSE100). This is because the process of fetching the information for all 100 stocks takes a long time. Upon refreshing the database, the following is show:



Note that there are now 100 entries. These entries, now that they have been created, will be kept up to date by Invesu Server Program, as it will update the entries when it receives new data, every time the timer ticks.

The view of the first 25 entries, from the top of the table, looks as follows:

tblTradeHistory | tblTeamUsers | tblUserInfo | tblTeams | tblStockDetails

StockSymbc	StockName	Market Sector	Price	Change
AAL.L	ANGLO AMERICAN		1699.2	32.2
ABF.L	ASSOCIAT BRIT FOODS		2700	15
ADM.L	ADMIRAL GROUP		1995.5	20
ADN.L	ABERDEEN ASSET MGMT		316.33	0
AGK.L	AGGREKO		734.2	0.6
AMEC.L	AMEC		0	0
ANTO.L	ANTOFAGASTA		954	2.8
ARM.L	ARM HOLDINGS		0	0
ASHM.L	ASHMORE GRP		410	2.6
AV.L	AVIVA		525	0.2
AZN.L	ASTRAZENECA		5083	91
BA.L	BAE SYSTEMS		613.4	2.2
BARC.L	BARCLAYS		208.69	-1.31
BATS.L	BRIT AMER TOBACCO		4030.5	65
BG.L	BG GROUP		0	0
BLND.L	BRIT LAND CO REIT		672	5.4
BLT.L	BHP BILLITON		1534.6	-1.6
BNZL.L	BUNZL		2098	20
BP.L	BP		537.4	2.1
BRBY.L	BURBERRY GROUP		1822	54
BSY.L	B SKY B GROUP		0	0
BT-A.L	BT GROUP		248	2.55

'Market Sector' is an attribute that will have to be manually entered into the database. That's because there is no way to retrieve the market sector of a company using the current method of querying Google Sheets. In a future development it's possible that a method could be found to automatically set this attribute and keep it updated.

This indicates that the requirements set out in the test description have been met and the test has therefore been passed.

Show that the information in 'tblStockDetails' is kept up to date every time the timer ticks.	The following shows a screenshot of the table 'tblStockDetails', after the button 'FillDB' has been pressed inside Investu Server Program: tblUserInfo | tblStockDetails | StockSymbc	StockName	Market Sector	Price	Change	Click to A
---	---	---	---	---	---
AAL.L	ANGLO AMERICAN		1708.6	9.4	
ABF.L	ASSOCIAT BRIT FOODS		2703	3	
ADM.L	ADMIRAL GROUP		2011	15.5	
ADN.L	ABERDEEN ASSET MGMT		316.33	0	
AGK.L	AGGREKO		735.6	1.4	
AMEC.L	AMEC		0	0	
ANTO.L	ANTOFAGASTA		959	5	
ARM.L	ARM HOLDINGS		0	0	
ASHM.L	ASHMORE GRP		412	2	
AV.L	AVIVA		529.06	4.06	
AZN.L	ASTRAZENECA		5097	14	
BA.L	BAE SYSTEMS		615.4	2	
BARC.L	BARCLAYS		208.1	-0.85	
BATS.L	BRIT AMER TOBACCO		4005	-25.5	
BG.L	BG GROUP		0	0	
BLND.L	BRIT LAND CO REIT		676.4	4.4	
BLT.L	BHP BILLITON		1535.4	0.8	
BNZL.L	BUNZL		2109	11	
BP.L	BP		534.6	-2.8	
BRBY.L	BURBERRY GROUP		1814.5	-7.5	
BSY.L	B SKY B GROUP		0	0	
BT-A.L	BT GROUP		250.35	2.35	
CCL.L	CARNIVAL		4762	30	
CNA.L	CENTRICA		153.81	0.96	
CPG.L	COMPASS GROUP		1557.5	3	
CPI.L	CAPITA		186.45	-0.45		 Now, the server program is left to run for a few minutes. The result is shown in the screenshot below:

tblStockDetails

StockSymbc	StockName	Market Sector	Price	Change
AAL.L	ANGLO AMERICAN		1712.6	13.4
ABF.L	ASSOCIAT BRIT FOODS		2700	0
ADM.L	ADMIRAL GROUP		2003	7.5
ADN.L	ABERDEEN ASSET MGMT		316.33	0
AGK.L	AGGREKO		735	0.8
AMEC.L	AMEC		0	0
ANTO.L	ANTOFAGASTA		955.6	1.6
ARM.L	ARM HOLDINGS		0	0
ASHM.L	ASHMORE GRP		411.6	1.6
AV.L	AVIVA		528.2	3.2
AZN.L	ASTRAZENECA		5089	6
BA.L	BAE SYSTEMS		614.2	0.8
BARC.L	BARCLAYS		207.9	-1.05
BATS.L	BRIT AMER TOBACCO		4009.36	-19.5
BG.L	BG GROUP		0	0
BLND.L	BRIT LAND CO REIT		674.2	2.2
BLT.L	BHP BILLITON		1540	5.4
BNZL.L	BUNZL		2107	9
BP.L	BP		535.1	-2.3
BRBY.L	BURBERRY GROUP		1824	2
BSY.L	B SKY B GROUP		0	0
BT-A.L	BT GROUP		251.1	3.1
CCL.L	CARNIVAL		4763	31
CNA.L	CENTRICA		153.95	1.1
CPG.L	COMPASS GROUP		1560	5.5
CPI.L	CAPITA		186.28	-0.62

Note how the prices in the ‘Price’ attribute are virtually all different in comparison to the screenshot taken only a few minutes earlier.

This proves that the table is updating autimatically, and so the data is being kept up to date.

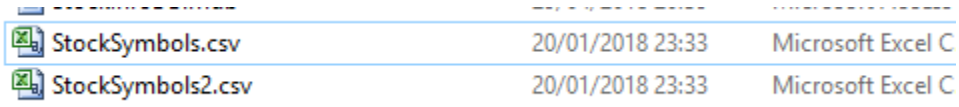
Show that the data inserted into ‘tblStockDetails’ is displayed in the main simulation in the stock details display area .

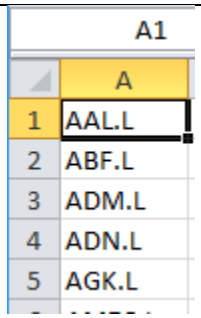
tblTradeHistory | tblTeamUsers | tblUserInfo | tblTeams | **tblStockDetails**

StockSymbc	StockName	Market Sector	Price	Change
AAL.L	ANGLO AMERICAN		1699.2	32.2
ABF.L	ASSOCIAT BRIT FOODS		2700	15
ADM.L	ADMIRAL GROUP		1995.5	20
ADN.L	ABERDEEN ASSET MGMT		316.33	0
AGK.L	AGGREKO		734.2	0.6
AMEC.L	AMEC		0	0
ANTO.L	ANTOFAGASTA		954	2.8
ARM.L	ARM HOLDINGS		0	0
ASHM.L	ASHMORE GRP		410	2.6
AV.L	AVIVA		525	0.2
AZN.L	ASTRAZENECA		5083	91
BA.L	BAE SYSTEMS		613.4	2.2
BARC.L	BARCLAYS		208.69	-1.31
BATS.L	BRIT AMER TOBACCO		4030.5	65
BG.L	BG GROUP		0	0
BLND.L	BRIT LAND CO REIT		672	5.4
BLT.L	BHP BILLITON		1534.6	-1.6
BNZL.L	BUNZL		2098	20
BP.L	BP		537.4	2.1
BRBY.L	BURBERRY GROUP		1822	54
BSY.L	B SKY B GROUP		0	0
BT-A.L	BT GROUP		248	2.55

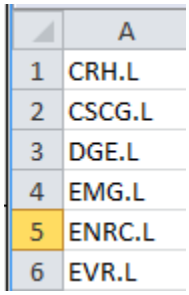
The above screenshot shows the data in the table. This data is originally set by clicking ‘FillDB’, then kept up to date via the timer tick, which updates the relevant entry every time new information is retrieved.

Loading the program and selecting the ‘Stock Details’ tab in the tab control shows the following:

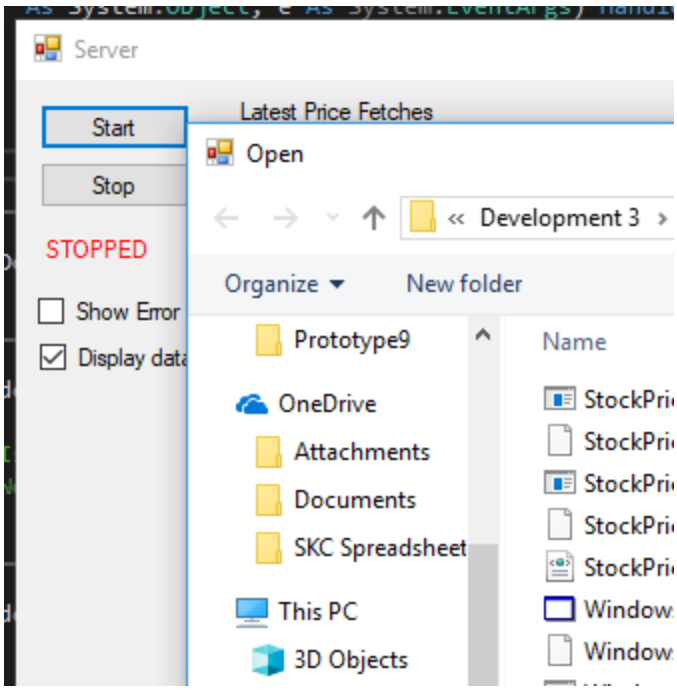
	 This indicates that the requirements set out in the test description have been met and the test has therefore been passed.
Show that the database file path and StockSymbols.CSV file path can be changed.	For this test two CSV files with symbols are created:  The first begins with the symbol ‘AAL.L’:



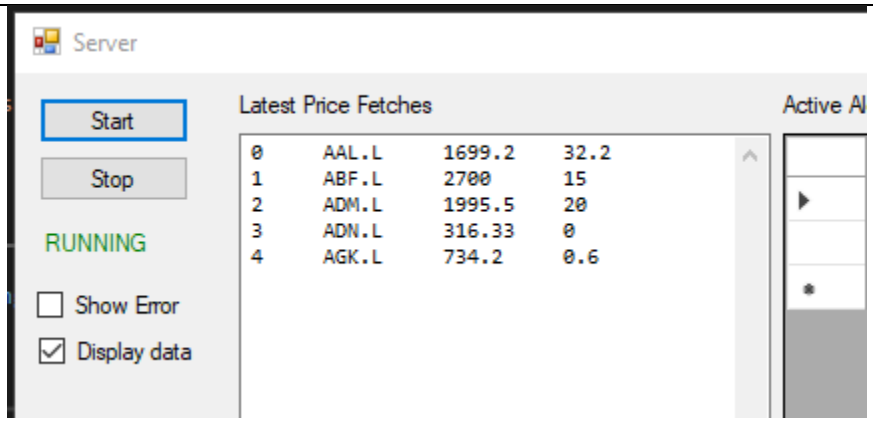
The second beginds with the symbol ‘CRH.L’



Clicking ‘Start’ in the program shows a file finder dialogue:

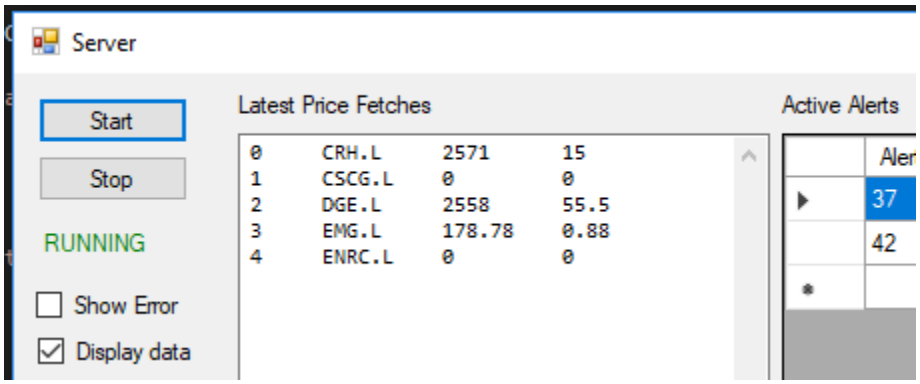


We can navigate through the file directory to the two different CSV files. For the first run of the program, the first CSV file is selected. The following is the result:



Note how the program begins from 'AAL.L' – the first symbol in the first CSV file.

