Plymouth University

School of Computing, Electronics and Mathematics

PRCO304

Final Stage Computing Project

2017/2018

BSc (Hons) Computer Science

Joseph Kellaway

10503639

Cryptocurrency Value Predictor

# Acknowledgements

*This has been deliberately left blank until I know how many words I have left for it.*

# Abstract

This report describes a software development project aimed at developing an application that can predict the value cryptocurrencies in the near future in order to exchange to the currency that will grow the most in order to make a profit.

The report begins with a summary of cryptocurrency and blockchain technologies to give some education to the reader before declaring project goals. Research into competitor applications is presented, a solution outlined, and requirements are established.

The development process is outlined as an incremental process focused a single cycle of data through the application, then focused on customer facing functionality once the data collection and calculation cycle was completed.

A post-mortem of the project is presented, discussing how the project could be improved upon, followed by a conclusion of how the project faired overall.

Appendices include a user guide, project management artefacts, designs deemed too cumbersome to fit into the report, test results, and user testing materials.

# Table of Contents

Contents

[Acknowledgements 2](#_Toc512775822)

[Abstract 2](#_Toc512775823)

[Table of Contents 2](#_Toc512775824)

[1 Background, Objectives, & Deliverables 5](#_Toc512775825)

[1.1 Concerning Cryptocurrencies 5](#_Toc512775826)

[1.2 Concerning Blockchain 5](#_Toc512775827)

[1.3 Introduction 6](#_Toc512775828)

[1.3.1 Statement 6](#_Toc512775829)

[1.3.2 Motivation 6](#_Toc512775830)

[1.3.3 Goals 6](#_Toc512775831)

[1.4 Overview of Solution 6](#_Toc512775832)

[1.5 Objectives 7](#_Toc512775833)

[1.6 Requirements 7](#_Toc512775834)

[1.7 Initial Scope 8](#_Toc512775835)

[1.8 Deliverable Parts 8](#_Toc512775836)

[1.8.1 Desktop Application 8](#_Toc512775837)

[1.8.2 Storage & API 9](#_Toc512775838)

[2 Method of Approach 9](#_Toc512775839)

[2.1 Project Management 9](#_Toc512775840)

[2.2 Tools 10](#_Toc512775841)

[2.2.1 Desktop Application 10](#_Toc512775842)

[2.2.2 Storage & API 10](#_Toc512775843)

[2.3 Version Control 10](#_Toc512775844)

[2.4 Paperwork 10](#_Toc512775845)

[3 Development Stages 10](#_Toc512775846)

[3.1 Stage 0 – Initial Design 11](#_Toc512775847)

[3.2 Stage 1 – Data Harvester 12](#_Toc512775848)

[3.3 Stage 2 – Data Storage 13](#_Toc512775849)

[3.4 Stage 3 – API & Integration 14](#_Toc512775850)

[3.5 Stage 4 – GOFAI Algorithms 15](#_Toc512775851)

[3.6 Stage 5 – GUI 16](#_Toc512775852)

[3.7 Stage 6 – Benchmarking 21](#_Toc512775853)

[3.8 Stage 7 – User Trading 22](#_Toc512775854)

[4 Evaluation & Testing 23](#_Toc512775855)

[4.1 MoSCoW 23](#_Toc512775856)

[4.2 JUnit Testing 23](#_Toc512775857)

[4.3 User Testing 24](#_Toc512775858)

[5 Legal, Social, Ethical, & Professional Issues 24](#_Toc512775859)

[5.1 Ethical 24](#_Toc512775860)

[5.2 Security 25](#_Toc512775861)

[6 End-Project Report 26](#_Toc512775862)

[6.1 MoSCoW Review 26](#_Toc512775863)

[6.2 Project Objectives Review 26](#_Toc512775864)

[6.3 Changes 27](#_Toc512775865)

[6.3.1 Changes from the Proposal 27](#_Toc512775866)

[6.3.2 Changes from the PID 27](#_Toc512775867)

[7 Project Post-Mortem 28](#_Toc512775868)

[8 Conclusions 29](#_Toc512775869)

[9 References 29](#_Toc512775870)

[10 Appendices 30](#_Toc512775871)

[10.1 Appendix 1 – User Guide 30](#_Toc512775872)

[10.2 Appendix 2 – Project Management Artefacts 30](#_Toc512775873)

[10.2.1 Proposal 30](#_Toc512775874)

[10.2.2 PID 32](#_Toc512775875)

[10.2.3 Highlight 1 36](#_Toc512775876)

[10.2.4 Highlight 2 37](#_Toc512775877)

[10.2.5 Highlight 3 38](#_Toc512775878)

[10.2.6 Highlight 4 38](#_Toc512775879)

[10.2.7 Highlight 5 39](#_Toc512775880)

[10.2.8 Highlight 6 40](#_Toc512775881)

[10.2.9 Highlight 7 40](#_Toc512775882)

[10.3 Appendix 3 – Designs 41](#_Toc512775883)

[10.3.1 System Architecture Diagram 41](#_Toc512775884)

[10.3.2 UML Diagram 41](#_Toc512775885)

[10.3.3 Technology Review 41](#_Toc512775886)

[10.3.4 Interface Sketches 41](#_Toc512775887)

[10.3.5 Entity Relationship Diagram 41](#_Toc512775888)

[10.3.6 Database Normalisation 41](#_Toc512775889)

[10.3.7 Oracle Create Statements 41](#_Toc512775890)

[10.4 Appendix 4 – Test Results 43](#_Toc512775891)

[10.5 Appendix 5 – User Testing 43](#_Toc512775892)

[10.5.1 Instructions 43](#_Toc512775893)

[10.5.2 Questionnaire 43](#_Toc512775894)

Word count:

# 1 Background, Objectives, & Deliverables

## 1.1 Concerning Cryptocurrencies

Cryptocurrencies are “an electronic payment system based on cryptographic proof instead of trust” (Nakamoto, 2008), the first of which was Bitcoin which is the system the original dissertation was based on. There have been many successful and unsuccessful imitations (Coinmarketcap.com, 2013), some of which are near copies and some of which have new technologies that the developers believe is better.

Cryptocurrencies can be purchased from other people directly, like changing denominations of cash in person or exchanging your currency for a foreign one. However, exchanges have been set up which allows the safe organisation of trades.

Exchanges allocate an address to hold your cryptocurrency once you have purchased it, however, there have been several high-profile hackings of cryptocurrency exchanges. The guidelines given by most cryptocurrency development teams is to purchase a multi-signature wallet (Rosic, 2018). Wallets use public keys to receive cryptocurrency from others (allowing you to safely take payment from a stranger) and private keys to send cryptocurrency to others, provided you know their public key.

Exchanges regularly encourage investors to treat their safety with paramount importance, both of their investments when they have purchase cryptocurrencies, but also of their lives by not investing money that they can’t afford to lose – acknowledging that losses can be made also (White, 2017).

## 1.2 Concerning Blockchain

Blockchain was designed to be a public transaction ledger that is completely decentralised that would replace the banking sector for cryptocurrencies. The idea is that each block is made up of a list of transactions that are proved valid with cryptography. For a block to be confirmed, each machine verifying transactions must agree that it is correct. Once agreement has been made then the new block is placed and becomes immutable, making it impossible for a corrupt individual to “cheat” the system (Swan, 2015).

The idea behind blockchain is that all transactions will be transparent, and anyone is able to view transactions occurring freely. Many companies are finding other uses for blockchain as a technology, however, many uses require modification since this level of transparency isn’t useful for many fields including medical records (Azaria et al., 2016).

## 1.3 Introduction

### 1.3.1 Statement

Expert opinion on the future of cryptocurrencies is divided, with many economic experts conveying bleak outlooks, epitomised by American investment firm Elliott Management describing them as “one of the most brilliant scams in history” (Investopedia, 2018). This is vastly contrasted by many technological experts’ excitement at their potential to be the future of currency (Dorsey, 2018).

Whilst neither of the above may be entirely true, cryptocurrencies and blockchain are an exciting technology that have been heavily invested in by people all over the globe (Coinmarketcap.com, 2013). With investment comes growth and opportunity for profit, regardless of the reality of where cryptocurrencies will be long term.

### 1.3.2 Motivation

Selling a product or service for more than it cost you to acquire is one of the foundations of business. Being presented with a market that has several items with hugely fluctuating values is an opportunity to regularly purchase and sell those items, ideally for a profit. This is where cryptocurrency exchanges can be utilised to find opportunity to buy and sell as necessary.

The problem with leaving the matter there is that humans, being animals, are unable to achieve perfection. They get tired and need to stop to rest, they get hungry and need to stop to eat, they get emotional about both positive and negative events in their lives that can both have a negative effect on performance (Shockley et al., 2012).

### 1.3.3 Goals

To create a piece of software that removes humans from trading decisions to maximise profits.

By using pre-programmed computing logic, a piece of software that handles trading, it is possible to remove the emotional biases that a human may create towards currencies that have previously been used to make significant profit or loss.

## 1.4 Overview of Solution

An overview of the system that would be the solution to the discussed issues:

1. A system that collects data from the chosen cryptocurrency exchange. It will calculate the average price for each minute and the growth from the previous minute and store that information for future reference.
2. The system will use stored price information to determine the most profitable currency to obtain and take action if it is currently holding an alternative currency.
3. The system should a process that once started is automated so that it allows the user to protect invested capital from themselves.
4. An option to withdraw an amount of invested capital at target values to protect invested capital from system failure.
5. The system should be able to maintain efficiency over time, with the potential ability to retrain its understanding of the cryptocurrency market.

