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 of in the near future in order to exchange to the currency that will grow the most with the aim of making a profit.

The report begins with a summary of cryptocurrency and blockchain technologies to educate the reader on the topic 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 on 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](#_Toc513223186)

[Abstract 2](#_Toc513223187)

[Table of Contents 2](#_Toc513223188)

[1 Background, Objectives, & Deliverables 5](#_Toc513223189)

[1.1 Concerning Cryptocurrencies 5](#_Toc513223190)

[1.2 Concerning Blockchain 6](#_Toc513223191)

[1.3 Introduction 6](#_Toc513223192)

[1.3.1 Statement 6](#_Toc513223193)

[1.3.2 Motivation 6](#_Toc513223194)

[1.3.3 Goals 7](#_Toc513223195)

[1.4 Competitor Applications 7](#_Toc513223196)

[1.4.1 Competitors 7](#_Toc513223197)

[1.4.2 Analysis 8](#_Toc513223198)

[1.5 Overview of Solution 9](#_Toc513223199)

[1.6 Objectives 9](#_Toc513223200)

[1.7 Requirements 9](#_Toc513223201)

[1.8 Initial Scope 10](#_Toc513223202)

[1.9 Deliverable Parts 11](#_Toc513223203)

[1.9.1 Desktop Application 11](#_Toc513223204)

[1.9.2 Storage & API 11](#_Toc513223205)

[1.10 Achieving Profitability 11](#_Toc513223206)

[1.10.1 Buy Low, Sell High 11](#_Toc513223207)

[1.10.2 Momentum Trading 11](#_Toc513223208)

[1.10.3 Analysis 12](#_Toc513223209)

[1.11 Quantifying Momentum 12](#_Toc513223210)

[2 Method of Approach 13](#_Toc513223211)

[2.1 Project Management 13](#_Toc513223212)

[2.2 Tools 13](#_Toc513223213)

[2.2.1 Desktop Application 13](#_Toc513223214)

[2.2.2 Storage & API 14](#_Toc513223215)

[2.3 Version Control 14](#_Toc513223216)

[2.4 Paperwork 14](#_Toc513223217)

[3 Development Stages 14](#_Toc513223218)

[3.1 Stage 0 – Initial Design 14](#_Toc513223219)

[3.2 Stage 1 – Data Harvester 16](#_Toc513223220)

[3.3 Stage 2 – Data Storage 17](#_Toc513223221)

[3.4 Stage 3 – API & Integration 18](#_Toc513223222)

[3.5 Stage 4 – Prediction Algorithms 19](#_Toc513223223)

[3.6 Stage 5 – GUI 21](#_Toc513223224)

[3.7 Stage 6 – Benchmarking 25](#_Toc513223225)

[3.8 Stage 7 – User Trading 26](#_Toc513223226)

[4 Evaluation & Testing 27](#_Toc513223227)

[4.1 MoSCoW 27](#_Toc513223228)

[4.2 JUnit Testing 27](#_Toc513223229)

[4.3 User Testing 28](#_Toc513223230)

[5 Legal, Social, Ethical, & Professional Issues 28](#_Toc513223231)

[5.1 Ethical 28](#_Toc513223232)

[5.2 Security 29](#_Toc513223233)

[6 End-Project Report 30](#_Toc513223234)

[6.1 MoSCoW Review 30](#_Toc513223235)

[6.2 Project Objectives Review 30](#_Toc513223236)

[6.3 Changes 30](#_Toc513223237)

[6.3.1 Changes from the Proposal 31](#_Toc513223238)

[6.3.2 Changes from the PID 31](#_Toc513223239)

[7 Project Post-Mortem 31](#_Toc513223240)

[8 Conclusions 32](#_Toc513223241)

[9 References 33](#_Toc513223242)

[10 Appendices 34](#_Toc513223243)

[10.1 Appendix 1 – User Guide 34](#_Toc513223244)

[10.1.1 Requirements 34](#_Toc513223245)

[10.1.2 Database Connection 34](#_Toc513223246)

[10.1.3 Launching the Application 35](#_Toc513223247)

[10.1.4 Trading 35](#_Toc513223248)

[10.2 Appendix 2 – Project Management Artefacts 35](#_Toc513223249)