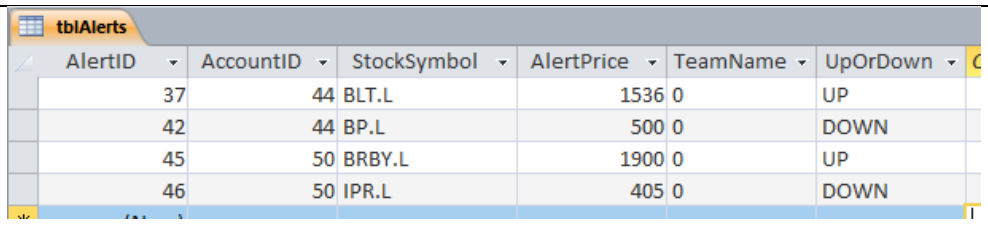
The second run of the program uses the second CSV file:



Note how the first symbol is now 'CRH.L', the same as in the second CSV file.

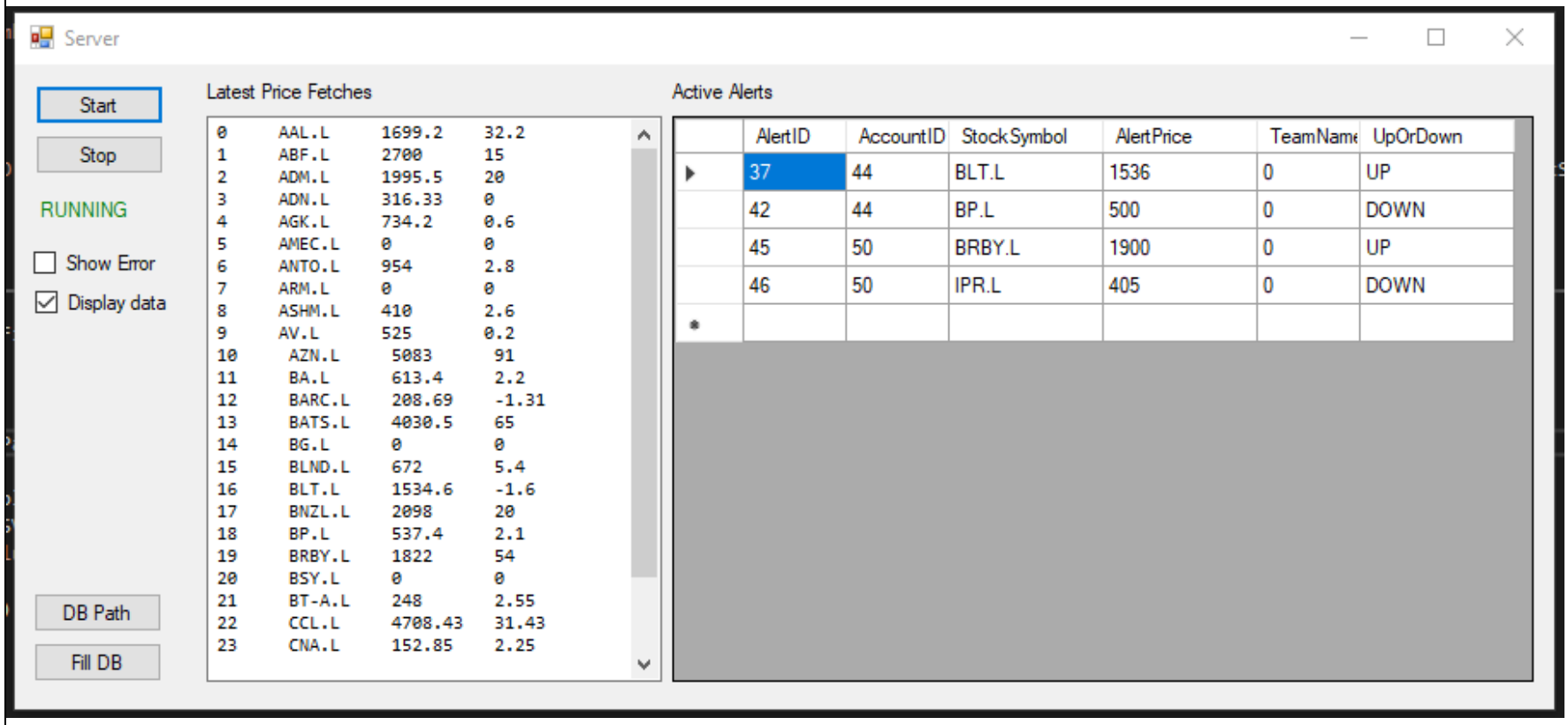
This indicates that the requirements set out in the test description have been met and the test has therefore been passed.

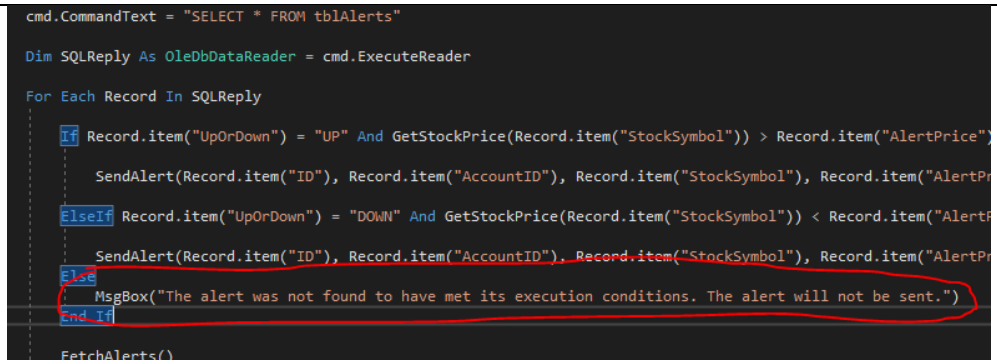
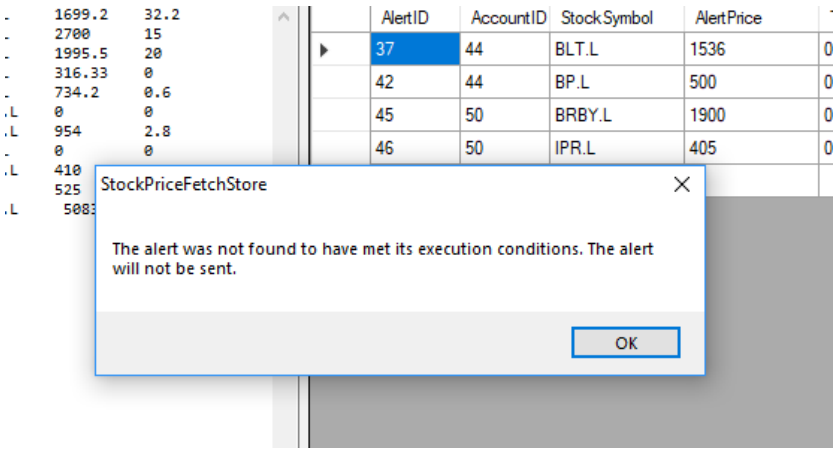
Show that alerts created by users within the main simulation are loaded into the list of alerts in the server program.

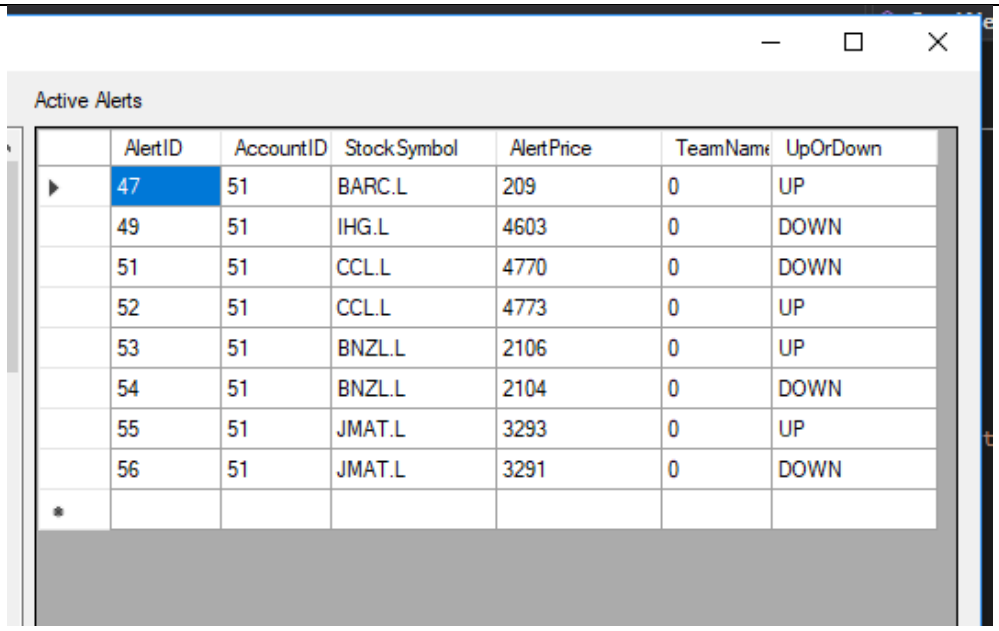


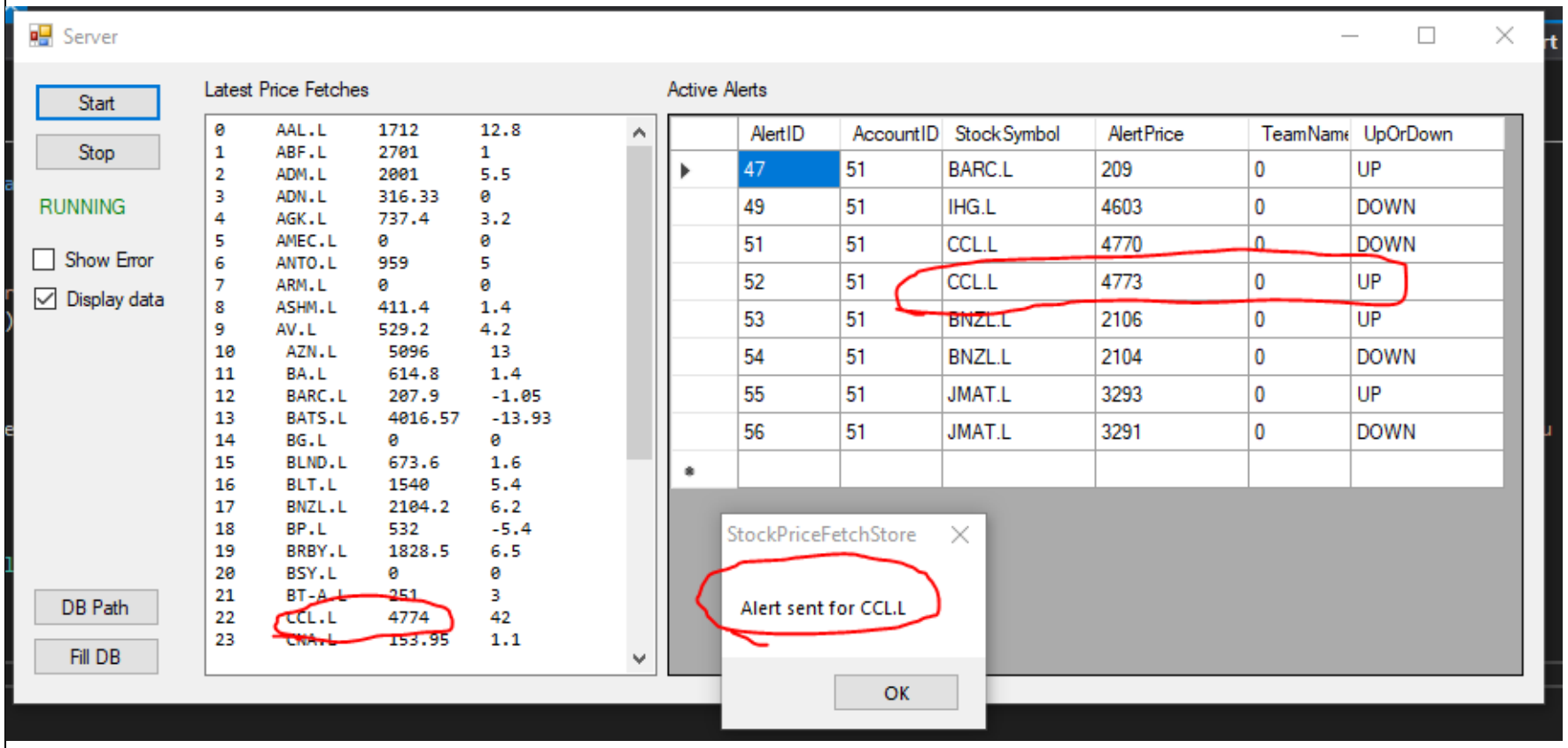
AlertID	AccountID	StockSymbol	AlertPrice	TeamName	UpOrDown
37	44	BLT.L	1536	0	UP
42	44	BP.L	500	0	DOWN
45	50	BRBY.L	1900	0	UP
46	50	IPR.L	405	0	DOWN