## 1.5 Objectives

To develop a system that allows for automated trading in a way that:

1. Removes the need for user interaction once the system has been initialised.
2. Improves trading efficiency by:
   1. Reacting to the market rather than allowing human biases to interfere.
   2. Operating continuously when the application is active – does not need to take breaks.
   3. Ensuring no human error causes miscalculations.
3. Allows the user to automatically cash out profits or the initial investment once a target has been reached to prevent loss.

## 1.6 Requirements

The above objectives have been expanded into deliverables and sorted into the following MoSCoW prioritisation. The reasoning behind this requirements breakdown is that the system then has the opportunity to have its success measured without having to consider prediction accuracy, which is liable to rise and fall routinely.

Must have:

* Ability to collect live data from GDAX API endpoint.
* Ability to make predictions of next value based on the data collected with “**G**ood **O**ld-**F**ashioned **AI**” predictions.
* Ability to collect historic data from GDAX API to decrease time required to collect enough data to make above predictions.
* Ability to store historic data so that minimal data collection is required on each runtime.

Should have:

* Ability to see gaps in held data and fill as follows;
  + Re-attempt to collect prices (in the event of a failed API call when storing data).
  + Fill with the average price of data either side of the gap.
* Ability to make predictions of next value based on the data collected with Neural Network predictions.
* Ability to automatically trade based on above predictions.
* Ability to display neural network accuracy;
  + And allow the user to manually retrain.

Could have:

* Ability to automatically retrain the neural network based on falling accuracy as training data becomes more historic.
* Ability to allow the user to input investment protection levels;
  + Ability to set single withdrawal of X when investment value reaches Y (i.e. withdraw initial investment when a large enough profit is made).
  + Ability to set regular withdrawals of X when investment value reaches Y (i.e. withdraw profit each time it reaches a large enough value).
  + Ability to halt trading if certain conditions are met (i.e. a loss threshold is reached within a certain time period);
    - Halt converts investment to base currency (USD).
    - Ability to set base currency to another traded currency (i.e. the user’s “most trusted” cryptocurrency).

Would have:

* Ability to collect data from multiple exchange APIs and consider how accurate GDAX prices are.
* Ability to trade concurrently on multiple exchanges based on both likely growth and prices local to each exchange.
  + Due to the usual delay of currency arriving at exchange wallets it would be unlikely that the system would swap value between exchanges, a value would be deposited into each exchange and traded based on global prices rather than the exchange’s price.

## 1.7 Initial Scope

The initial scope for the project is for all parts in the MoSCoW “must” to be complete and with no complications all of the “should” category to be completed also. Implementing the neural network and trading live on the GDAX platform are the parts that have been deemed less critical to the project should any issues arise that delay development progression.

## 1.8 Deliverable Parts

The project can be split into two main deliverable parts:

### 1.8.1 Desktop Application

The nature of this project requires a device that is either mains powered or has a long battery life since it is likely to be left on for vast amounts of time and potentially large amounts of processing to collect historic data to start.

Java and C# were highlighted as potential languages to write the application in, with Java being preferred since it is cross-platform and using C#, although potentially faster, would prevent almost 11.3% of devices from using the application (Netmarketshare.com, 2018).

### 1.8.2 Storage & API

Since there is a clear one-to-many relationship between currencies and their prices storage in a relational database management system was a logical decision. The university has an Oracle server available to students, and so making use of that was also logical.

Creating an API for Oracle in C#/.NET using Entity Framework is a logical route to take due to the author’s previous experience doing so. It would also minimise time required on a non-core, yet essential part of the project.

# 2 Method of Approach

Since there were several easily distinguishable tasks within the project, software development was employed with an incremental approach with 7 increments, similarly to as discussed in the project initiation document:

1. Implement the data harvester that collects data and calculates prices.
2. Implement data storage.
3. Implement database API and integrate with harvester.
4. Implement GOFAI algorithms.
5. Create GUI.
6. Benchmark the GOFAI algorithms.
7. Implement user trading.

## 2.1 Project Management

The project was heavily managed using the principles of PRINCE2, specifically:

* The project was initially [proposed](#_10.2.1_Proposal) via SPMS.
* A [project initiation document](#_10.2.2_PID) was created detailing initial outlines and plans for the project.
* The project was separated into the [stages](#_2_Method_of).
* Weekly [highlights](#_10.2.3_Highlight_1) were written, and review meetings were organised to ensure that development was ongoing, even if circumstances had put it off schedule.
* A risk management plan was put in place as part of the [PID](#_10.2.2_PID).
* Communication was discussed with the project supervisor and it was agreed that emails could be exchanged at any time.

## 2.2 Tools

The following section is a breakdown of how the project was developed and what tools were used to implement each part.

### 2.2.1 Desktop Application

The majority of the functionality of the project comes from the Java application, which was written using NetBeans IDE version 8.2 and a constantly up-to-date version of Java.

Maven version 3.5.2 was used to support development of the project by managing dependencies, this included the GUI’s dependency on the library and the library’s dependency on both JUnit and Google’s Gson.

Gson version 2.8.2 was used for JSON parsing (both to and from JSON when collecting/transmitting data), with the exception of parsing data pulled from the GDAX API endpoint.

### 2.2.2 Storage & API

SQL Developer version 4.1.5.21 was used to create and modify the Oracle database, see Appendix 3.7 for table create statements.

Visual Studio 2015 version 14.0.25431.01 Update 3 was used in combination with Entity Framework version 6.1.3 to implement the API endpoints that expose the Oracle database to the Java application.

## 2.3 Version Control

GitHub was used as version control for the project to ensure that development could be reverted to previous implementations if the need should arise. Since it was an individual project, the need to branch and merge wasn’t as prevalent as it could have been in a group project and the main benefit of using version control was to maintain familiarity with the terminal commands and having an additional backup for the project.

## 2.4 Paperwork

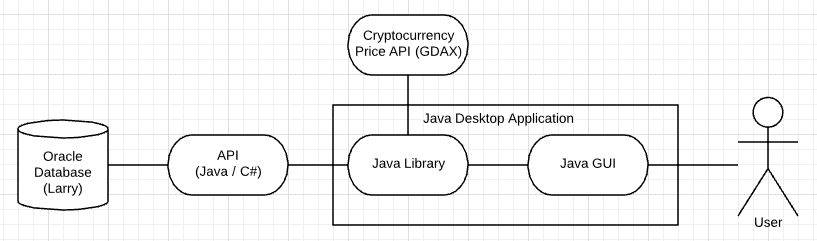
The report was written using Microsoft Office 365’s Word, with diagrams drawn using Lucidchart and sketches completed using pen and paper. Pen and paper was also used for planning logic prior to each development stage and for problem solving when issues arose.

# 3 Development Stages

Development of the project can be segmented into the following stages. This chapter is largely organised in chronological order of development, however, it was logical to separate tasks out modularly and as such some integration took place between stages as necessary.

## 3.1 Stage 0 – Initial Design

The first prerequisite to starting implementation was understanding how the system was going to work through an entire data cycle. The user would run a Java application that they would interact with in the form of a graphical user interface, this would utilise a library to handle the majority of processing required. The library would collect data from a cryptocurrency exchange’s API endpoint, calculate an average price for the minute, as well as growth from the previous minute and post it to storage that was exposed through an API implementation. When enough recent data is collected, the library would then be responsible for estimating the next value based on recent price changes.



\*\*

The next requirement would be to determine what data would be stored within the database. This meant briefly sketching the GUI. Due to the fact that the application is intended to be used with minimal monitoring displaying the predictions through the GUI would be illogical. Instead the user would be able to see ummarized current statistics for each currency and their trading data for when they are monitoring the program whilst it is trading. There would also need to be a display that allowed them to compare how the trading algorithms are currently performing against other algorithms, against purchasing a single cryptocurrency, and against purchasing no cryptocurrency at all.