[10.2.1 Proposal 35](#_Toc513223250)

[10.2.2 PID 36](#_Toc513223251)

[10.2.3 Highlights 41](#_Toc513223252)

[10.2.4 End-Stage Reports 46](#_Toc513223253)

[10.2.5 Log of Hours 54](#_Toc513223254)

[10.3 Appendix 3 – Designs 54](#_Toc513223255)

[10.3.1 System Architecture Diagram 54](#_Toc513223256)

[10.3.2 UML Class Diagram 54](#_Toc513223257)

[10.3.3 Technology Review 55](#_Toc513223258)

[10.3.4 Interface Sketches 56](#_Toc513223259)

[10.3.5 Entity Relationship Diagram 56](#_Toc513223260)

[10.3.6 Database Normalisation 57](#_Toc513223261)

[10.3.7 Oracle Create Statements 57](#_Toc513223262)

[10.4 Appendix 4 – Benchmarking 59](#_Toc513223263)

[10.4.1 Results 60](#_Toc513223264)

[10.4.2 Conclusions 62](#_Toc513223265)

[10.5 Appendix 5 – User Testing 63](#_Toc513223266)

[10.5.1 Instructions 63](#_Toc513223267)

[10.5.2 Questionnaire 64](#_Toc513223268)

[10.5.3 Feedback 65](#_Toc513223269)

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 others 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, online 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 publicly visible transaction ledger that is completely decentralised and 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 longevity of cryptocurrencies existence.

### 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 opportunities 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 Competitor Applications

### 1.4.1 Competitors

CryptoWorldEvolution (Cryptoworldevolution.trade, 2018) offer three different applications with four different subscription options varying from $500 to $2000 per annum. The automated trading application packages start at $1000 and all packages have a maximum currency value to be traded at one time (with the $2000 package allowing up to 2 BTC). Statistics available on the website offer no profit figures but focus on the applications’ successful transaction rates (the best being approximately 99%). There is a disclaimer stating profit is reliant on both the market and user’s own decision making. User safety is guaranteed by setting up an API key for the application to use. The API key can have various permissions denied, including withdrawals.

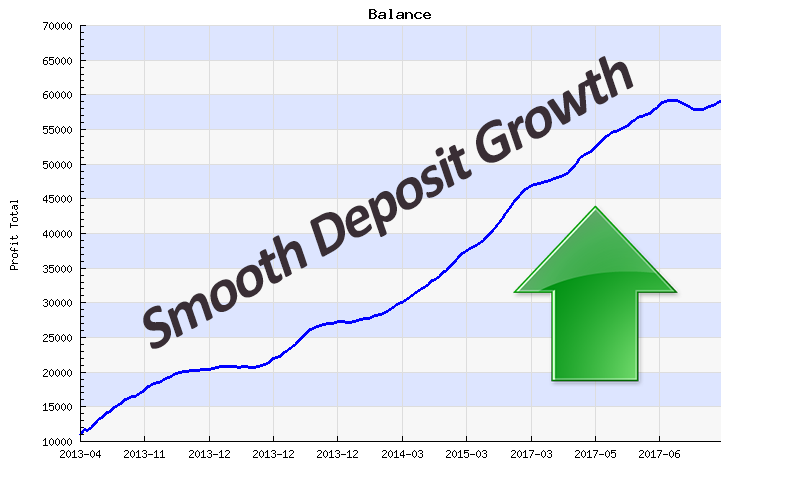


Figure 1: BTC Robot trading profit claim (Btcrobot.com, 2018).

BTC Robot (Btcrobot.com, 2018) offers three different subscriptions varying from $239.88 to $479.88 per annum. All subscriptions are automated with all but the cheapest offering the ability to run on 3 cryptocurrency exchanges simultaneously. The above chart suggests over 50 months of trading the application made an account $50,000 profit, which would have cost at most $2,399.40. This trading profit is historic, and the most recent trading profits on the chart appear to have stagnated. There are a variety of disclaimers explaining that profit is not guaranteed and a liability waiver, however, there is a 60-day money back guarantee.