The above screenshot shows the table 'tblAlerts'. Loading the server program shows the display below:


Server
Start
Stop
RUNNING
Show Error
Display data
DB Path
Fill DB
Latest Price Fetches
0 AAL.L 1699.2 32.2
1 ABF.L 2700 15
2 ADM.L 1995.5 20
3 ADN.L 316.33 0
4 AGK.L 734.2 0.6
5 AMEC.L 0 0
6 ANTO.L 954 2.8
7 ARM.L 0 0
8 ASHM.L 410 2.6
9 AV.L 525 0.2
10 AZN.L 5083 91
11 BA.L 613.4 2.2
12 BARC.L 208.69 -1.31
13 BATS.L 4030.5 65
14 BG.L 0 0
15 BLND.L 672 5.4
16 BLT.L 1534.6 -1.6
17 BNZL.L 2098 20
18 BP.L 537.4 2.1
19 BRBY.L 1822 54
20 BSY.L 0 0
21 BT-A.L 248 2.55
22 CCL.L 4708.43 31.43
23 CNA.L 152.85 2.25
Active Alerts
AlertID AccountID StockSymbol AlertPrice TeamName UpOrDown
37 44 BLT.L 1536 0 UP
42 44 BP.L 500 0 DOWN
45 50 BRBY.L 1900 0 UP
46 50 IPR.L 405 0 DOWN

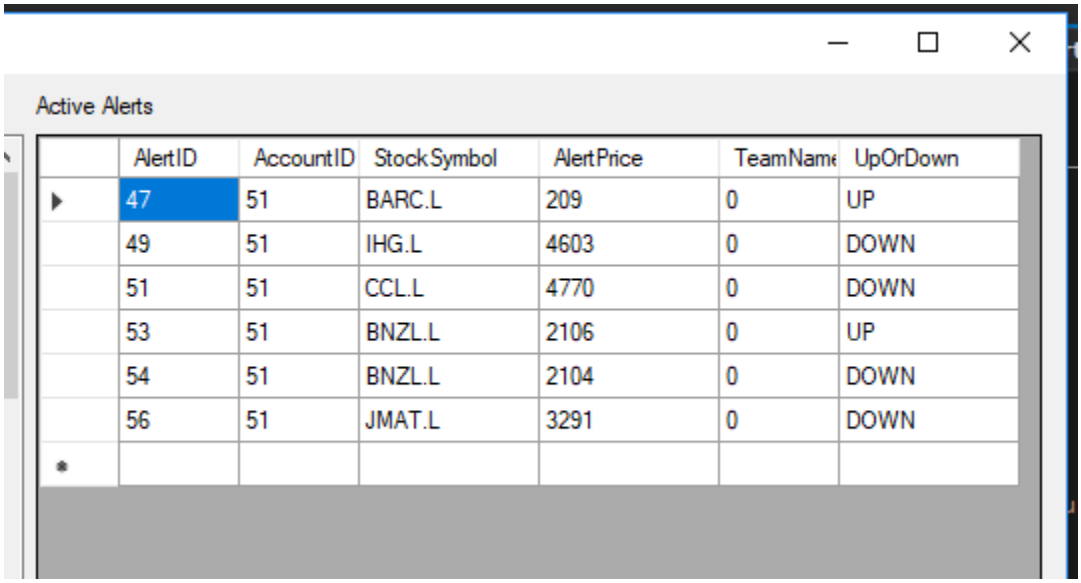
	Note how all 4 of the alerts are displayed within the program. These alerts will then be checked constantly until the conditions are met, at which point they will be executed, and the alert deleted. This indicates that the requirements set out in the test description have been met and the test has therefore been passed.
Show that the list of alerts is checked regularly.	 This test requires an addition to the code. This addition is a message box, that will show every time an alert is checked, but not executed due to not meeting the execution conditions. Immediately after loading the program, the following dialogue appears:  This alert appears 4 times, once for each alert. Clicking ‘Ok’ on all 4 results in no boxes. They reappear after about a minute – this is because alerts are checked every minute in the program. This indicates that the requirements set out in the test description have been met and the test has therefore been passed.
Show that alerts are executed and users notified when	A number of alerts were created, as seen in the right hand display in the below screenshot.

the the alert price is exceeded.	 Now that these alerts have been set, we wait for stock price changes. When the prices exceed the alert prices, an alert will be sent.


Server
Start
Stop
RUNNING
Show Error
Display data
DB Path
Fill DB
Latest Price Fetches
0 AAL.L 1712 12.8
1 ABF.L 2701 1
2 ADM.L 2001 5.5
3 ADN.L 316.33 0
4 AGK.L 737.4 3.2
5 AMEC.L 0 0
6 ANTO.L 959 5
7 ARM.L 0 0
8 ASHM.L 411.4 1.4
9 AV.L 529.2 4.2
10 AZN.L 5096 13
11 BA.L 614.8 1.4
12 BARC.L 207.9 -1.05
13 BATS.L 4016.57 -13.93
14 BG.L 0 0
15 BLND.L 673.6 1.6
16 BLT.L 1540 5.4
17 BNZL.L 2104.2 6.2
18 BP.L 532 -5.4
19 BRBY.L 1828.5 6.5
20 BSY.L 0 0
21 BT-A.L 251 3
22 CCL.L 4774 42
23 CNA.L 153.95 1.1
Active Alerts
AlertID AccountID StockSymbol AlertPrice TeamName UpOrDown
47 51 BARC.L 209 0 UP
49 51 IHG.L 4603 0 DOWN
51 51 CCL.L 4770 0 DOWN
52 51 CCL.L 4773 0 UP
53 51 BNZL.L 2106 0 UP
54 51 BNZL.L 2104 0 DOWN
55 51 JMAT.L 3293 0 UP
56 51 JMAT.L 3291 0 DOWN
StockPriceFetchStore
Alert sent for CCL.L
OK

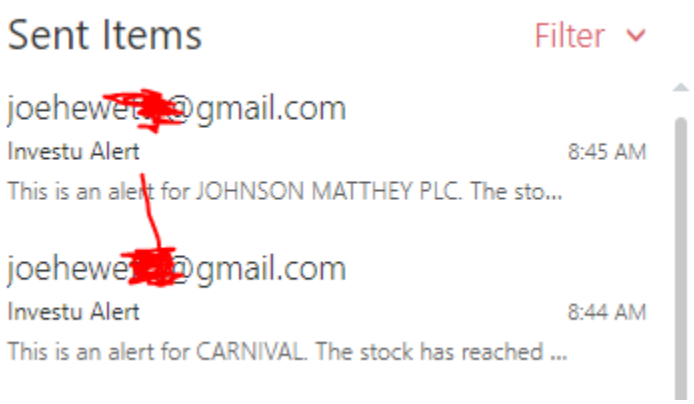
After leaving the program a while, one of the alerts sends. Notice that the alert price in the top right red circle is 4773. Then, notice the left hand red circle shows that the current price has just changed to 4774. Because the alert is an 'Up' alert (The alert was set when the actual price was below the alert price), the alert has triggered, hence the message box.

At the time of taking this screenshot, two alerts have triggered, and as such they are no longer in the list of alerts within the program.

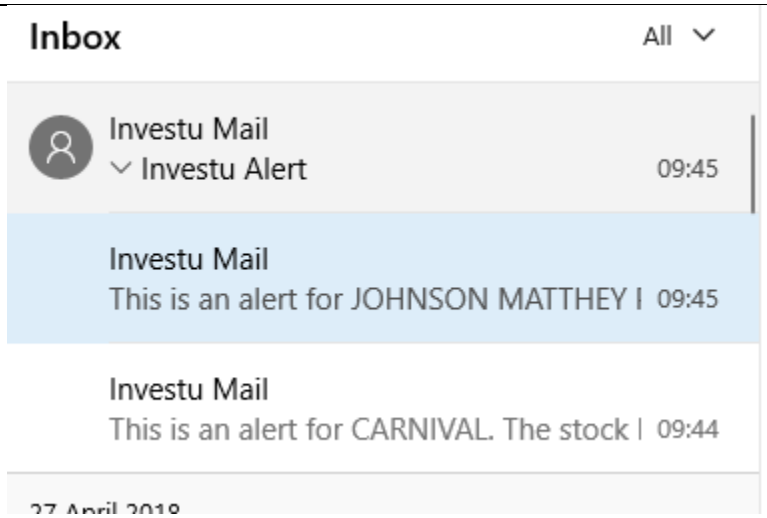


	AlertID	AccountID	StockSymbol	AlertPrice	TeamName	UpOrDown
▶	47	51	BARC.L	209	0	UP
	49	51	IHG.L	4603	0	DOWN
	51	51	CCL.L	4770	0	DOWN
	53	51	BNZL.L	2106	0	UP
	54	51	BNZL.L	2104	0	DOWN
	56	51	JMAT.L	3291	0	DOWN
*						

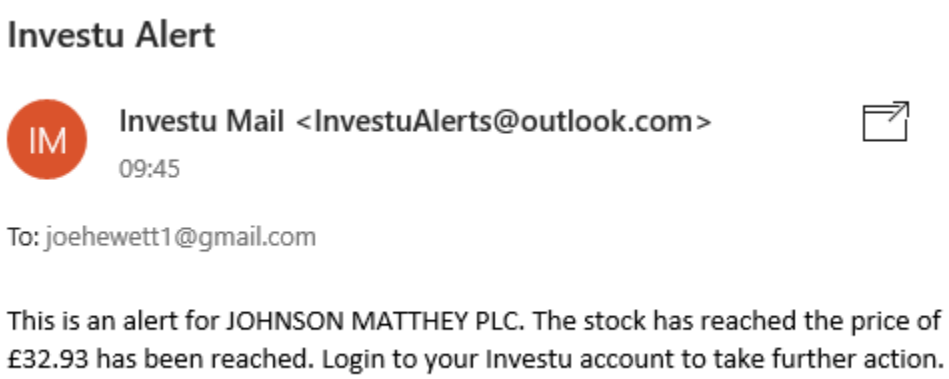
Checking the Investu outlook account shows the following emails have been sent:



Visiting the email that is shown in the screenshot above, we see the following:



Clicking on one of the emails displays the following text:



Conclusion

Feedback #5 – Users - Investu – Development 3

For the final development of the program I wanted to contact some users and the client in order to gauge their satisfaction with the final product. I gave 4 users from the clients class the program and went back a week later to discuss their experience. The following is a recorded in-person conversation that was had, with the users.

Me
“Can you give me an insight into your experiences over the last week of using Investu?”

Ben
“It’s been good – I’ve been using it a little bit each day just checking how things are doing and I’ve made a lot of progress. I got Mr B. to make us a team so I’ve been using it with my SIC group and it’s been good fun”

Me
“How is the trading going – have you had any success?”

Alex
“We’ve had a lot of success. We’ve found the news features in this last update really useful – if you don’t have any idea about which stock to invest in it’s great to just be able to check the professional’s opinion easily within the program. We connected our school emails and so we’ve all been getting alerts throughout the day, which has made it really easy to trade as you and can just wait until the perfect time to buy or sell. It’s simple really.”

Me
“How has the team dynamic worked out?”

Ollie
“Great – it was really simple to get together, we just asked Mr B. for the code and we were away. We found the process really smooth – you can easily leave and join other teams. My team is doing try-outs for our fourth position, as someone can’t make it for the next challenge, so we’ve been using that feature to take people in then remove them and test other people. The notes system has been really good for that too. It’s great to be able to see who’s making each trade and why – sometimes you forget the goal of a trade or why you made it and so that’s good to be able to see.

Me
“Which positives are there of having Investu, in your opinion?”

Ben
“I think there’s a lot of positives – it’s so easy to keep in the trading mindset outside of the SIC season. You have to be on the ball when you’re trading – constantly checking your positions and the market and reading all the time to make sure you’re making good choices. It’s easy to fall out of the rhythm when you don’t do it for a long time, so having Investu is going to be great for keeping in the mindset while we get ready before and after. The fact Mr B. can see the progress of our account is great too – we’ve been

having a running conversation over the last few days about which trades we've made and he's had some good feedback."

Me
"Any negatives or things that need improving?"

Alex
"I wouldn't exactly call them negatives. the simulation does exactly what is needed. I think there are things that could make the experience even better though. I think more analysis would be good. Maybe the option to apply a list of models to the price data of a stock and see what they say? - there are some models that take quantitative data and produce an output saying whether to buy or sell. I know you can get those in Python. I'm not sure about Visual Basic. Why did you choose Visual Basic? I feel like Python would be much more suited to this type of thing. Anyway, things like that would be really useful. Also, I think the graph system could be expanded. The graphs show recent data which is great, but what if we could see the data from as far back as possible? Being able to scrub through it and see trends would be great. Then we could try and associate patterns to current events and predict things. I think it would just give a broader picture of what's going on."