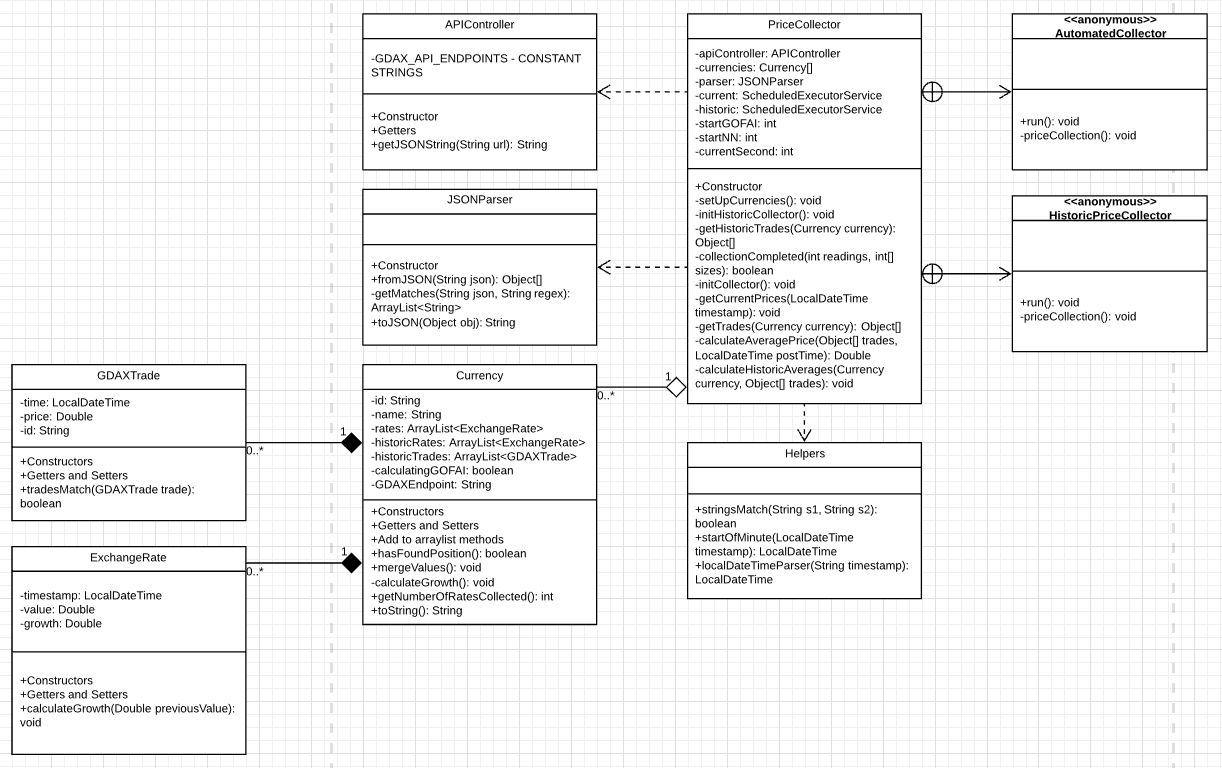
\*\*GUI Sketch

After viewing all of this information on a single screen, the user will be able to change tabs to a different page that allows them to input the specifics of the automated trading that they would like to start. By having a single screen for all benchmarking details and having a single screen for trading details it will only require the user to remember which algorithm they wish to use to trade when switching from one screen to the other.

With these designs in mind, it was possible to start the first stage of development – collecting the data from the chosen cryptocurrency exchange, GDAX.

## 3.2 Stage 1 – Data Harvester

The initial plan for the data harvester was to have a main PriceCollector class that when initialised would collect Currency information from the database (although, it had to be hard coded at this stage since the database did not yet exist) and then start two collection threads that used the ScheduledExecutorService to periodically submit get requests to GDAX’s API endpoint. The first collection thread would run at the start of each minute to collect a single up-to-date price from the GDAXTrades available. If this thread had not run, then historic data would be collected with a 1 second interval between each batch of get requests to adhere to GDAX’s requests per second limitation.



\*\*

An APIController would be created to handle communication with GDAX’s API and a JSONParser would parse the JSON strings into GDAXTrade objects. From here the PriceCollector itself would assess which GDAXTrades related to the minute an ExchangeRate was required for, calculate the average price and growth, and add the new rate to the relevant Currency.

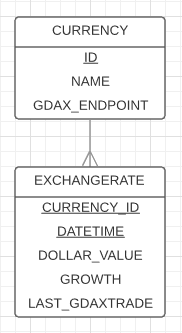
Throughout the project constant monitoring, testing, and improvement of previously implemented systems occurred to ensure that integration didn’t create issues with prior development. This resulted in the PriceCollector’s modification to using only a single ScheduledExecutorService (due to a concurrency issue that occasionally occurred), the APIController being changed to a GDAXAPIController that inherited shared functionality from a new APIController class, the JSONParser was allocated directly to this class and the Helpers class was separated out into a number of helper classes that had a specific functionality (i.e. performing mathematical calculations or modifying a LocalDateTime).

For historic collection batches of prices are collected with each get request (consisting of 100 GDAXTrades) and the PriceCollector averages minute data into ExchangeRates to add to a list of historic prices which is merged when collection is completed. GDAX issues an id number to each trade which is used to traverse pages of trade get requests. The oldest trade id relevant to the ExchangeRate is stored with it to enable the system to have a marker of where it last collected data in the event of the program being turned off, there being a power failure etc.

## 3.3 Stage 2 – Data Storage

Once data was being collected, brief considerations of GOFAI algorithms were made before it very quickly became obvious that storing collected data would be essential for any algorithm to be meaningfully assessed without wasting vast amounts of time.

Data to be stored was extracted from the [proposed GUI interface](#_3.1_Stage_0), then data required to access prices from the GDAX API endpoint was added and normalisation was performed to create a database with the following ERD.



\*\*

It is shown that each currency may have numerous exchange rates, but only one at a given time. The exchange rate’s compound key consists of the currency’s id as a foreign key and a timestamp and will return the value in United States dollars, growth as a percentage and the last GDAXTrade’s id.

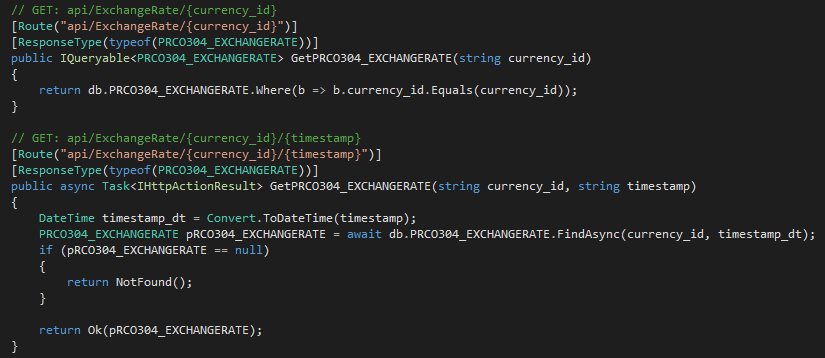
The storage medium used was [Oracle](#_1.8.2_Storage_&) and it was created using the [create statements](#_10.3.7_Oracle_Create).

## 3.4 Stage 3 – API & Integration

Now that data was able to be harvested and there was suitable storage to place it in, exposing the database through an API endpoint was necessary.

First, a connection to the Oracle database was made using an ODP.NET managed driver. Next an ASP.NET web project was made using the “Web API” template and Entity Framework was used to create ADO.NET Entity Data Models from the database for both database tables and then Web API 2 controllers with actions.

This method allowed quick progress due to the reasons [discussed](#_1.8.2_Storage_&), however, the default methods exposing the ExchangeRate table in database needed to be adapted because the method above does not accommodate compound keys. The result was creating new endpoints that were fed additional information in the pathway. A pathway that was fed the currency\_id was created to reduce sorting required by the Java application’s get method and then a pathway was created that took both the currency\_id and a timestamp to select a single ExchangeRate from the database. This second method was particularly useful because the Route was replicated for both put and delete methods, allowing the option to edit and remove data from the database, should the application require it at a later date.



\*\*

After the database had been exposed through its API endpoints, the Java application had to be modified to integrate the additional functionality. APIController was extended into two separate classes; CurrencyAPIController and ExchangeRateAPIController, which handled converting to and from JSON using Gson, with APIController already equipped to send all of the REST requests required if given the url and, if relevant, JSON data to send to the database.

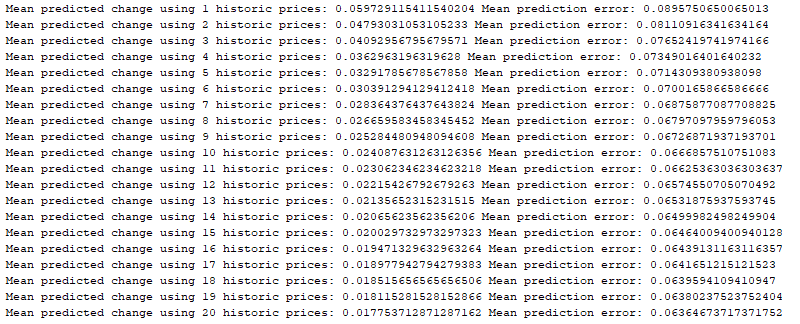
## 3.5 Stage 4 – GOFAI Algorithms

After data was being harvested and stored it would become possible to analyse GOFAI algorithms more efficiently and so the GOFAIPredictor class was made. Since the initial focus would be to find a successful algorithm, the class initially had one purpose – to take the collected price data for each Currency and then iterate through each price in chronological order and attempt to predict the next price based on the GOFAI algorithm it was given.

As described \*\*earlier, the GOFAI predictor would ultimately attempt to find which Currency had the most momentum at a given time. It would then purchase the Currency and sell when it predicted that another Currency would have more growth in the next minute.

Given the information that we have about each Currency, 20 different algorithms were created to predict the next growth values. The first would be that momentum would continue as it is currently (i.e. growthn+1 = growthn), the second would be the average of the current and previous growths (i.e. growthn+1 = (growthn + growthn-1) / 2) and so on.