Me
"Thanks for the feedback."

Feedback #6 – Client - Investu – Development 3

Me
“How has your experience been over the last week of using the simulation?”

Client
“It’s been a lot of fun – the group and I have been using it on and off between lessons and such and it’s been really nice – it allows me to interact with them in a way that I haven’t been able to previously. I think it’s a resounding success.

Me
“What areas do you think Investu is strong in?”

Client
“Theres a lot of great things – it’s simple and easy to use, the process of trading is very simple. The users have been really active as well, which is nice to see. I think thats because they can use it in a team, which creates a much more social and enjoyable experience. The fact that they can receive notifications on their phones also keeps the idea of trading in their mind. I’ve been observing through my teachers account and it’s good to be able to see their though process and the progress they’re making.”

Me
“Do you feel as though all of these aims, that we discussed in our initial conversations, have been met?”

Client
“I think so. Investu is definitely what we need it to be and I’m happy with the result. It’s easy to use, and does everything I can remember asking for. I think the idea has been executed very well.”

Me
“What would you consider some of the weaker aspects of the simulation? What would you like to see added in future versions?”

Client
“The simulation is very strong – but depending on how far you’d want to go into it theres lots that could be done to further it. Making an app would be great, for example – would mean everyone could use it an any point without the need for a computer. Things like analysis as well would be really good additions – the teachers account would benefit a lot from statistics and analysis of the progress of the students. If I could see a score board of all the teams in order of the highest balance, or most profit in a week and so on, I think it could really add something.

Me
“Thanks for your help.”

Analysis of Feedback from Users

In general, the users seem satisfied with the state of the simulation. The users report using the features provided to enhance their trading experience, and to effectively profit within the simulation. The users also seems to find the simulation beneficial for their SIC preparation. The team system appears to be working as intended.

The users report that more analysis would be beneficial to the experience. They also want more functionality provided with the graph section.

Analysis of Feedback from Client

The client seems overall satisfied with the produced work. They particularly enjoy the interactive nature of the simulation and the simplicity/ease of use. The client believes that the simulation specification stated in the analysis has been effectively executed.

The client reports that additions to the admin mode feature allowing for team progress analysis would be beneficial. The client thinks features such as a team leaderboard would be beneficial to the simulation.

Updated Feature List

Development 3 has successfully been implemented, with all of the features in the feature list implemented. Testing was a success with a minimal amount of errors found. The following is the updated feature list, with green, teal and pink representing features implemented in Development 1, 2 and 3 respectively.

- Ability to create and login to accounts (client)
- Ability to join and trade on a team account (client)
- Ability for users to be designated as admins (client)
- Ability for account progress on team and personal accounts to be saved between sessions (inferred from client and user)
- Ability for admins to view teams list (inferred from client)
- Ability for admins to view team details and progress (inferred from client)
- Ability to view real-time information for all FTSE 100 stocks (client and user)
- Ability to select an amount of stocks and buy that amount using an up to date virtual balance, at real current price (client/user)
- Display for all stocks currently held in portfolio (client/SIC)
- Ability to sell stocks in portfolio at real current price (client)
- Graphs to display current day price trends of all stocks (user)
- Graph to show all time price changes of all stocks (inferred from user)
- Ability to create price alerts and be notified when stock reaches current price (inferred from user)
- Interface allowing users to see all current alerts on their account (inferred from user)
- Interface allowing user to see entire trade history (SIC)
- Interface allowing user to see all stocks in the FTSE 100 in a single screen, with details such as price (SIC)
- Notes section displayed in trade history and portfolio with reasons for trade decision (user)

The only feature that has not been implemented is the historic graph. The user reported that the ability to see the entire price history of a stock would be a useful tool to provide insight into price patterns over the companies trading history. However, during development I was unable to find a data source to provide, in most cases over a decade worth of stock data. Furthermore, if I was able to find this data source, plotting the data into a graph would be impractical, as it would contain millions of data points over a period of up to 20 years. One could argue that select data points every week or month could be selected and plotted, but this would still require a large amount of processing power that is not practical on the school computers.

For this reason, the graph to show the entire price history has been excluded from the program. Instead, the user has access to an intra-day graph that shows data from the current trading day.

Analysing Objectives

At the beginning of the project, a series of objectives were described, derived from interviews with the client and users, and research of stock market simulators that already exists. That list of objectives is displayed below. The colour code represents the extent to which the objectives have been met.

Objective	Success?
Create an intuitive interface with easy-to-use controls	From the feedback received from the client and the user, it seems that they are very pleased with the intuitive, easy to use nature of the program. I would therfore say that this objective has been met
Ability to select stocks and view the price info within a 30 second delay of real price	This objective has been met, with an even smaller delay than allowed. The estimated delay from real time prices is around 5-10 seconds. (Note that this only applies to the drop-down box, and does not include the graph, which displays data at around 100 second delay)
Visualization of price changes of current stock, and ability to display historic price data	This objective has also been met. A graph system has been implemented, that displays the stock data from the current day.
Ability for users to buy and sell stocks using a virtual balance that is kept up to date.	The balance system works effectively, with high accuracy. The system allows users to keep a balance that will be saved when they log out.
Ability for users to trade on a private account or on a team account	The team system was integrated into the simulation during Development 2. The system has proven effective and achieved its purpose, according to the interviews the client and users.
Ability for teacher to create teams and observe their progress	The AdminView form was successfully been added during Development 3. It allows the teacher to create teams and observe their progress.
Implementation of all secondary features stated in the feature list, such as a news feed, trade history, alerts system etc.	Every feature in the secondary feature list that was created through feedback from client and user interviews, except one. That feature is the historic graph, that was unable to be added as a data source could not be found to provide data from previous years. This is explained in the previous section.

Changes for Future Developments

Development 3 puts Investu in a state that is acceptable for the client, and achieves the goal set out in the analysis: create a real-time stock market investment simulator for students to practice trading in preperation for the Student Investor Challenge.

However, there is definitely more that could be added to Investu to develop it further. Based on feedback received from users and the client, the following are potential updates that may come in future developments of Investu:

Improved analyitcal capabilities

Both the users and client mentioned that the analytical aspect of the simulation could be improved to help them with their respective aims.

For the user, this would include analysis of both quantitative and qualititive data, both of their trading progress and stock market data, to produce insightful and reliable results. For example, the analysis of ability for users to analyse quantitative price data of certain stocks, and potentially extrapolate future price changes. Combined with analysis of qualitative data such as market news relating to specific companies in the FTSE100, this could provide the user useful tools to aid their trading experience. Furthermore, analysis of user trading history could also be a beneficial addition. For example, the ability for users to see a summary of their trading history with profit/loss displayed as a function of time, and the ability to see ordered lists of most impactful/profitable trades and some form of analysis of these would definitely aid the user in recognising trends regarding their trading history, and potentially aid them in improving the effectiveness of their strategy.

For the client, improved analytical capabilities would entail the ability for the client to analyse the quantitative data that results from the trades made by their teams. In theory, an active group of 10 or so teams that use the simulation multiple times per day and make a significant number of trades over a period of time, could provide a useful dataset that could be analysed, and show trends that could benefit the students through improving their trading strategy. Furthermore, analysis of a number of attributes relating to team trading such as ‘profit per week’ or ‘highest risk trade’ could lead to interesting learning opportunities, in which the teacher, being able to see the leading and trailing teams in each area, could provide useful insights to teams, such as advising teams which areas they are trailing in in relation to other teams. This could also create an interesting competetive aspect to the simulation, in which users could compete in certain categories, and the teacher could award prizes to the team at the top of the leaderboard.

Improved Potential for Expansion of User Base

Currently, Investu is designed with the client in mind. In future, if this software proves to be successful in its aim, then teachers may wish to share the simulation. This could be done a number of ways, however the most interesting way would be via centralising the database and server program on a remote server, instead of locally, and then connecting to the database/server via the internet.

This would be beneficial for a number of reasons, one of which would be the resulting dataset. Multiple classes per school across multiple schools would produce a significant amount of data that could be

analysed, as well as creating scoreboards and competitions for teams. Another benefit would be protecting the database. Currently, the database has to be stored on the local system, and in order for users to access it on their simulation, it needs to be in a public area. This leaves the database vulnerable, as it is easily accessible. Even if the database was in read-only mode and protected, the fact that it is stored in the public file system is not ideal.

However, in order to do this, the program itself needs significant changes. One of these changes would be an upgraded admin system, to allow for multiple admins to use the system simultaneously. Currently, the only way to create an admin is to hard-code the credentials into the database. This would ofcourse not be possible with a remote, encrypted database which was accessible to hundreds of students. The program would therefore need to be updated to allow for the creation of admins, and a new attribute would need to be added to the team table to show which admin created which team. This would then allow for only the teams of each admin to display in the AdminView form.

A further change necessary would be in the connection to the database. The database is currently connected to locally as it exists in the local file system. If the database was stored on a remote server, then the connection for each query would need to be changed to account for that.

Further changes that would increase the possibility of one day having a larger, multi-school user base would be an overhaul of the hard-coded values in the system. Currently, there are a lot of values that have been written into the code that cannot be changed. Some of these include database file locations. In order for the user base to expand successfully, as many of these hard-coded values as possible would need to be changed so that they could be editted by the user or teacher.

Expanded Access to Indexes and Securities

Currently, the simulation is hard-coded to only process FTSE100 information. In future versions, it would be beneficial to be able to trade on a range of different markets. The ability to trade on NASDAQ, S&P500, FTSE250, Dow Jones Industrial Average and other indexes would theoretically not be too difficult, as it would require a similar process as the FTSE100 that is already implemented. The difficult part would be revamping the current code to accept other indexes, as almost all instances relating the list of symbols has been hard-coded to 100, representing the 100 companies in the FTSE100. If this symbol list was suddenly expanded to more than 100, then almost all aspects of the code would cease to function.

Securities are tradeable financial assets. Not all securities are stocks and shares. Securities can include things like debt, equity, currencies, futures and options. The integration of these securities into Investu would greatly increase the complexity of the simulation, closer to a real life trading experience. In the future, this could be a good step for the simulation to take, as it would provide a lot of new content for users to try, and increase the realism.