Once these growths were calculated they were compared to the real growth that occurred. Since negative and positive error would cancel each other out the absolute value was used. A mean of these absolute values for each algorithm was calculated, which would show the average error of each algorithm and the lowest error would be the best algorithm that had been tested. However, when comparing these errors, it appeared that simply using more prices reduced the error. The average predicted change was calculated for each algorithm and placed alongside the error and it showed the following results:



\*\*

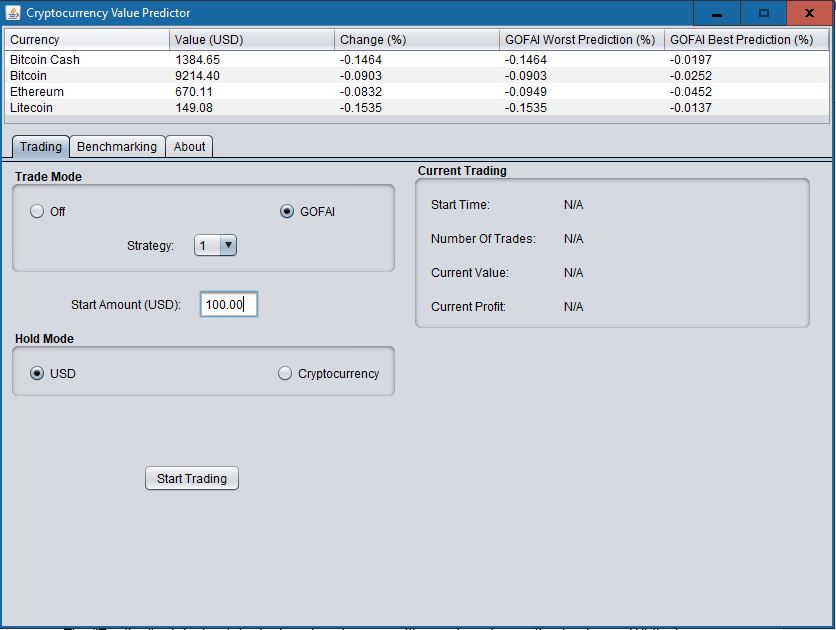
Using a larger number of prices meant less error but it also meant a much more conservative prediction, which wasn’t necessarily truly a better prediction algorithm in terms of making profit, it just happened to guess closer to the right value because it minimised mistakes. An alternative way of testing algorithm accuracy would be necessary – benchmarking them against each other in terms of profit or loss generated over the collected prices. To show these benchmarks effectively, it would be necessary to first progress on to implementing the GUI and display the benchmarking results on there.

Realising that the application had grown too large to run from the PriceCollector any longer, the CryptocurrencyValuePredictor wrapper class was implemented to tie the different functionality together in a more logical format. Implementing the wrapper class at this stage created benefit the project in a number of positive ways, including both not implementing it so early that it felt like an empty “shell” class that didn’t really do anything for a long period of time, and also not implementing it so late that there was a lot of functionality to move around between classes and create unnecessary integration work.

## 3.6 Stage 5 – GUI

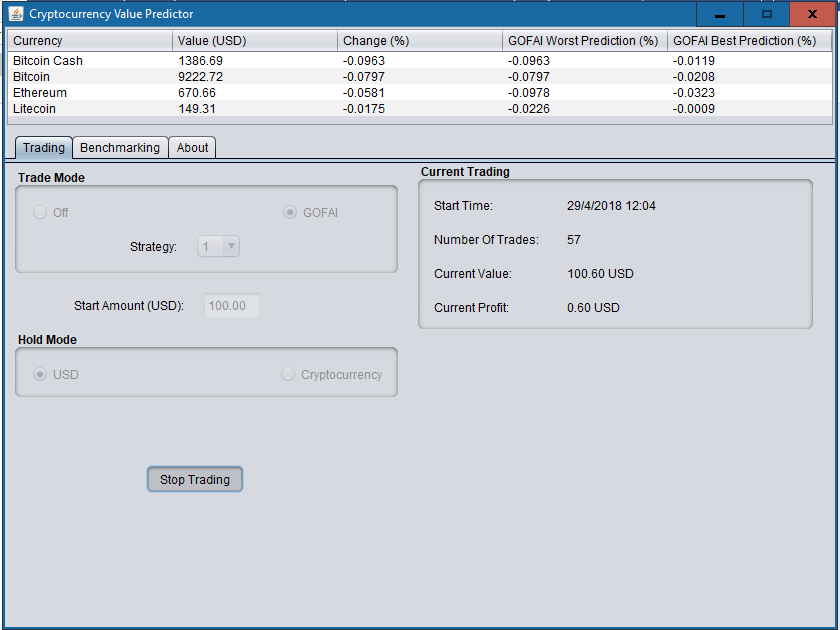
Implementing the GUI as a second Java application for the project meant that the main logic behind the system could be separated from it. This would mean that if the project were brought to market as a product then the amount of processing done on their local machine could be minimised which would reduce the demands on their machine, it would reduce their power usage and it would also revoke the opportunity for someone who purchased the product to reverse engineer the system to clone the product.

The GUI stayed largely as [designed](#_3.1_Stage_0) but with a few minor exceptions. It consists of a single JFrame split into a JScrollPane with a JTable of current cryptocurrency price data being displayed and a JTabbedPane which has 3 JPanels offering the views “Trading”, “Benchmarking” and “About”.



\*\*

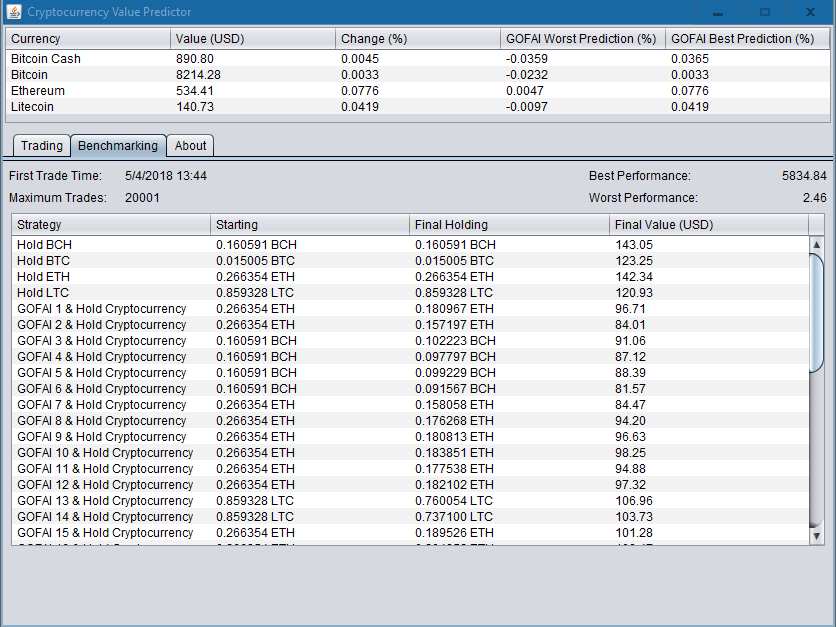
The “Trading” tab is the default view that the user will see when the application is run. Whilst in most circumstances the user will wish to change to the “Benchmarking” tab to inspect the market before initialising trading, [user testing](#_4.3_User_Testing) suggested it would be more appropriate to display the primary function of the application as the first view that the user gets. It also allows the user to very quickly set up trading in the event of a power failure (assuming they know how they wish to configure the Trader).



\*\*

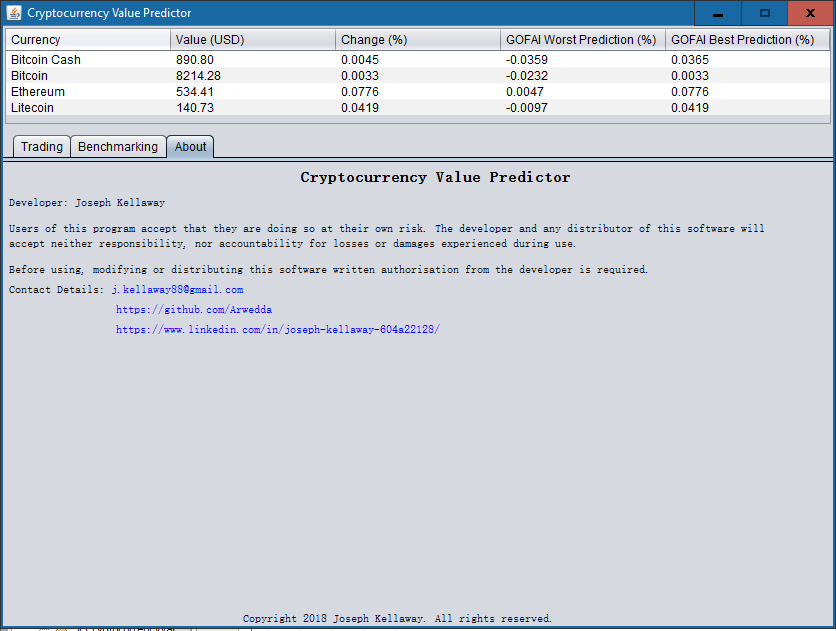
The “Trading” tab is split into two sections – configuration and current statistics. On the configuration side JRadioButtons were used to allow configuration with a small number of options available, a JComboBox was used for the strategy number as 20 was deemed too many and the starting trade value is entered via a JTextBox as [user testing](#_4.3_User_Testing) didn’t approve of the JSpinner method that was originally in place. The interface is robust in a way that prevents it from starting to trade with an unacceptable configuration of inputs and it is impossible to start trading before enough data has been collected to benchmark.

The other half of the “Trading” tab shows the statistics of the current trading session – the start time, the number of trades performed, the current value (and current value in USD if it is holding a cryptocurrency) and the profit.



\*\*

The “Benchmarking” tab shows very little until data collection is complete, then it shows the number of trades between the defined time and the current time, the best and worst performances possible if knowledge of future values were obtainable and a breakdown of how each trading strategy is performing in the current market in a JTable.



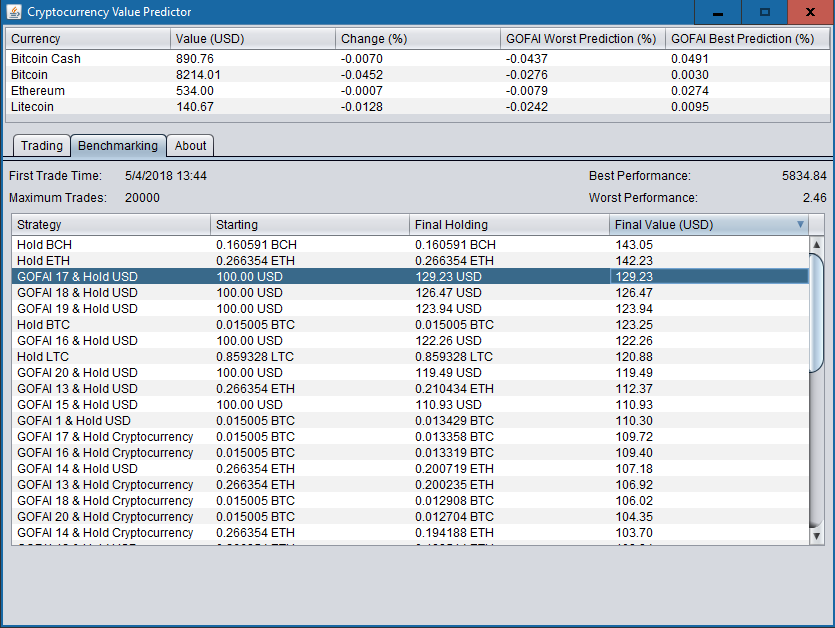
\*\*

The “About” tab shows developer contact details, as well as an accountability disclaimer and a brief copyright message. Clicking the contact details uses the system default email application and web browser to open an email to or navigate to the websites as appropriate.



\*\*

Each of the JTables used has been configured so that clicking on the content selects a whole single row only. This is to allow the user to clearly see the details of a single item. Clicking a header of a column sorts the column in descending order. Descending order was chosen due to the nature of the data displayed – the user would likely want to quickly single out the best performing algorithm and select it to start trading. Obviously, the standard way for a computer system to display strings of numbers that have been ordered would be unacceptable (see Figure \*\*), so an algorithm was written to perform a more natural ordering of the numbers. User testing did not notice an issue with descending order when using this feature.



\*\*

## 3.7 Stage 6 – Benchmarking

The prediction algorithm that is the most accurate would not necessarily be the most profitable since matching positive predictions with positive growth and negative ones with negative growth makes profit and saves losses; predicting accurately but with the opposite sign would result in an incorrect purchase or sale decision.

This meant developing the trading functionality that would in turn mostly be the same that is used by the user trading. The Trader class makes use of two enumeration classes and an integer to hold the configuration settings selected for it, as well as a Wallet class that was implemented simply to hold the current currency held and value of the user’s investment. The Wallet class is also responsible for holding initial trading values so that profit can be calculated.

The main benchmarking algorithm’s purpose is to loop over every minute of trading collected that is within the relevant time period and perform trades from the beginning until the end as if it were live data and does not have any more information than it would have under normal circumstances. It compares the selected algorithm’s growth predictions for each Currency and converts its Wallet’s Currency to the one with the highest value. If all of the predictions suggest loss rather than growth, the Trader will either convert to USD or hold the current cryptocurrency, depending on how it is configured.

The best and worst performance traders are given the knowledge of future trades so that they may indicate whether currency values have fluctuated much in this time period – a worst value much lower than the starting value indicates lots of periods of negative growth and a best value that is much higher than the starting value indicates lots of periods of positive growth.

Once benchmarking is complete the values are displayed on the GUI for the user to compare and select a trading method. At the time of writing the most successful trading practice would be to simply have been to purchase Bitcoin Cash (BCH) and hold it, though GOFAI algorithms 16-19 with a hold mode of USD outperform some individual cryptocurrency growth (\*\*see screenshot).

As the \*\*screenshot indicates, investigation into the performance of algorithms proved it is impossible to single out a single algorithm as the one with the best performance all of the time. Some work very well over short periods of time and others better over longer periods of time. The cryptocurrency market experiences uptrends and downtrends much like the stock market \*\* which also influences which is the most effective algorithm at the time.

It was due to this that the decision was made to leave multiple algorithms in the application for the user to select when trading. This also distances the developer from liability, as the user will find it more difficult to claim that they believed the application couldn’t ever make a loss.

## 3.8 Stage 7 – User Trading

Adding user trading to the application was relatively straight forward once the benchmarking had already been completed. A blank Trader was set up which is configured by the user’s selections on the GUI and it trades similarly to the benchmarking traders, with modification to only trade based on the most recently collected ExchangeRate and its predictions. This single trade functionality was extended to the benchmarking process to allow the benchmark to remain up to date.

# 4 Evaluation & Testing

Early in the project it was decided that since profitability would be ever-changing, MoSCoW would be a much better way to evaluate the project.

## 4.1 MoSCoW

The project meets the following criteria as outlined by the [requirements](#_1.6_Requirements) section of this document:

* Ability to collect live data from GDAX API endpoint.
* Ability to make predictions of next value based on the data collected with “**G**ood **O**ld-**F**ashioned **AI**” predictions.
* Ability to collect historic data from GDAX API to decrease time required to collect enough data to make above predictions.
* Ability to store historic data so that minimal data collection is required on each runtime.
* Ability to see gaps in held data and fill as follows;
  + Re-attempt to collect prices (in the event of a failed API call when storing data).
  + Fill with the average price of data either side of the gap.
* Ability to make predictions of next value based on the data collected with Neural Network predictions.
* Ability to display neural network accuracy;
  + And allow the user to manually retrain.

The following criteria from the should section of the [MoSCoW](#_1.6_Requirements) requirements has been partially met:

* Ability to automatically trade based on above predictions.

The project at present handles automatic trading, but in a sandbox environment so that no money can be lost whilst showcasing the project, this would obviously make it an exaggeration to suggest that automated trading occurs since no trades are posted to the GDAX API.

## 4.2 JUnit Testing

JUnit was used to test each public method within the project to ensure that functions performed as anticipated in the scenarios that it would face at runtime. Private methods cannot directly be tested with JUnit testing, but by testing the public methods that call them it was possible to ensure that they are behaving correctly. Furthermore, since the majority of the logic is within a self-contained library that could be used by another developer or application in the future, it has been rigorously tested in an attempt to ensure that each function can handle any value, including null, in an elegant way.

After the unit testing had been completed Javadoc was written for every class in the library. This was reserved until later in the project due to the fact that there was a need to integrate parts of the project with new functionality. This would have caused the need to change the Javadoc that had previously been considered finished, and therefore could potentially have been erroneously missed.

## 4.3 User Testing

User testing was carried out to ensure the robustness of the application as well as highlight any issues with the GUI that were unnoticed by the developer. Specific [instructions](#_10.5.1_Instructions) and [questions](#_10.5.2_Questionnaire) were planned out and asked to each tester to ensure that bias was not created.

The main issue discovered by user testing was the display of numerical data. Some clear rounding rules were required to prevent circumstances that caused scientific notation to be used.

A similar issue to the rounding that was uncovered was the unclear use of the dollar symbol ($). Most users preferred the use of the currency abbreviation “USD” to denote United States dollars since there are over 20 countries that use or have used “dollars”, many of which with the same symbol.

The original wording on the button that initiates trading prior to the system having enough data was also changed due to it not giving a clear message to the end user. Some users felt that “Needs Data” was misleading and denoted error, and so it was changed to “Collecting Data…”.

Users also disliked the practice of using a JSpinner for dollars and a separate one for cents when entering the starting value of a trading session. A textbox was preferred and so the GUI was changed, which led to the implementation of a helper class that only allowed the user to enter numerical values.

# 5 Legal, Social, Ethical, & Professional Issues

## 5.1 Ethical

The application’s main aim – to produce a profit from trading with other users – does create several ethical dilemmas, some of which have already been solved.

The first question would be whether it is ethical to use a program to profit from other users of the exchange? Well, that answer has been given by both society deeming trading bots to be legal and GDAX allowing trading bots to operate on their platform. Exchanges such as GDAX also regularly insist on responsible investment from their users (White, 2017), arguably suggesting that the only investor who the application is ethically responsible for is its user.

The second question would be whether the developer has any responsibility for the loss of monetary value experienced by the user. The application follows the example set by exchanges by telling the user to invest responsibly, but it was able to go further still. Rather than selecting a single algorithm that was profitable at the time of development, the benchmarking process is open for the user to inspect what algorithms are performing well when they wish to trade, placing the onus on the user rather application or developer.

User testing was carried out under the conditions specified in the pre-approved ethics application to ensure that no ethical issues were caused by user testing. This means that there were less than 20 testers, they all completed testing on Plymouth University property, and they were all students or staff that were over 18. To further ensure that user testing was carried out ethically, results were collected anonymously, which would also negate the need for consent forms.

## 5.2 Security

Storage of personal information is ever becoming a more important topic as more companies report breaches, and more worrying, report historic breaches that they failed to report in a timely manner. This led to a variety of decisions with regards to data storage:

Do not store any data that isn’t necessary – whilst it may be an inconvenience to the user to be forced to copy and paste a GDAX API key into the end program each time they run the program, it isn’t actually essential for the implemented system to hold user accounts. GDAX already holds that data and to send a buy or sell request only requires the API key. This meant that when live trading is implemented, the system would have a JPasswordField added to the GUI and then when the user presses “Start Trading” it would be encrypted and stored in a variable, cleared from the GUI and decrypted and re-encrypted each time it is used. This would remove the possibility of a hacker accessing a database that would grant them access to a collection of GDAX accounts.