Appendix 1 – Development 3 Code

MainForm

```
Imports System.IO
Imports System.Xml
Imports System.Windows.Forms.DataVisualization.Charting
Imports System.Data.OleDb

Public Class MainForm

    Public AccessDatabaseConnection As String = "Provider=Microsoft.ACE.OLEDB.12.0;Data
Source=; Data Source=R:\Students\Computing ICT\Handin\Year13\StockInfoDB.mdb"

    Public OpenPositions As New List(Of StockAttributes)
    Public Symbols As New List(Of String)

    Dim Series1 As New Series
    Dim LastValue As Decimal

    Public AccountID As Integer
    Public TeamMode As Boolean
    Public TeamName As String
    Public Balance As Decimal

    Public ErrorMsg As String

    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles MyBase.Load

        AccountID = LoginForm.AccountID
        TeamMode = LoginForm.TeamMode

        If TeamMode = True Then
            TeamName = LoginForm.TeamName
            LoginLabel.Text = "Welcome, " & LoginForm.Username & ". (" & TeamName & ")"
        ElseIf TeamMode = False Then
            TeamName = "0"
            LoginLabel.Text = "Welcome, " & LoginForm.Username & ""
        End If

        Balance = Math.Round(FetchBalance(), 2)
        BalanceBox.Text = "£" & Math.Round((Balance / 100), 2)

        FetchOpenPositions()
        FetchWorldNews()
        FetchMarketNews()
        FetchTradeHistory()
        LoadDetailsGrid()
        FetchAlerts()
        CreateChart()
        GraphSettings()


        GraphScaleComboBox.SelectedItem = "2"

        PopulateSymbolArray("C:\Users\Joe\Documents\visual studio 2010\Investu
Writeup\Development 3\StockSymbols.csv")
        For L = 0 To Symbols.Count - 1
```

```
            SelectStockComboBox.Items.Add(Symbols(L))
        Next

    End Sub

    Public Sub PopulateSymbolArray(ByVal FilePath As String)

        Dim CSVData() As String

        Using SR As New StreamReader(FilePath)

            While Not SR.EndOfStream
                CSVData = SR.ReadLine().Split(",")

                If String.IsNullOrEmpty(CSVData(0)) Then
                    MsgBox("Error loading FTSE 100")
                Else
                    Symbols.Add(CSVData(0).Trim)
                End If

            End While
        End Using

    End Sub

    Sub FetchAlerts()

        Dim QueryString As String

        If TeamMode Then
            QueryString = "SELECT * FROM tblAlerts WHERE TeamName='" & TeamName & "'"
        Else
            QueryString = "SELECT * FROM tblAlerts WHERE TeamName='0' AND AccountID=" &
AccountID & ""
        End If

        Dim ConnectionDb As New OleDbConnection(AccessDatabaseConnection)
        If ConnectionDb.State = ConnectionState.Closed Then ConnectionDb.Open()

        Dim cmd As OleDbCommand = ConnectionDb.CreateCommand
        cmd.CommandText = QueryString
        Dim SQLReply As OleDbDataReader = cmd.ExecuteReader

        For Each Record In SQLReply
            If TeamMode Then
                AlertsIDListBox.Items.Add(Record.Item("AlertID"))
                AlertsListBox.Items.Add(GetNameUsingID(Record.item("AccountID")) & " - "
& Record.item("StockSymbol") & " - " & Record.item("AlertPrice"))
            Else
                AlertsIDListBox.Items.Add(Record.Item("AlertID"))
                AlertsListBox.Items.Add(Record.item("StockSymbol") & " - " &
Record.item("AlertPrice"))
            End If

        Next

        ConnectionDb.Close()

    End Sub

    Sub DeleteAlert(ByVal AlertID As Integer)
```

```
        MsgBox(AlertID)
        Dim ConnectionDb As New OleDbConnection(AccessDatabaseConnection)
        If ConnectionDb.State = ConnectionState.Closed Then ConnectionDb.Open()
        Dim cmd As OleDbCommand = ConnectionDb.CreateCommand
        cmd.CommandText = "DELETE * FROM tblAlerts WHERE AlertID=" & AlertID & ""
        cmd.ExecuteNonQuery()
        ConnectionDb.Close()
    End Sub

    Function GetNameUsingID(ByVal AccountID)
        Dim AccountName As String = ""

        Dim ConnectionDb As New OleDbConnection(AccessDatabaseConnection)
        If ConnectionDb.State = ConnectionState.Closed Then ConnectionDb.Open()

        Dim cmd As OleDbCommand = ConnectionDb.CreateCommand
        cmd.CommandText = "SELECT Username FROM tblUserInfo WHERE AccountID=" & AccountID
& ""
        Dim SQLReply As OleDbDataReader = cmd.ExecuteReader

        For Each Record In SQLReply
            AccountName = Record.item("Username")
        Next

        ConnectionDb.Close()
        Return AccountName

    End Function

    Private Sub Timer1_Tick(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles Timer1.Tick

        Dim StockInfoString

        Try
            Timer1.Interval = 60000
            StockInfoString = FetchStockDetailsString(SelectStockComboBox.SelectedItem)
            NameBox.Text = SplitStockInfo(StockInfoString, "Name")
            PriceBox.Text = SplitStockInfo(StockInfoString, "Price")
            ChangeBox.Text = SplitStockInfo(StockInfoString, "Change")

            Series1.Points.Clear()
            Plot24hrData()
            LoadDetailsGrid()

            UpdatePortfolio()

            GraphSettings()

            VolatilityBox.Text = CalculateVolatility(PriceBox.Text, ChangeBox.Text) & "%"

        Catch ex As Exception
            MsgBox(ex.ToString())
        End Try

    End Sub

    Sub FetchTradeHistory()

        Dim MyConnection As OleDbConnection
        Dim Adapter As OleDbDataAdapter
```

```
        Dim DataSet As DataSet
        Dim Tables As DataTableCollection
        Dim Source As New BindingSource

        MyConnection = New OleDbConnection
        MyConnection.ConnectionString = AccessDatabaseConnection
        DataSet = New DataSet
        Tables = DataSet.Tables

        If TeamMode Then
            Adapter = New OleDbDataAdapter("SELECT * FROM tblTradeHistory WHERE
TeamName='" & TeamName & "'", MyConnection)
        Else
            Adapter = New OleDbDataAdapter("SELECT * FROM tblTradeHistory WHERE
AccountID=" & AccountID & " AND TeamName='0'", MyConnection)
        End If

        Adapter.Fill(DataSet, "[tblTradeHistory]")
        Dim View As New DataView(Tables(0))
        Source.DataSource = View
        DataGridView1.DataSource = View

        DataGridView1.Columns(0).Width = 50
        DataGridView1.Columns(1).Width = 50
        DataGridView1.Columns(2).Width = 50
        DataGridView1.Columns(3).Width = 75
        DataGridView1.Columns(4).Width = 75
        DataGridView1.Columns(5).Width = 125
        DataGridView1.Columns(6).Width = 50
        DataGridView1.Columns(7).Width = 50
        DataGridView1.Columns(8).Width = 80

    End Sub

    Sub LoadDetailsGrid()

        Dim MyConnection As OleDbConnection
        Dim Adapter As OleDbDataAdapter
        Dim DataSet As DataSet
        Dim Tables As DataTableCollection
        Dim Source As New BindingSource

        MyConnection = New OleDbConnection
        MyConnection.ConnectionString = AccessDatabaseConnection
        DataSet = New DataSet
        Tables = DataSet.Tables
        Adapter = New OleDbDataAdapter("SELECT * FROM [tblStockDetails]", MyConnection)
        Adapter.Fill(DataSet, "tblStockDetails")
        Dim View As New DataView(Tables(0))
        Source.DataSource = view
        StockDetailsGrid.DataSource = view

        StockDetailsGrid.Columns(0).Width = 66
        StockDetailsGrid.Columns(1).Width = 140
        StockDetailsGrid.Columns(4).Width = 187


    End Sub

    Function CalculateVolatility(ByVal Price As Decimal, ByVal Change As Decimal)
        Dim Volatility As Decimal
```

```
        Price = Math.Abs(Price)
        Change = Math.Abs(Change)

        If Price <> 0 And Change <> 0 Then
            Volatility = (Change / Price) * 100
            Volatility = Math.Round(Volatility, 2)
        Else
            Volatility = 0
        End If


        Return Volatility

    End Function

    Function FetchBalance()

        Dim CommandString As String

        If TeamMode = True Then
            CommandString = "SELECT Balance FROM tblTeams WHERE TeamName='" & TeamName &
"'"
        Else
            CommandString = "SELECT Balance FROM tblUserInfo WHERE AccountID=" &
AccountID & ""
        End If

        Using Connection As New OleDbConnection(AccessDatabaseConnection)

            Dim Command As New OleDbCommand(CommandString, Connection)
            Connection.Open()
            Dim reader As OleDbDataReader = Command.ExecuteReader()

            While reader.Read()
                Balance = reader(0)
            End While

            reader.Close()
        End Using

        Return Balance
    End Function

    Sub FetchOpenPositions()

        Dim ConnectionDb As New OleDbConnection(AccessDatabaseConnection)
        If ConnectionDb.State = ConnectionState.Closed Then ConnectionDb.Open()

        Dim cmd As OleDbCommand = ConnectionDb.CreateCommand

        If TeamMode = True Then
            cmd.CommandText = "SELECT StockSymbol, StockName, StockQuantity, BuyPrice,
AccountID, OpenPositionID, TradeDate FROM tblOpenPositions WHERE
tblOpenPositions.TeamName='" & TeamName & "'"
        Else
            cmd.CommandText = "SELECT StockSymbol, StockName, StockQuantity, BuyPrice,
AccountID, OpenPositionID, TradeDate FROM tblOpenPositions WHERE AccountID='" & AccountID
& "' AND TeamName='0'"

        End If
```

```
        Dim SQLReply As OleDbDataReader = cmd.ExecuteReader

        For Each Record In SQLReply
            OpenPositions.Add(New StockAttributes With {.StockSymbol =
Record.item("StockSymbol"), .StockValue = Record.item("BuyPrice"), .StockQuantity =
Record.item("StockQuantity"), .OpenPositionID = Record.item("OpenPositionID"), .StockName
= Record.item("StockName"), .BuyDate = Record.item("TradeDate")})
            UpdatePortfolio()

        Next

        ConnectionDb.Close()

    End Sub

    Function FetchStockDetailsString(ByVal StockSymbol As String)

        Dim InformationString As String = ""

        Dim Document As XmlDocument
        Dim Nodelist As XmlNodeList
        Dim Node As XmlNode

        Document = New XmlDocument()

Document.Load("https://spreadsheets.google.com/feeds/list/0AhySzEddwIC1dEtpWF9hQUhCWURZNE
ViUmpUeVgwdGc/1/public/basic?sq=symbol=" & StockSymbol)
        Nodelist = Document.GetElementsByTagName("entry")

        For Each Node In Nodelist
            InformationString = Node.ChildNodes.Item(4).InnerText
        Next

        Return InformationString

    End Function

    Sub FetchMarketNews()

        WebBrowser2.DocumentText = ""

        Dim stocknews As String = ""

        Try
            Dim document As XmlDocument
            Dim DescriptionNodes As XmlNodeList

            document = New XmlDocument()

document.Load("https://finance.google.com/finance/company_news?q=INDEXFTSE:UKX&ei=HynOWZC
1MpKKUunwl-gF&output=rss")

            DescriptionNodes = document.GetElementsByTagName("description")

            For L = 1 To DescriptionNodes.Count - 1
                stocknews += DescriptionNodes.Item(L).InnerText
                MsgBox("")
            Next

            MsgBox("")
```

```
            WebBrowser2.DocumentText = stocknews

        Catch errorVariable As Exception

            MsgBox(errorVariable.ToString())
        End Try

    End Sub

    Sub FetchWorldNews()

        WebBrowser1.DocumentText = ""
        Dim StockNews As String = ""

        Try
            Dim document As XmlDocument
            Dim TitleNodes, DescriptionNodes, LinkNodes, ArticleNodes As XmlNodeList

            document = New XmlDocument()
            document.Load("http://feeds.bbci.co.uk/news/world/rss.xml")

            TitleNodes = document.GetElementsByTagName("title")
            DescriptionNodes = document.GetElementsByTagName("description")
            LinkNodes = document.GetElementsByTagName("link")
            ArticleNodes = document.GetElementsByTagName("pubDate")

            For L = 0 To 25
                stocknews += "<font size=" & "+1" & ">" & "<b>" & TitleNodes.Item(L +
2).InnerText & "    " & "</b>" & "</font>"
                stocknews += "<font size=" & "-2" & ">" & ArticleNodes.Item(L).InnerText
& "</font>" & "<br>"
                stocknews += DescriptionNodes.Item(L + 1).InnerText & "<Br>"
                stocknews += "<font size=" & "-1" & ">" & "Read more at " &
LinkNodes.Item(L + 2).InnerText & "<font>" & "<Br><Br><Br>"
            Next

            WebBrowser1.DocumentText += StockNews

        Catch errorVariable As Exception
            MsgBox(errorVariable.ToString())
        End Try

    End Sub

    Function SplitStockInfo(ByVal StringToSplit As String, ByVal DetailsToExtract As
String)

        Dim ExtractedDetails As String = ""

        Dim ArrayList() As String = StringToSplit.Split(":")
        Dim SubArrayList() As String = ArrayList(1).Split(",")
        Dim SubArrayList1() As String = ArrayList(2).Split(",")

        Select Case DetailsToExtract

            Case "Name"
                If Trim(SubArrayList(0)) = "#N/A" Then
                    ExtractedDetails = "ERROR"
                Else
                    ExtractedDetails = Trim(SubArrayList(0))
                End If
```

```
            Case "Price"
                If Trim(SubArrayList1(0)) = "#N/A" Then
                    ExtractedDetails = "0"
                Else
                    ExtractedDetails = Trim(SubArrayList1(0))
                End If

            Case "Change"
                If Trim(ArrayList(3)) = "#N/A" Then
                    ExtractedDetails = "0"
                Else
                    ExtractedDetails = Trim(ArrayList(3))
                End If

        End Select

        Return ExtractedDetails
    End Function

    Private Sub BuyButton_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles BuyButton.Click
        If SelectStockComboBox.SelectedItem <> "" Then
            If PriceBox.Text <> "0" Then
                BuyForm.Show()
            Else
                MsgBox("You cannot buy this stock due to an error.")
            End If
        Else
            MsgBox("Please select the stock you wish to buy, from the drop down menu
provided.")
        End If

    End Sub

    Sub UpdatePortfolio()

        Dim TotalTradePrice As Decimal
        Dim CurrentTotalPrice As Decimal

        OpenPositionsListBox.Items.Clear()

        For l = 0 To OpenPositions.Count - 1
            TotalTradePrice = Math.Round(((OpenPositions(l).StockValue *
OpenPositions(l).StockQuantity) / 100), 2)

            CurrentTotalPrice = Math.Round(((GetStockPrice(OpenPositions(l).StockSymbol)
* OpenPositions(l).StockQuantity) / 100), 2)
            OpenPositionsListBox.Items.Add(OpenPositions(l).StockName & " - Bought " &
OpenPositions(l).StockQuantity & " FOR £" & TotalTradePrice & " (" &
OpenPositions(l).StockValue & " each)" & vbNewLine)
        Next

    End Sub

    Private Sub SelectStockComboBox_SelectedIndexChanged(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles SelectStockComboBox.SelectedIndexChanged

        Timer1.Start()

        Series1.Points.Clear()
```

```
        Timer1.Interval = 1
        Plot24hrData()

    End Sub

    Public Sub Plot24hrData()

        Dim Query As String = "SELECT FetchDate, StockPrice FROM tblStockPriceHistory
WHERE StockSymbol = '" & SelectStockComboBox.SelectedItem & "' ORDER BY FetchDate"

        Using connection As New OleDbConnection(AccessDatabaseConnection)
            Dim command As New OleDbCommand(Query, connection)

            connection.Open()

            Dim reader As OleDbDataReader = command.ExecuteReader()

            While reader.Read()

                If reader(0) >= DateTime.Today Then
                    PlotNewPoint((reader(0)).ToOADate(), reader(1))