\*\*reference 2018 Data Protection Bill

Security of the calculated trade data is important, but less so than personal data. Obviously the database is protected by the university’s standard username format and a complex password, but it is further protected by prefixing the tables with “PRCO304\_” to minimise the possibility of a hacker guessing table names. This data has not been encrypted since it would increase the time required to collect the necessary data to begin trading, but encrypting the data would be logical if the application were to be developed further and eventually brought to the point where it is a service that can be sold to users.

# 6 End-Project Report

Overall the project went well, there were a variety of complications during development that slowed progress, but no issues that prevented development completely. Benchmarking and sandbox trading are both easy to understand tools that the user can utilise to understand the current cryptocurrency market and develop a trading strategy.

## 6.1 MoSCoW Review

All “must” and most “should” targets have been achieved, so although the project didn’t fulfill the entirety of the ambitious workload that was initially set, it can be considered a success.

## 6.2 Project Objectives Review

The project objects were to develop a system that allows for automated trading in a way that:

1. Removes the need for user interaction once the system has been initialised.
2. Improves trading efficiency by:
   1. Reacting to the market rather than allowing human biases to interfere.
   2. Operating continuously when the application is active – does not need to take breaks.
   3. Ensuring no human error causes miscalculations.
3. Allows the user to automatically cash out profits or the initial investment once a target has been reached to prevent loss.

In its current state, the project could not fully be considered the application it was designed to be because it has no functionality that allows the user to set automated cashing out, however, the project does fulfill the first two objectives:

1. With the exception of halting trading, the user does not need to interact with the application once trading has been initialised.
   1. The user is given market information that allows them to select a trading strategy that is carried out. This strategy reacts directly to the market and not to any biases.
   2. The application does not need to halt once it has collected the initial set of data that it requires.
   3. The application performs calculations for the user so that errors are minimised.

## 6.3 Changes

Throughout the project, a variety of changes have been made for a variety of reasons. From the initial proposal it was clear that changes would be likely because the project relied on working with new technologies, the most notable being the GDAX API endpoint.

### 6.3.1 Changes from the Proposal

The project proposal was written based on an idea rather than a business or user’s set of needs. The proposal was a speculative look at what could be done with a variety of technologies that could give an indication of how the value of a particular cryptocurrency could change in the imminent future. There were a large variety of ideas that were contemplated and refined to produce the PID and so although changes were obviously made, they weren’t so developed that removing or replacing them could be deemed unexpected.

### 6.3.2 Changes from the PID

There have been a variety of changes to the project since the project initiation document. These changes have mainly occurred as a result of time constraints imposed on the project:

The first notable change from the PID (although the PID doesn’t explicitly mention the intention of it) is the lack of live trading in the current application. Sandbox trading occurs to both benchmark the trading algorithms and give the user the impression of how live trading would occur within a more finished product.

Without the implementation of live trading to the GDAX platform, investment protection was less critical than other functionality within the application. As per the [MoSCoW analysis](#_1.6_Requirements) of the project, it was a indisputably important part of a finished product due to the expectation that profits and losses would be made, but it was less essential in a system that would not make real trades.

The implementation of a neural network was desirable from the offset – it would theoretically add more accuracy to the GOFAI algorithm by applying a weight to each value used to predict the next growth based on the link between each value and the actual growth in recent cases. But even with that in mind, it was still essential to ensure that the application was capable of reliably performing benchmarked GOFAI predictions before design of a neural network had even begun.

# 7 Project Post-Mortem

Obviously if the project were to be repeated, time allocated to researching technologies that were used would be able to be assigned elsewhere. This would likely be either to alternative technologies that would potentially improve the project in its current state, or to implementation time itself to advance the state that the project is currently at.

Whilst it was auspicious to begin the report with nearly 2 months of development time left, starting earlier would have further improved the synergy between developing and documenting. Development was fresh when documentation began which improved efficiency greatly, however, if detailing development was an ongoing process throughout development it would likely have further improved the project.

Two other changes that would be made if the project was repeated would be to switch from NetBeans to IntelliJ IDEA by JetBrains as an IDE to develop the Java application in and switch from Maven to an alternative build automation tool such as Gradle. The reasoning behind these changes are not necessarily to improve the product, but to improve myself as a developer by learning additional new skills.

Ideally if the project were to be repeated then the GOFAI prediction algorithms would be improved upon by creating a mirroring set of algorithms that are trained using machine learning. These algorithms wouldn’t simply create a mean of the prices to be considered as their prediction, but to apply weights to each of the prices so that minutes calculated to be less relevant to the prediction can be scaled down and those more so scaled up.

# 8 Conclusions

The project was founded on creating an application that would be able to profit from the cryptocurrency market. In reality, it has proven impossible to achieve this goal for every single minute of trading and it is likely to always be the case. However, it is possible to create a profit overall, depending on how performance is being benchmarked and which algorithm is selected. Presenting benchmark results and giving algorithm selection to the user is a way of considering the project to be success, even if only certain algorithms are successful at any one given time.

It would be fair to say that only the core elements of the project have been achieved. However, when considering the time limitations of the project and the constraints that they imposed, and lost development time due to complications faced during development, it would also be fair to say that a lot has been achieved during the time frame. And it would certainly be fair to state that the project in its current state could be the basis for a more thorough implementation of the cryptocurrency trader that was planned. It would also be fair to say that even with an optimal neural network implementation of the prediction algorithm by todays standards, it would be likely that future studies will discover an even more efficient form of artificial intelligence that would surpass it.

Despite feeling quite well prepared to initialise this project, I feel that a lot of skills have been learned whilst carrying out this project. These skills include interacting with a 3rd party API, problem solving unforeseen setbacks and unguided research to problem solve.

So overall, I feel that the project can be described as successful. Perhaps not as successful as the overly ambitious scope of the MoSCoW plan set out, but some very useful functionality has been implemented and an exciting foundation for a future project has been created. Furthermore, the project as a whole has been an enjoyable learning experience.

# 9 References

Azaria, A., Ekblaw, A., Vieira, T. and Lippman, A. (2016). MedRec: Using Blockchain for Medical Data Access and Permission Management. *2016 2nd International Conference on Open and Big Data (OBD)*.

Coinmarketcap.com. (2018). *All Cryptocurrencies | CoinMarketCap*. [online] Available at: <https://coinmarketcap.com/all/views/all/> [Accessed 19 Apr. 2018].

Dorsey, J. (2018). *Bitcoin will rule world by 2028, says Twitter CEO Jack Dorsey*. [online] CNET. Available at: <https://www.cnet.com/news/twitter-ceo-says-bitcoin-will-be-the-worlds-single-currency-in-10-years/> [Accessed 20 Apr. 2018].

Investopedia. (2018). *Elliott Management Calls Cryptocurrencies a Scam, Bubble and Fraud*. [online] Available at: <https://www.investopedia.com/news/hedge-fund-elliott-management-calls-cryptocurrencies-one-most-brilliant-scams-history/> [Accessed 20 Apr. 2018].

Nakamoto, S. (2008). *Bitcoin*. [online] Bitcoin.org. Available at: <https://bitcoin.org/bitcoin.pdf> [Accessed 19 Apr. 2018].

Netmarketshare.com. (2018). *Operating system market share*. [online] Available at: <https://goo.gl/sBmZVR> [Accessed 20 Apr. 2018].

Rosic, A. (2018). *5 High Profile Cryptocurrency Hacks - Blockgeeks*. [online] Blockgeeks. Available at: <https://blockgeeks.com/guides/cryptocurrency-hacks/> [Accessed 27 Apr. 2018].

Shockley, K., Ispas, D., Rossi, M. and Levine, E. (2012). A Meta-Analytic Investigation of the Relationship Between State Affect, Discrete Emotions, and Job Performance. *Human Performance*, 25(5), pp.377-411.

Swan, M. (2015). *Blockchain*. 1st ed. Sebastopol, CA: O'Reilly.

White, A. (2017). *Please invest responsibly — an important message from the GDAX team*. [online] The GDAX Blog. Available at: <https://blog.gdax.com/please-invest-responsibly-an-important-message-from-the-gdax-team-c32f68b13e3a> [Accessed 23 Apr. 2018].