                    LastValue = reader(1)
                End If

            End While

            reader.Close()
        End Using
    End Sub

    Sub PlotNewPoint(ByVal XValue As Decimal, ByVal YValue As Decimal)
        Series1.Points.AddXY(XValue, YValue)
    End Sub

    Function GetStockChange(ByVal StockSymbol As String)

        Dim StockChange As Decimal

        Try

            Dim document As XmlDocument
            Dim nodelist As XmlNodeList
            Dim node As XmlNode

            document = New XmlDocument()

document.Load("https://spreadsheets.google.com/feeds/list/0AhySzEddwIC1dEtpWF9hQUhCWURZNE
ViUmpUeVgwdGc/1/public/basic?sq=symbol=" & StockSymbol)
            nodelist = document.GetElementsByTagName("entry")

            For Each node In nodelist
                StockChange = SplitStockInfo(node.ChildNodes.Item(4).InnerText, "Change")

            Next

        Catch errorVariable As Exception
            Timer1.Stop()

        End Try
        Return StockChange
    End Function
```

```
    Function GetStockPrice(ByVal StockSymbol As String)

        Dim StockPrice As Decimal


        Try

            Dim document As XmlDocument
            Dim nodelist As XmlNodeList
            Dim node As XmlNode

            document = New XmlDocument()

document.Load("https://spreadsheets.google.com/feeds/list/0AhySzEddwIC1dEtpWF9hQUhCWURZNE
ViUmpUeVgwdGc/1/public/basic?sq=symbol=" & StockSymbol)
            nodelist = document.GetElementsByTagName("entry")

            For Each node In nodelist
                StockPrice = SplitStockInfo(node.ChildNodes.Item(4).InnerText, "Price")

            Next

        Catch errorVariable As Exception
            Timer1.Stop()
        End Try
        Return StockPrice

    End Function
    Function GetStockName(ByVal StockSymbol As String)

        Dim StockName As String = "Error"

        Try

            Dim document As XmlDocument
            Dim nodelist As XmlNodeList
            Dim node As XmlNode

            document = New XmlDocument()

document.Load("https://spreadsheets.google.com/feeds/list/0AhySzEddwIC1dEtpWF9hQUhCWURZNE
ViUmpUeVgwdGc/1/public/basic?sq=symbol=" & StockSymbol)
            nodelist = document.GetElementsByTagName("entry")

            For Each node In nodelist
                StockName = SplitStockInfo(node.ChildNodes.Item(4).InnerText, "Name")
            Next

        Catch errorVariable As Exception
            Timer1.Stop()

        End Try

        Return StockName

    End Function

    Private Sub ClosePositionsButton_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles ClosePositionsButton.Click

        Try
```

```
            Dim NewStockPrice As Decimal
            Dim SelectedStock As String = OpenPositionsListBox.SelectedIndex

            If OpenPositionsListBox.Items.Count > 0 And
OpenPositionsListBox.CheckedItems.Count > 0 Then

                NewStockPrice = GetStockPrice(OpenPositions(SelectedStock).StockSymbol)

                Balance = Balance + (OpenPositions(SelectedStock).StockQuantity *
NewStockPrice)

                BalanceBox.Text = "£" & Math.Round((Balance / 100), 2)

                Dim CommandString As String

                If TeamMode Then
                    CommandString = "UPDATE tblTeams SET tblTeams.Balance=" & Balance & "
WHERE tblTeams.TeamName='" & TeamName & "';"
                Else
                    CommandString = "UPDATE tblUserInfo SET tblUserInfo.Balance=" &
Balance & " WHERE (((tblUserInfo.AccountID)=" & AccountID & "));"
                End If


                Dim ConnectionDb As New OleDbConnection(AccessDatabaseConnection)
                If ConnectionDb.State = ConnectionState.Closed Then ConnectionDb.Open()

                Dim cmd As OleDbCommand = ConnectionDb.CreateCommand

                StoreNewTrade(OpenPositions(SelectedStock).OpenPositionID, NewStockPrice)

                cmd.CommandText = "DELETE * FROM tblOpenPositions WHERE OpenPositionID='"
& OpenPositions(SelectedStock).OpenPositionID & "' "
                cmd.ExecuteNonQuery()

                cmd.CommandText = CommandString
                cmd.ExecuteNonQuery()


                ConnectionDb.Close()

                OpenPositions.RemoveAt(OpenPositionsListBox.SelectedIndex)
                UpdatePortfolio()

            Else
                MsgBox("Please select the position you'd like to close.")
            End If

            FetchTradeHistory()
        Catch ex As Exception
            MsgBox(ex.ToString())
        End Try


    End Sub

    Private Sub Button1_Click(sender As System.Object, e As System.EventArgs) Handles
InfoButton.Click

        Dim SelectedStock As String = OpenPositionsListBox.SelectedIndex
```

```
        Dim Value As Decimal = OpenPositions(SelectedStock).StockValue
        Dim StockName As String = OpenPositions(SelectedStock).StockName
        Dim Quantity As String = OpenPositions(SelectedStock).StockQuantity
        Dim StockSymbol As String = OpenPositions(SelectedStock).StockSymbol


        Dim CurrentTotalPrice As Decimal = Math.Round(((GetStockPrice(StockSymbol) *
Quantity) / 100), 2)
        Dim TotalTradePrice As Decimal = Math.Round(((Value * Quantity) / 100), 2)

        MsgBox("You bought" & Quantity & " " & StockName & " shares for a price of " &
Value & " each, costing a total of £" & TotalTradePrice & ". " & vbNewLine & StockName &
" shares are now worth " & GetStockPrice(StockSymbol) & " each, making your shares worth
a total of £" & CurrentTotalPrice & "." & vbNewLine & "Your net gain from this trade is
£" & TotalTradePrice - CurrentTotalPrice & ".")
    End Sub

    Sub StoreNewTrade(ByVal OpenPositionID As String, ByVal CurrentPrice As Integer)

        Dim InsertString As String = ""

        Dim ConnectionDb As New OleDbConnection(AccessDatabaseConnection)
        If ConnectionDb.State = ConnectionState.Closed Then ConnectionDb.Open()

        Dim cmd As OleDbCommand = ConnectionDb.CreateCommand
        cmd.CommandText = "SELECT StockSymbol, StockQuantity FROM tblOpenPositions WHERE
OpenPositionID='" & OpenPositionID & "'"
        Dim SQLReply As OleDbDataReader = cmd.ExecuteReader

        For Each Record In SQLReply
            InsertString = "INSERT INTO tblTradeHistory (AccountID, StockSymbol,
StockQuantity, BuyOrSell, TradePrice, TradeDate, TeamName) VALUES ('" & AccountID & "','"
& Record.item("StockSymbol") & "','" & Record.item("StockQuantity") & "','Sell','" &
CurrentPrice & "','" & DateTime.Now & "','" & TeamName & "')"
        Next

        SQLReply.Close()

        cmd.CommandText = InsertString
        cmd.ExecuteNonQuery()
        ConnectionDb.Close()
    End Sub

    Private Sub CreateAlertButton_Click(sender As System.Object, e As System.EventArgs)
Handles CreateAlertButton.Click

        Dim UpOrDown As String

        If SelectStockComboBox.Text <> "Select Stock Symbol" Then
            If AlertPriceBox.Text > PriceBox.Text Then
                UpOrDown = "UP"
            Else
                UpOrDown = "DOWN"
            End If

            CreateNewAlert(UpOrDown)
            AlertsListBox.Items.Clear()
            AlertsIDListBox.Items.Clear()
            FetchAlerts()
        Else
```

```
            MsgBox("You need to select a stock from the drop down menu first.")
        End If




    End Sub

    Sub CreateNewAlert(ByVal UpOrDown As String)

        If ValidateAlertPrice(AlertPriceBox.Text) Then

            Dim ConnectionDb As New OleDbConnection(AccessDatabaseConnection)
            If ConnectionDb.State = ConnectionState.Closed Then ConnectionDb.Open()

            Dim cmd As OleDbCommand = ConnectionDb.CreateCommand
            cmd.CommandText = "INSERT INTO tblAlerts (AccountID, StockSymbol, AlertPrice,
TeamName, UpOrDown) VALUES (" & AccountID & ",'" & SelectStockComboBox.SelectedItem &
"','" & AlertPriceBox.Text & "','" & TeamName & "','" & UpOrDown & "')"
            cmd.ExecuteNonQuery()
            ConnectionDb.Close()

            MsgBox("A new alert for " & NameBox.Text & " at " & AlertPriceBox.Text & "
has been set.")
        Else
            MsgBox("Please enter a valid alert price.")
        End If

    End Sub

    Function ValidateAlertPrice(ByVal Price As String)

        Dim Numbers As New System.Text.RegularExpressions.Regex("[0-9]")
        Dim NotNumbers As New System.Text.RegularExpressions.Regex("[^0-9]")

        If Len(Price) < 2 Then Return False
        If NotNumbers.Matches(Price).Count > 0 Then Return False

        Return True
    End Function

    Private Sub LogoutButton_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles LogoutButton.Click

        LoginForm.Show()
        Me.Close()

    End Sub

    Private Sub GraphScaleComboBox_SelectedIndexChanged(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles GraphScaleComboBox.SelectedIndexChanged
        Timer1.Interval = 1
    End Sub

    Private Sub OpenPositionsListBox_ItemCheck(ByVal sender As Object, ByVal e As
System.Windows.Forms.ItemCheckEventArgs) Handles OpenPositionsListBox.ItemCheck

        If e.NewValue = CheckState.Checked Then
            For i As Integer = 0 To Me.OpenPositionsListBox.Items.Count - 1 Step 1
                If i <> e.Index Then Me.OpenPositionsListBox.SetItemChecked(i, False)
            Next
```

```
        End If
    End Sub

    Sub CreateChart()

        Series1.Name = SelectStockComboBox.SelectedItem
        Series1.ChartType = SeriesChartType.Line
        Series1.BorderWidth = 4
        Chart1.Series.Add(Series1)
        Chart1.Legends.Clear()
        Series1.XValueType = ChartValueType.DateTime
        Series1.BorderWidth = 2

    End Sub

    Sub GraphSettings()

        Chart1.ChartAreas(0).AxisY.Minimum = LastValue - Val(GraphScaleComboBox.Text)
        Chart1.ChartAreas(0).AxisY.Maximum = LastValue + Val(GraphScaleComboBox.Text)
        Chart1.Update()

    End Sub

    Private Sub Timer2_Tick(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles Timer2.Tick
        GraphSettings()
        Timer2.Interval = 5000

    End Sub

    Private Sub OpenToolStripButton_Click(sender As System.Object, e As System.EventArgs)
Handles OpenToolStripButton.Click

        Dim TeamCode As String
        TeamCode = InputBox("Please input the 5 character Team Code here, issued to you
by your teacher", "Join Team", "")

        If ValidTeamCode(TeamCode) Then
            If UserAlreadyInTeam(AccountID) Then
                DeleteUserFromTeam(AccountID)
                AddNewPlayerToTeam(AccountID, TeamCode)
            Else
                AddNewPlayerToTeam(AccountID, TeamCode)

            End If

            LoginForm.Show()
            Me.Close()
        Else
            MsgBox(ErrorMsg)
        End If


    End Sub

    Sub DeleteUserFromTeam(ByVal AccountID As Integer)

        Dim ConnectionDb As New OleDbConnection(AccessDatabaseConnection)
        If ConnectionDb.State = ConnectionState.Closed Then ConnectionDb.Open()

        Dim cmd As OleDbCommand = ConnectionDb.CreateCommand
```

```
        cmd.CommandText = "DELETE * FROM tblTeamUsers WHERE AccountID=" & AccountID & ""
        cmd.ExecuteNonQuery()

        MsgBox("You have been removed from your current team.")

        ConnectionDb.Close()

    End Sub

    Function UserAlreadyInTeam(ByVal AccountID As Integer)
        Dim AlreadyInTeam As Boolean

        Dim ConnectionDb As New OleDbConnection(AccessDatabaseConnection)
        If ConnectionDb.State = ConnectionState.Closed Then ConnectionDb.Open()

        Dim cmd As OleDbCommand
        Dim SQLReply As OleDbDataReader

        cmd = ConnectionDb.CreateCommand
        cmd.CommandText = "SELECT AccountID FROM tblTeamUsers WHERE AccountID=" &
AccountID & ""
        SQLReply = cmd.ExecuteReader

        For Each Record In SQLReply
            AlreadyInTeam = True
        Next
        ConnectionDb.Close()

        Return AlreadyInTeam
    End Function

    Function ValidTeamCode(ByVal TeamCode As String)

        If CheckTeamCodeExists(TeamCode) Then

            If EmptySpaceInTeam(TeamCode) Then
                Return True
            Else
                ErrorMsg = "The team you are trying to join is already full."
            End If
        Else
            ErrorMsg = "The Team Code you entered does not exist."
        End If

        Return False

    End Function

    Function CheckTeamCodeExists(ByVal TeamCode As String)

        If TeamCode = "" Then
            Return False
        End If

        Dim ConnectionDb As New OleDbConnection(AccessDatabaseConnection)
        If ConnectionDb.State = ConnectionState.Closed Then ConnectionDb.Open()

        Dim cmd As OleDbCommand
        Dim SQLReply As OleDbDataReader
```

```
        cmd = ConnectionDb.CreateCommand
        cmd.CommandText = "SELECT TeamCode FROM tblTeams"
        SQLReply = cmd.ExecuteReader

        For Each Record In SQLReply

            If Record.item("TeamCode") = TeamCode Then

                Return True
            End If
        Next
        ConnectionDb.Close()

        Return False
    End Function

    Function EmptySpaceInTeam(ByVal TeamCode As String)

        Dim EmptySpace As Boolean = False
        Dim TeamID As Integer
        Dim UsersAlreadyInTeam As Integer = 0

        Dim ConnectionDb As New OleDbConnection(AccessDatabaseConnection)
        If ConnectionDb.State = ConnectionState.Closed Then ConnectionDb.Open()

        Dim cmd As OleDbCommand
        Dim SQLReply As OleDbDataReader

        cmd = ConnectionDb.CreateCommand
        cmd.CommandText = "SELECT TeamID FROM tblTeams WHERE TeamCode='" & TeamCode & "'"
        SQLReply = cmd.ExecuteReader

        For Each Record In SQLReply
            TeamID = Record.item("TeamID")
        Next

        cmd = ConnectionDb.CreateCommand
        cmd.CommandText = "SELECT * FROM tblTeamUsers WHERE TeamID=" & TeamID & ""
        SQLReply = cmd.ExecuteReader

        For Each Record In SQLReply
            UsersAlreadyInTeam += 1
        Next

        If UsersAlreadyInTeam < 4 Then
            EmptySpace = True
        End If

        ConnectionDb.Close()

        Return EmptySpace
    End Function

    Sub AddNewPlayerToTeam(ByVal AccountID As Integer, ByVal TeamCode As String)


        Dim TeamID As Integer
        Dim TeamName As String = ""


        Dim ConnectionDb As New OleDbConnection(AccessDatabaseConnection)
```

```
        If ConnectionDb.State = ConnectionState.Closed Then ConnectionDb.Open()

        Dim cmd As OleDbCommand
        Dim SQLReply As OleDbDataReader

        cmd = ConnectionDb.CreateCommand
        cmd.CommandText = "SELECT TeamID, TeamName FROM tblTeams WHERE TeamCode='" &
TeamCode & "'"

        SQLReply = cmd.ExecuteReader

        For Each Record In SQLReply
            TeamID = Record.item("TeamID")
            TeamName = Record.item("TeamName")
        Next

        SQLReply.Close()

        cmd.CommandText = "INSERT INTO tblTeamUsers (AccountID, TeamID) VALUES (" &
AccountID & ",'" & TeamID & "')"

        MsgBox("You have joined " & TeamName & "")

        cmd.ExecuteNonQuery()

        ConnectionDb.Close()

    End Sub

    Private Sub Button1_Click_1(sender As Object, e As EventArgs) Handles Button1.Click
        DeleteAlert(AlertsIDListBox.SelectedItem)
        AlertsListBox.Items.Clear()
        AlertsIDListBox.Items.Clear()
        FetchAlerts()
    End Sub
End Class



Public Class StockAttributes

    Public StockSymbol As String
    Public StockName As String
    Public OpenPositionID As String
    Public StockValue As Decimal
    Public StockQuantity As Integer
    Public BuyDate As DateTime

End Class
```