# 10 Appendices

## 10.1 Appendix 1 – User Guide

## 10.2 Appendix 2 – Project Management Artefacts

### 10.2.1 Proposal

|  |  |
| --- | --- |
| Project title | Cryptocurrency Value Prediction Tool |
| Project keywords | Software development, database development, data analysis |
| Project description | I want to make a system that pulls an average trade value from the GDAX API and posts it to my database, analyses the price change against prior collected data and makes a prediction based on previous data held within the database.  To be used by:  Cryptocurrency stock holders (Bitcoin, potentially Ethereum).  Why do they want it:  Efficiency - program able to perform analysis faster than humans.  Accuracy - program able to ensure zero mistakes in calculations (not necessarily that the recommendation is correct, just that the mathematics is).  Impartiality - program able to ensure no emotional decision factored into recommendation.  To produce:  A desktop system that pulls data from GDAX API to create an average price for a time period, stores it in a database and makes predictions for future changes based on prior price movements. Provides the client the ability to see likely change in currency value and act accordingly (potentially with an automated feature built into the system).  Method:  AGILE planning.  GDAX API – Python/PHP to pull from and populate my database.  Oracle DB - C# for my own API endpoints if Matlab can't access the database directly.  Matlab to analyse the data/create an interface for the user.  No hardware/software requirements other than those already supplied by university (i.e. space on university oracle server, use of university matlab license, etc.).  Learning requirements:  How to access data through GDAX API, they supply basic help on how to do so in (several languages including Python and PHP).  Storing above data in Oracle DB in Python/PHP.  Access Oracle DB through Matlab (or C# API endpoints if necessary).  Research into stock market prediction algorithms.  Matlab - anything that I require that isn't taught in AINT351.  Risks & course of action:  GDAX API discontinuing - source alternative API endpoint.  Oracle server goes offline - wait for Oracle/university to resolve, research and consult with supervisor about alternative data storage methods such as a NoSQL database that could be stored locally.  Loss of entire project - keep numerous backups.  Loss of small portions of project - use repository to ensure up to date version accessible.  Equipment loss/damage - switch from desktop to laptop/spare laptop or use university equipment such as open access  Difficulty with learning requirements mentioned above - apply more time to learning the specific skills required to complete the task.  I lack specific knowledge to complete a segment of the project - accept that all new projects have a learning process attached to them, allow extra time for parts that haven’t been attempted previously. |

### 10.2.2 PID

Project Initiation Document

Contents

|  |  |
| --- | --- |
| Chapter | Page |
| Introduction | 1 |
| Business Case | 1-2 |
| Project Objectives | 2 |
| Initial Scope | 2 |
| Resources and Dependencies | 2-3 |
| Method of Approach | 3 |
| Initial Project Plan | 3 |
| Initial Risk List | 4 |
| Quality Plan | 5 |
| Legal, Social, Ethical and/or Professional Issues | 5 |

Introduction

Cryptocurrencies such as Bitcoin are a store of monetary value with a volatile price. Purchasing them can result in huge profits or losses of fiat value depending on whether you act at the right time or not. Due to the basic needs of a human being – the need for rest, nourishment, etc. – it is impossible for a single investor to monitor the market all of the time, but with cryptocurrency exchanges never closing, there is a need to do so. It is also possible for a human to apply bias to the decision itself when the need to act arises – favouring currencies that they like the name of, have earned them profitable positions in previous experiences etc. A good night’s sleep coupled with a poor decision can very quickly turn a profitable portfolio into an unsuccessful one.

Business Case

Business Need

Currently investors are required to spend a considerable amount of time researching technology behind a particular currency to decide whether to invest or not, potentially missing the optimal opportunity to purchase and therefore missing out on potential profits. Coin prices often spike up and down following news articles, endorsements and criticisms from celebrities, and various other factors including investors simply following the market because they are worried about the value of their portfolio as one currency reduces in value, so they switch to a currency that is holding or likely to increase in value. It is impossible for an individual to follow all potential reasons for price changes simultaneously and calculate their combined influence on the price at any given time, let alone to operate 24/7. It is also possible to ignore such signs due to an affiliation with a particular currency based on past performance.

Business Objectives

Develop a system that allows for automated trading in a way that:

1. Removes the need for user interaction once the system has been initialised.
2. Allows the user to automatically cash out profits or the initial investment once a target has been reached to prevent loss.
3. Improves trading efficiency by:
   1. Reacting to the market rather than allowing human biases to interfere.
   2. Operating whenever the application is active.
   3. Ensuring no human error causes miscalculations.

Project Objectives

1. To implement a system that collects data from the GDAX API endpoint and calculates price changes.
2. To implement a process that interprets whether action needs to be taken and acts accordingly.
3. To implement a process that allows the user to protect investment capital.
4. To implement a process that retrains the system as necessary.

Initial Scope

1. The application collects trading information and calculates the price change percentage against the previous collection, displaying this change to the user.
2. The application is able to make an interpretation of market movements.
   1. The application is able to make a recommendation based upon its interpretation of market movements.
   2. The application is able to act based upon its interpretation of the market movements.
3. The user is able to protect investment:
   1. Option to cash out profits of a set amount when the currency’s value reaches a certain target. (Recurring if constant price rises).
   2. Option to cash out a set amount of the currency once it reaches a certain value. (Single withdrawal to protect investment).
4. The system monitors its prediction accuracy and can be retrained.
   1. Retraining button.
   2. Automated retraining after set amount of time.

Resources and Dependencies

Space on the Plymouth University Oracle server (Larry) has been requested. Two contingency plans are in place in the event that it isn’t available:

* Space on Xserve is available should I require it because Larry is unavailable.
* I am able to gain access to an old laptop that can be wiped and used, though a free technology will be used (Neo4j is among the candidates).

Method of Approach

Due to the fact that there are several easily distinguishable tasks within the project, software development will employ an incremental approach with 7 increments:

1. Collect data from source, calculate average prices and store in database.
2. Output recommendation based on recently collected prices.
3. Automated trading based on recently collected prices.
4. Investment protection options.
5. Create recommendation based on more complex system (neural network).
6. System prediction accuracy monitoring including option to retrain neural network.
7. Automated retraining of neural network at set intervals.

Possible technologies are:

* Java
* MySQL
* Oracle

The above list will be compared with alternatives once the project commences.

Initial Project Plan

|  |  |  |  |
| --- | --- | --- | --- |
| **Stage** | **Start Date** | **Completion Date** | **Details** |
| Initiation | 25/01/2018 | 28/01/2018 | Project initiation document. |
| Technology Evaluation | 29/01/2018 | 29/01/2018 | Review into technologies available; relevance, cost, knowledge etc. |
| Initial High-Level Design | 30/01/2018 | 31/01/2018 | Design documentation; Architecture, database design, user interface sketches. |
| Increment 1 | 01/02/2018 | 08/02/2018 | Increment requirements, design, and implementation; data collection and storage; test results. |
| Increment 2 | 09/02/2018 | 10/02/2018 | Increment requirements, design, and implementation; recommendation; test results. |
| Increment 3 | 11/02/2018 | 18/02/2018 | Increment requirements, design, and implementation; automated trading; test results. |
| Increment 4 | 19/02/2018 | 23/02/2018 | Increment requirements, design, and implementation; investment protection; test results. |
| Increment 5 | 24/02/2018 | 06/03/2018 | Increment requirements, design, and implementation; neural network; test results. |
| Increment 6 | 07/03/2018 | 14/03/2018 | Increment requirements, design, and implementation; retrain neural network; test results. |
| Increment 7 | 15/03/2018 | 23/03/2018 | Increment requirements, design, and implementation; automated retraining; test results. |
| Easter | 24/03/2018 | 15/04/2018 | Catch up / get ahead where possible. |
| System and user acceptance testing | 16/04/2018 | 22/04/2018 | Final testing, ensure final system works, test system with potential end user. |
| Assemble and complete final report | 23/04/2018 | 04/05/2018 | PRCO304 Report. |

Communication Plan

Review meetings will be held with Marco Palomino (supervisor) each Friday at 15:30. Further communication such as emails may occur as required.

Initial Risk List

|  |  |  |  |
| --- | --- | --- | --- |
| **Risk** | **Probability** | **Impact** | **Management Strategy** |
| GDAX API discontinuing | Low | High | Source alternative API endpoint. |
| Oracle server goes offline | Low | High | Wait for Oracle/university to resolve, research and consult with supervisor about alternative data storage methods such as a NoSQL database that could be stored locally. |
| Loss of entire project | Low | High | Keep numerous backups. |
| Loss of small portions of project | Medium | Low | Use repository to ensure up to date version accessible. |
| Equipment loss/damage | Medium | Low | Switch from desktop to laptop/spare laptop or use university equipment such as open access. |
| Difficulty with learning requirements mentioned above | Medium | Medium | Apply more time to learning the specific skills required to complete the task. |
| I lack specific knowledge to complete a segment of the project | High | Medium | Accept that all new projects have a learning process attached to them, allow extra time for parts that haven’t been attempted previously. |

Quality Plan

|  |  |
| --- | --- |
| **Quality Check** | **Strategy** |
| Requirements | It must be ensured that requirements are correct and relevant before implementation.  It must also be ensured that they are complete, achievable, and demonstrable after implementation.  User testing will be used. |
| Design validation | Designs will be checked against requirements.  An entity relationship diagram will be drawn up and database normalisation performed.  Software design principles will be followed. |
| Sub-system verification and validation | To be conducted with testing at the end of each increment. |
| Total system verification and validation | To be conducted at the end of implementation. |

Legal, Social, Ethical, and/or Professional Issues

The project aims to stick within the limitations of the approved ethics application.

### 10.2.3 Highlight 1

|  |
| --- |
| **PRCO304: Highlight Report** |
| **Name:** Joseph Kellaway |
| **Date:** 08/02/2018 |
| **Review of work undertaken**   * A technology review has been carried out, Java has been selected as the primary programming language for the project. * Rough designs for user interface, normalisation, entity relationship diagram and a system architecture diagram have been created. * Development of data collector underway: * Data collection from the GDAX API (RESTful endpoint) commences once per minute and average prices are calculated. Started work to replace with a WebsocketListener. * Data collection for historic data initiates on start-up at once per second, skipping the one second the other collection occurs due to GDAX maximum API requests per second. * Unsuccessful research into alternative exchange APIs that would allow faster collection of historic data.   This week 33.25 hours have been spent working on the project. |
| **Plan of work for the next week**   * Unit testing and Javadoc for implemented data collector. * Implement data storage and use of it pending conversation with supervisor. * Implement and test various price prediction algorithms (GOFAI) to make currency recommendation, select the best one(s) for use. |
| **Date(s) of supervisory meeting(s) since last Highlight**   * No meetings – exchanged emails throughout week |
| **Brief notes from supervisory meeting(s) since last Highlight**   * Marco is happy with my PID. * Contingency plans for data storage were made. |

### 10.2.4 Highlight 2

|  |
| --- |
| **PRCO304: Highlight Report** |
| **Name:** Joseph Kellaway |
| **Date:** 15/02/2018 |
| **Review of work undertaken**   * Draw up UML, ERD & architecture diagram using Lucidchart. * Bug fixes for known/discovered issues with data harvester. * JUnit testing for data harvester (including models etc.). Bug fixes uncovered by JUnit testing. * Research into static (GOFAI) stock/cryptocurrency prediction techniques. * Design and implementation of GOFAI prediction algorithm.   This week 36 hours have been spent working on the project. |
| **Plan of work for the next week**   * JUnit test GOFAI algorithm and calculate accuracy. * Integrate selected GOFAI algorithm into (sandbox/spoofed) automated trading, JUnit test. |
| **Date(s) of supervisory meeting(s) since last Highlight**   * No meetings – exchanged emails throughout week |
| **Brief notes from supervisory meeting(s) since last Highlight**   * Upload ERD and system architecture diagrams (completed) |

### 10.2.5 Highlight 3

|  |
| --- |
| **PRCO304: Highlight Report** |
| **Name:** Joseph Kellaway |
| **Date:** 22/02/2018 |
| **Review of work undertaken**   * Due to the time required to gather data to test various GOFAI algorithms (and subsequent lost developing time in the event of erroneous code), focus was switched to database integration. * Data normalisation was carried out and Oracle database tables were set up accordingly. * A C# API was implemented to expose the database to the Java application and Java code was designed and is currently being implemented to consume the API endpoints allowing for RESTful processing of the data. Creating the API took longer than anticipated due to conflicts with the Visual Studio version and a reinstallation of both Visual Studio and Oracle Developer Tools was required.   This week 32 hours have been spent working on the project. |
| **Plan of work for the next week**   * Finish Java code to handle RESTful API calls. * Fill database with trade data and test GOFAI algorithms. * Integrate selected GOFAI algorithm into (sandbox/spoofed) automated trading, JUnit test. |
| **Date(s) of supervisory meeting(s) since last Highlight**   * No meetings – exchanged emails throughout week |
| **Brief notes from supervisory meeting(s) since last Highlight**   * None |

### 10.2.6 Highlight 4

|  |
| --- |
| **PRCO304: Highlight Report** |
| **Name:** Joseph Kellaway |
| **Date:** 01/03/2018 |
| **Review of work undertaken**   * Modify API GET and PUT calls so that they could be achieved with the current compound key set up. * Expand Java application to attempt to connect to database and act accordingly – if the database connection fails it operates without the use of storage, if the database is empty is begins from scratch and if there is data missing after down time the system begins to fill gap(s) starting with the most recent data. * Use above functionality to fill database with 10,000 prices to enable testing of GOFAI algorithms.   This week 36.75 hours have been spent working on the project. |
| **Plan of work for the next week**   * Test GOFAI algorithms. * Integrate selected GOFAI algorithm into (sandbox/spoofed) automated trading, JUnit test. |
| **Date(s) of supervisory meeting(s) since last Highlight**   * No meetings – exchanged emails throughout week |
| **Brief notes from supervisory meeting(s) since last Highlight**   * None |

### 10.2.7 Highlight 5

|  |
| --- |
| **PRCO304: Highlight Report** |
| **Name:** Joseph Kellaway |
| **Date:** 08/03/2018 |
| **Review of work undertaken**   * Added table descriptions to normalisation and uploaded. * Write up MoSCoW method for project and uploaded. * Various bug fixes/logic improvements for price harvester. * Sandbox/spoof trading implementation to test GOFAI algorithms. * Fix more bugs uncovered by spoof trading.   This week 32 hours have been spent working on the project. |
| **Plan of work for the next week**   * Compare GOFAI algorithms and select best performing algorithm over extended period of time. * Develop neural network implementation. |
| **Date(s) of supervisory meeting(s) since last Highlight**   * No meetings – exchanged emails throughout week |
| **Brief notes from supervisory meeting(s) since last Highlight**   * Do user case diagrams or user requirements based on MoSCoW (or any other) method. * Upload normalisation & table description. * Consider using Google Trends or Google Finance Client with the project to monitor the market |

### 10.2.8 Highlight 6

|  |
| --- |
| **PRCO304: Highlight Report** |
| **Name:** Joseph Kellaway |
| **Date:** 15/03/2018 |
| **Review of work undertaken**   * Prepare for demo with Marco. * Build and integrate GUI to display functionality. * Rework price collector to single thread due to concurrency issue discovered testing GOFAI algorithms.   This week 33 hours have been spent working on the project. |
| **Plan of work for the next week**   * Implement trading restrictions and issuing trade orders to GDAX endpoint. * OR (pending demo with Marco) * Implement neural network trading predictions. |
| **Date(s) of supervisory meeting(s) since last Highlight**   * 09/03/2018 – 15:30 |
| **Brief notes from supervisory meeting(s) since last Highlight**   * Work towards getting current program in a position to demo it to Marco next Friday. |

### 10.2.9 Highlight 7

|  |
| --- |
| **PRCO304: Highlight Report** |
| **Name:** Joseph Kellaway |
| **Date:** 22/03/2018 |
| **Review of work undertaken**   * Connection issues fixed. * Completion of benchmarking process. * Integrate benchmarking process into GUI. * Implement spoof trading – both backend logic and GUI. * Various bug fixes.   This week 32 hours have been spent working on the project. |
| **Plan of work for the next week**   * Implement trading restrictions and issuing trade orders to GDAX endpoint. * OR (pending demo with Marco) * Implement neural network trading predictions. |
| **Date(s) of supervisory meeting(s) since last Highlight**   * 16/03/2018 – 15:30 |
| **Brief notes from supervisory meeting(s) since last Highlight**   * Fix connection issues. Continue working on GOFAI trading mode – do not touch neural network until project cycle complete without it. |

## 10.3 Appendix 3 – Designs

### 10.3.1 System Architecture Diagram

### 10.3.2 UML Diagram

### 10.3.3 Technology Review

### 10.3.4 Interface Sketches

### 10.3.5 Entity Relationship Diagram

### 10.3.6 Database Normalisation

### 10.3.7 Oracle Create Statements

#### PRCO304\_CURRENCY

CREATE TABLE "JOSEPHKELLAWAY"."PRCO304\_CURRENCY"

("CURRENCY\_ID" VARCHAR2(5 BYTE) NOT NULL ENABLE,

"CURRENCY\_NAME" VARCHAR2(30 BYTE) NOT NULL ENABLE,

"GDAX\_ENDPOINT" VARCHAR2(60 BYTE) NOT NULL ENABLE,

CONSTRAINT "PRCO304\_CURRENCY\_PK" PRIMARY KEY ("CURRENCY\_ID")

USING INDEX PCTFREE 10 INITRANS 2 MAXTRANS 255 COMPUTE STATISTICS

STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645

PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1

BUFFER\_POOL DEFAULT FLASH\_CACHE DEFAULT CELL\_FLASH\_CACHE DEFAULT)

TABLESPACE "USERS" ENABLE

) SEGMENT CREATION IMMEDIATE

PCTFREE 10 PCTUSED 40 INITRANS 1 MAXTRANS 255

NOCOMPRESS LOGGING

STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645

PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1

BUFFER\_POOL DEFAULT FLASH\_CACHE DEFAULT CELL\_FLASH\_CACHE DEFAULT)

TABLESPACE "USERS";

#### PRCO304\_EXCHANGERATE

CREATE TABLE "JOSEPHKELLAWAY"."PRCO304\_EXCHANGERATE"

("CURRENCY\_ID" VARCHAR2(5 BYTE) NOT NULL ENABLE,

"DATETIME" TIMESTAMP (0) NOT NULL ENABLE,

"DOLLAR\_VALUE" NUMBER(8,2) NOT NULL ENABLE,

"GROWTH" NUMBER(7,4),

"LAST\_GDAXTRADE" NUMBER(10,0) NOT NULL ENABLE,

CONSTRAINT "PRCO304\_EXCHANGERATE\_PK" PRIMARY KEY ("CURRENCY\_ID", "DATETIME")

USING INDEX PCTFREE 10 INITRANS 2 MAXTRANS 255 COMPUTE STATISTICS

STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645

PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1

BUFFER\_POOL DEFAULT FLASH\_CACHE DEFAULT CELL\_FLASH\_CACHE DEFAULT)

TABLESPACE "USERS" ENABLE,

CONSTRAINT "PRCO304\_EXCHANGERATE\_FK1" FOREIGN KEY ("CURRENCY\_ID")

REFERENCES "JOSEPHKELLAWAY"."PRCO304\_CURRENCY" ("CURRENCY\_ID") ENABLE

) SEGMENT CREATION IMMEDIATE

PCTFREE 10 PCTUSED 40 INITRANS 1 MAXTRANS 255

NOCOMPRESS LOGGING

STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645

PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1

BUFFER\_POOL DEFAULT FLASH\_CACHE DEFAULT CELL\_FLASH\_CACHE DEFAULT)

TABLESPACE "USERS";

## 10.4 Appendix 4 – Test Results

## 10.5 Appendix 5 – User Testing

### 10.5.1 Instructions

### 10.5.2 Questionnaire