BuyForm

```
Imports System.Data.OleDb

Public Class BuyForm

    Dim Quantity As Integer
    Dim StockPrice As Decimal = MainForm.PriceBox.Text
    Dim Stocksymbol As String = MainForm.SelectStockComboBox.SelectedItem
    Dim Stockname As String = MainForm.NameBox.Text

    Dim FinalPrice As Decimal

    Dim TeamMode As Boolean
    Dim TeamName As String

    Dim AccessDatabaseConnection As String = MainForm.AccessDatabaseConnection


    Private Sub Form2_Load(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles MyBase.Load

        Quantity = 1
        QuantityBox.Text = Quantity

        FinalPrice = (StockPrice * Quantity)

        PriceBox.Text = "£" & Math.Round((FinalPrice) / 100, 2)

        TeamMode = MainForm.TeamMode
        TeamName = MainForm.TeamName

        StockDisplayBox.Clear()
        StockPriceBox.Clear()

        StockDisplayBox.Text = Stockname
        StockPriceBox.Text = StockPrice
    End Sub

    Private Sub TrackBar1_Scroll(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles QuantitySlider.Scroll


        QuantitySlider.Maximum = Int(MainForm.Balance / StockPrice)
        Quantity = QuantitySlider.Value
        QuantityBox.Text = Quantity

        FinalPrice = 1.2 * (StockPrice * Quantity)
        PriceBox.Text = "£" & Math.Round((FinalPrice) / 100, 2)

    End Sub

    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles Button1.Click

        If NotesBox.TextLength > 255 Then
            MsgBox("Your note is too long.")
        Else
```

```
            For L = 0 To MainForm.Symbols.Count - 1

                If MainForm.Symbols(L) = MainForm.SelectStockComboBox.SelectedItem Then

                    If MainForm.Balance > (Quantity * StockPrice) Then
                        MainForm.OpenPositions.Add(New StockAttributes With {.StockSymbol
= MainForm.Symbols(L), .StockValue = StockPrice, .StockQuantity =
Quantity, .OpenPositionID = MainForm.AccountID & DateTime.Now, .BuyDate =
DateTime.Now, .StockName = Stockname})
                        StoreNewPosition(MainForm.AccountID, Stockname, Stocksymbol,
Quantity, StockPrice, DateTime.Now, TeamName, NotesBox.Text)
                        UpdateBalance(MainForm.AccountID)
                        MainForm.UpdatePortfolio()

                        Me.Close()
                    Else
                        MsgBox("You don't have enough money to buy that many " &
Stockname & " stocks.")
                    End If
                End If
            Next
            MainForm.FetchTradeHistory()
        End If
    End Sub
    Sub UpdateBalance(ByVal AccountID As Integer)

        MainForm.Balance = MainForm.Balance - ((Quantity * StockPrice))
        MainForm.BalanceBox.Text = "£" & Math.Round((MainForm.Balance / 100), 2)


        Dim CommandString As String
        If TeamMode Then
            CommandString = "UPDATE tblTeams SET tblTeams.Balance=" & MainForm.Balance &
" WHERE tblTeams.TeamName='" & TeamName & "';"

        Else
            CommandString = "UPDATE tblUserInfo SET tblUserInfo.Balance=" &
MainForm.Balance & " WHERE (((tblUserInfo.AccountID)=" & AccountID & "));"
        End If

        Dim ConnectionDb As New OleDbConnection(AccessDatabaseConnection)
        If ConnectionDb.State = ConnectionState.Closed Then ConnectionDb.Open()

        Dim cmd As OleDbCommand = ConnectionDb.CreateCommand

        cmd.CommandText = CommandString

        cmd.ExecuteNonQuery()
        ConnectionDb.Close()
    End Sub
    Sub StoreNewPosition(ByVal ID As Integer, ByVal StockName As String, ByVal
StockSymbol As String, ByVal StockQuantity As Integer, ByVal StockValue As Decimal, ByVal
BuyDate As Date, ByVal TeamName As String, ByVal Notes As String)


        Dim ConnectionDb As New OleDbConnection(AccessDatabaseConnection)
        If ConnectionDb.State = ConnectionState.Closed Then ConnectionDb.Open()

        Dim cmd As OleDbCommand = ConnectionDb.CreateCommand
        cmd.CommandText = "INSERT INTO TblOpenPositions (AccountID, StockName,
StockSymbol, StockQuantity, BuyPrice, TradeDate, OpenPositionID, TeamName) VALUES ('" &
```

```
ID & "','" & StockName & "','" & StockSymbol & "','" & StockQuantity & "','" & StockValue
& "','" & BuyDate & "','" & ID & BuyDate & "','" & TeamName & "')"

        cmd.ExecuteNonQuery()

        cmd.CommandText = "INSERT INTO TblTradeHistory (AccountID, StockSymbol,
StockQuantity, BuyOrSell, TradePrice, TradeDate, TeamName, Notes) VALUES ('" & ID & "','"
& StockSymbol & "','" & StockQuantity & "','Buy','" & StockValue & "','" & BuyDate &
"','" & TeamName & "','" & Notes & "')"

        cmd.ExecuteNonQuery()
        ConnectionDb.Close()
    End Sub
End Class
```

SignUpForm

```
Imports System.Data.OleDb

Public Class SignUpForm

    Dim AccessDatabaseConnection As String = MainForm.AccessDatabaseConnection

    Dim ErrorMsg As String
    Dim EmptySlot As String

    Private Sub SignUpButton_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles CreateAccountButton.Click

        If ProceedToSignUp() Then
            CreateNewAccount(UsernameBox.Text, PasswordBox.Text, EmailBox.Text)

            LoginForm.UsernameTextBox.Text = UsernameBox.Text
            LoginForm.PasswordTextBox.Text = PasswordBox.Text

            MsgBox("Your account has been created! Click login to proceed.")

            Me.Close()
        End If

    End Sub

    Sub CreateNewAccount(ByVal Username As String, ByVal Password As String, ByVal Email
As String)

        Dim Balance As Integer = 10000000

        Dim TeamName As String = ""

        Dim ConnectionDb As New OleDbConnection(AccessDatabaseConnection)
        If ConnectionDb.State = ConnectionState.Closed Then ConnectionDb.Open()

        Dim cmd As OleDbCommand
        cmd = ConnectionDb.CreateCommand

        cmd.CommandText = "INSERT INTO tblUserInfo (Username, Balance, Passwrd) VALUES
('" & Username & "','" & Balance & "','" & Password & "')"

        cmd.ExecuteNonQuery()

        ConnectionDb.Close()

    End Sub

    Function ProceedToSignUp()

        If ValidatePassword(PasswordBox.Text) Then
            If ValidUsername(UsernameBox.Text) Then
                If ValidEmail(EmailBox.Text) Then
                    Return True
                Else
                    MsgBox("The email you have entered is invalid")
                End If
```

```
            Else
                MsgBox("The username you have entered is already taken.")
            End If
        Else
            MsgBox("Invalid Password - Passwords must have at least 1 upper case
character, 1 number and 8 total characters.")
        End If

        Return False
    End Function

    Function ValidUsername(ByVal NewUsername As String)

        If UsernameBox.Text = "" Or PasswordBox.Text = "" Then
            ErrorMsg = "The Username and Password are required fields."
            Return False
        Else

            Dim ConnectionDb As New OleDbConnection(AccessDatabaseConnection)
            If ConnectionDb.State = ConnectionState.Closed Then ConnectionDb.Open()

            Dim cmd As OleDbCommand
            Dim SQLReply As OleDbDataReader

            cmd = ConnectionDb.CreateCommand
            cmd.CommandText = "SELECT Username FROM tblUserInfo"
            SQLReply = cmd.ExecuteReader

            For Each Record In SQLReply

                If Record.item("Username") = NewUsername Then
                    Return False
                End If
            Next

            ConnectionDb.Close()

        End If
        Return True
    End Function

    Function ValidatePassword(ByVal Password As String, Optional ByVal MinLength As
Integer = 8, Optional ByVal NumUpper As Integer = 1, Optional ByVal NumLower As Integer =
1, Optional ByVal NumNumbers As Integer = 1, Optional ByVal NumSpecial As Integer = 0) As
Boolean

        Dim UpperCase As New System.Text.RegularExpressions.Regex("\p{Lu}")
        Dim LowerCase As New System.Text.RegularExpressions.Regex("[a-z]")
        Dim Numbers As New System.Text.RegularExpressions.Regex("[0-9]")

        Dim Specials As New System.Text.RegularExpressions.Regex("[^a-zA-Z0-9]")


        If Len(Password) < MinLength Then Return False

        If UpperCase.Matches(Password).Count < NumUpper Then Return False
        If LowerCase.Matches(Password).Count < NumLower Then Return False
        If Numbers.Matches(Password).Count < NumNumbers Then Return False
        If Specials.Matches(Password).Count < NumSpecial Then Return False

        Return True
```

```
    End Function


    Function ValidEmail(ByVal Email As String)
        Dim Valid As Boolean = False

        If Email = "" Then
            Valid = True
        Else
            Valid = True
        End If
        Return Valid

    End Function

End Class
```

AdminViewForm

```
Imports System.Data.OleDb
Public Class AdminView

    Dim AccessDatabaseConnection As String = MainForm.AccessDatabaseConnection

    Private Sub AdminView_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
        FetchTeams()
    End Sub

    Sub FetchTeams()

        Dim TeamID As Integer
        Dim TeamInfo As String


        TeamInfoCheckedListbox.Items.Clear()
        TeamIdCheckedListBox.Items.Clear()

        Dim ConnectionDb As New OleDbConnection(AccessDatabaseConnection)
        If ConnectionDb.State = ConnectionState.Closed Then ConnectionDb.Open()
        Dim cmd As OleDbCommand = ConnectionDb.CreateCommand
        cmd.CommandText = "SELECT TeamID, TeamName, Balance, TeamCode FROM tblTeams"

        Dim SQLReply As OleDbDataReader = cmd.ExecuteReader

        For Each Record In SQLReply
            TeamID = Record.item("TeamID")
            TeamIdCheckedListBox.Items.Add(TeamID)

            TeamInfo = Record.item("TeamName") & " - " & Record.Item("TeamCode") & " - "
& Record.item("Balance")
            TeamInfoCheckedListbox.Items.Add(TeamInfo)
        Next

        ConnectionDb.Close()
    End Sub


    Function ValidateInputs(ByVal NewTeamName As String, ByVal NewTeamCode As String)
        Dim ValidTeamInfo As Boolean = True

        If TeamNameBox.Text = "" Or TeamCodeBox.Text = "" Or BalanceBox.Text = "" Then

            ValidTeamInfo = False
        Else

            Dim ConnectionDb As New OleDbConnection(AccessDatabaseConnection)
            If ConnectionDb.State = ConnectionState.Closed Then ConnectionDb.Open()

            Dim cmd As OleDbCommand = ConnectionDb.CreateCommand

            cmd.CommandText = "SELECT TeamName, TeamCode FROM tblTeams"

            Dim SQLReply As OleDbDataReader = cmd.ExecuteReader
```

```
            For Each Record In SQLReply

                If Record.item("TeamName") = NewTeamName Or Record.item("TeamCode") =
NewTeamCode Then
                    ValidTeamInfo = False
                End If
            Next

        End If
        Return ValidTeamInfo
    End Function


    Sub CreateNewTeam(ByVal NewTeamName As String, ByVal NewTeamCode As String)

        Dim Balance As Integer = BalanceBox.Text * 100

        Dim ConnectionDb As New OleDbConnection(AccessDatabaseConnection)
        If ConnectionDb.State = ConnectionState.Closed Then ConnectionDb.Open()

        Dim cmd As OleDbCommand = ConnectionDb.CreateCommand

        cmd.CommandText = "INSERT INTO tblTeams (TeamName, TeamCode, Balance) VALUES ('"
& NewTeamName & "','" & NewTeamCode & "','" & Balance & "')"
        cmd.ExecuteNonQuery()

        ConnectionDb.Close()

        MsgBox("A new team with the name " & NewTeamName & " and team code " &
NewTeamCode & " has been created.")

        FetchTeams()

    End Sub

    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles Button1.Click
        If ValidateInputs(TeamNameBox.Text, TeamCodeBox.Text) Then
            CreateNewTeam(TeamNameBox.Text, TeamCodeBox.Text)
        Else
            MsgBox("There was an error creating the new team. Your team name may already
be taken or you have entered invalid information.")
        End If
    End Sub

    Private Sub TeamIdCheckedListBox_ItemCheck(ByVal sender As Object, ByVal box As
System.Windows.Forms.ItemCheckEventArgs) Handles TeamIdCheckedListBox.ItemCheck

        If box.NewValue = CheckState.Checked Then

            For index = 0 To TeamIdCheckedListBox.Items.Count - 1

                If index <> box.Index Then
                    Me.TeamIdCheckedListBox.SetItemChecked(index, False)
                    Me.TeamInfoCheckedListbox.SetItemChecked(index, False)
                Else
                    TeamInfoCheckedListbox.SetItemChecked(index, True)
                End If
            Next
        End If
    End Sub
```

```
    Sub FetchTeamInfo()

        TeamDetailsListBox.Items.Clear()

        Dim TeamID As Integer = TeamIdCheckedListBox.Text
        Dim Balance As Integer
        Dim TeamName As String = ""


        Dim ConnectionDb As New OleDbConnection(AccessDatabaseConnection)
        If ConnectionDb.State = ConnectionState.Closed Then ConnectionDb.Open()

        Dim cmd As OleDbCommand
        Dim SQLReply As OleDbDataReader

        cmd = ConnectionDb.CreateCommand
        cmd.CommandText = "SELECT * FROM tblTeams WHERE TeamID=" & TeamID & ""

        SQLReply = cmd.ExecuteReader

        For Each Record In SQLReply

            TeamName = Record.item("TeamName")
            Balance = Record.item("Balance") / 100

            TeamDetailsListBox.Items.Add("You are viewing the details of " &
Record.item("TeamName") & " - Team Code: " & Record.Item("TeamCode"))

            TeamDetailsListBox.Items.Add("The team currently has £" & Balance)
            TeamDetailsListBox.Items.Add("")
            TeamDetailsListBox.Items.Add("The following are the members of this team:")


            FetchUsersInTeam(TeamID)
        Next

        TeamDetailsListBox.Items.Add("")
        TeamDetailsListBox.Items.Add(TeamName & " has the following open positions;")

        cmd = ConnectionDb.CreateCommand

        cmd.CommandText = "SELECT * FROM tblOpenPositions WHERE TeamName='" & TeamName &
"'"

        SQLReply = cmd.ExecuteReader

        For Each Record In SQLReply
            TeamDetailsListBox.Items.Add(Record.item("StockSymbol") & "  -  " &
Record.item("StockQuantity") & "  -  " & Record.item("BuyPrice") & "  -  " &
Record.item("TradeDate"))
        Next

    End Sub

    Sub FetchUsersInTeam(ByVal TeamID As Integer)

        Dim ConnectionDb As New OleDbConnection(AccessDatabaseConnection)
        If ConnectionDb.State = ConnectionState.Closed Then ConnectionDb.Open()

        Dim cmd As OleDbCommand
        Dim SQLReply As OleDbDataReader
```

```
        cmd = ConnectionDb.CreateCommand
        cmd.CommandText = "SELECT tblUserInfo.Username FROM tblUserInfo, tblTeamUsers,
tblTeams WHERE tblTeams.TeamID=" & TeamID & " AND tblTeamUsers.TeamID = tblTeams.TeamID
AND tblTeamUsers.AccountID = tblUserInfo.AccountID"

        SQLReply = cmd.ExecuteReader

        For Each Record In SQLReply
            TeamDetailsListBox.Items.Add(Record.item("Username"))
        Next

    End Sub

    Function FetchMemberInfo(ByVal UserID As Integer)
        Dim MemberInfoString As String = ""

        Dim ConnectionDb As New OleDbConnection(AccessDatabaseConnection)
        If ConnectionDb.State = ConnectionState.Closed Then ConnectionDb.Open()

        Dim cmd As OleDbCommand = ConnectionDb.CreateCommand

        cmd.CommandText = "SELECT Username, Passwrd, AccountID FROM tblUserInfo WHERE
AccountID=" & UserID & ""


        Dim SQLReply As OleDbDataReader = cmd.ExecuteReader

        For Each Record In SQLReply
            MemberInfoString += "Username: " & Record.Item("Username") & "       "
            MemberInfoString += "Password: " & Record.item("Passwrd")
        Next


        Return MemberInfoString
    End Function
    Private Sub TeamIdCheckedListBox_SelectedIndexChanged(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles TeamIdCheckedListBox.SelectedIndexChanged

        FetchTeamInfo()
    End Sub

    Private Sub TeamInfoCheckedListbox_SelectedIndexChanged(sender As System.Object, e As
System.EventArgs) Handles TeamInfoCheckedListbox.SelectedIndexChanged

    End Sub
End Class
```

LoginForm

```
Imports System.Data.OleDb

Public Class LoginForm

    Dim AccessDatabaseConnection As String = MainForm.AccessDatabaseConnection

    Public AccountID As Integer
    Public Admin As Boolean
    Public TeamName As String
    Public TeamMode As Boolean = False
    Public Username As String

    Private Sub OK_Click(sender As System.Object, e As System.EventArgs) Handles OK.Click

        Username = UsernameTextBox.Text

        If ValidUserLogin(Username, PasswordTextBox.Text) Then
            LoadUserInfo(AccountID)

            If Admin = True Then

                AdminView.Show()
                Me.Close()

            Else
                If TeamModeCheckBox.Checked Then


                    If TeamName = "" Then
                        MsgBox("You don't have a team! You will be loaded into single
user mode.")
                    Else
                        TeamMode = True
                    End If
                End If

                MainForm.Show()
                Me.Close()
            End If
        Else
            MsgBox("Invalid Username or Password.")
        End If
    End Sub

    Function ValidUserLogin(ByVal Username As String, ByVal Password As String)


        Dim ConnectionDb As New OleDbConnection(AccessDatabaseConnection)
        If ConnectionDb.State = ConnectionState.Closed Then ConnectionDb.Open()

        Dim cmd As OleDbCommand = ConnectionDb.CreateCommand
        cmd.CommandText = "SELECT AccountID FROM tblUserInfo WHERE Username='" & Username
& "' AND Passwrd='" & Password & "'"

        Dim SQLReply As OleDbDataReader = cmd.ExecuteReader
```

```
        For Each Record In SQLReply
            AccountID = Record.item("AccountID")
            Return True
        Next

        ConnectionDb.Close()

        Return False

    End Function

    Sub LoadUserInfo(ByVal AccountID As Integer)

        Dim UserValid As Boolean = False
        TeamName = ""

        Dim ConnectionDb As New OleDbConnection(AccessDatabaseConnection)
        If ConnectionDb.State = ConnectionState.Closed Then ConnectionDb.Open()

        Dim cmd As OleDbCommand = ConnectionDb.CreateCommand


        cmd.CommandText = "SELECT tblTeams.TeamName FROM tblTeams, tblTeamUsers,
tblUserInfo WHERE tblTeams.TeamID = tblTeamUsers.TeamID AND tblTeamUsers.AccountID =
tblUserInfo.AccountID AND tblUserInfo.AccountID=" & AccountID & ""

        Dim SQLReply As OleDbDataReader = cmd.ExecuteReader

        For Each Record In SQLReply

            TeamName = Record.item("TeamName")

        Next
        SQLReply.Close()

        cmd.CommandText = "SELECT Admin FROM tblUserInfo WHERE AccountID=" & AccountID &
""


        Dim SQLReply1 As OleDbDataReader = cmd.ExecuteReader

        For Each Record In SQLReply1
            Admin = Record.item("Admin")

        Next

        ConnectionDb.Close()
    End Sub


    Private Sub Cancel_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles Cancel.Click
        Me.Close()
    End Sub

    Private Sub Timer1_Tick(ByVal sender As System.Object, ByVal e As System.EventArgs)
        Me.Close()
    End Sub

    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles Button1.Click
        SignUpForm.Show()
```

```
    End Sub

End Class
```

Bibliography

[i] https://www.lseg.com/markets-products-and-services/our-markets/london-stock-exchange/derivatives-markets/equity-index-derivatives/ftse-100

[ii] auth. Biello David // Scientific American. - https://www.scientificamerican.com/article/can-math-beat-financial-markets/.

[iii] https://www.investopedia.com/dictionary/.

[iv] https://www.online-sciences.com/programming/visual-basics-programming-language-advantages-and-disadvantages/

[v] **Automatic database normalization and primary key generation** [Journal] / auth. Bahmani Amir Hassan, Naghibzadeh Mahmoud and Bahmani Behnam

[vi] https://msdn.microsoft.com/en-us/library/system.xml.xmldocument(v=vs.110).aspx

[vii] https://docs.microsoft.com/en-us/dotnet/framework/data/adonet/populating-a-dataset-from-a-dataadapter