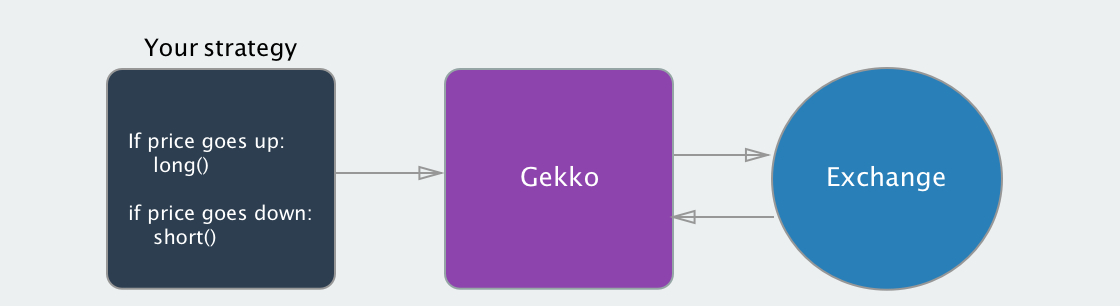


Figure 2: Gekko trading application explained (Gekko.wizb.it, 2018).

Gekko (Gekko.wizb.it, 2018) is an open-source trading application that aggregates market data, performs live and simulated trading, calculates profits, shows results graphically and performs a variety of other functionalities. It can be used with 25 different cryptocurrency exchanges and allows the user to create their own trading strategies to react to live market data as it is collected. Gekko offers no guarantees of profit as the user is responsible for creating their own trading algorithms, but users are still advised they are doing so at their own risk.

### 1.4.2 Analysis

The information on both CryptoWorldEvolution and BTC Robot largely resemble a sales pitch, however, some useful details can be extracted:

* The application accepting an API key allows the user a layer of security from attackers. It also reduces the amount of focus required on security for the developer.
* The project itself is possible since these applications claim to make a profit.
* Using the application made for this project would save the user at least $239.88 per year before trading initiates based on subscription fees.

The information available through researching Gekko offers some additional insight for planning the application:

* Allowing users to create their own algorithms is beyond the scope of this project, but allowing the users to select from a multitude of algorithms after showing them recent performance data distances the developer from ethical issues regarding losses.

## 1.5 Overview of Solution

An overview of the solution to the discussed issues:

1. A system that collects data from the chosen cryptocurrency exchange. Calculates 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.

## 1.6 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.

## 1.7 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.

## 1.8 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.

## 1.9 Deliverable Parts

The project can be split into two main deliverable parts:

### 1.9.1 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](#_10.3.3_Technology_Review) 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).

### 1.9.2 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](#_10.3.3_Technology_Review). 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.

## 1.10 Achieving Profitability

Trading cryptocurrencies for a profit is currently synonymous with trading stocks because they are not currently widely accepted as a form of currency for everyday transactions. This means that researching stock trading strategies provides a useful insight into suitable cryptocurrency trading strategies.

### 1.10.1 Buy Low, Sell High

At first this strategy appears to simply be “sell at a higher price than you purchase”, but research suggests otherwise (Koolen and Vovk, 2014). The strategy behind this label is actually to assess the volume of purchases and sales against their normal levels. This essentially means acting against the typical trader’s current strategy – if many people are selling because they fear loss (such as during the 2008 financial crisis) then purchasing in the belief that the stock is being devalued by its availability is the correct move, and if many people are purchasing then selling to take advantage of high demand overinflating value is the correct move.

### 1.10.2 Momentum Trading

This strategy relies on purchasing stocks whilst they are growing in value, in the belief that they will continue growing afterwards (Lui, Strong and Xu, 1999). Momentum investing was proven to exist in the UK stock market in the late 20th century, with many stocks maintaining growth beyond their value over the course of several months. Making use of this knowledge has allowed investors profitable strategies based around the concept.

### 1.10.3 Analysis

Given that the intention of the project was to make long-term trading profits, either of these strategies would appear suitable. Momentum trading was highlighted as preferable because:

* Momentum trading provides a more natural feeling route to profitability to the end user since short-term losses are extremely likely with the “Buy Low, Sell High” approach.
* Extra data storage would be required to hold the number of purchase orders and sale orders each minute with the “Buy Low, Sell High” approach, not just actual sales data but incomplete orders.

## 1.11 Quantifying Momentum



Figure 3: ETH-USD chart 1st May 2018 (Gdax.com, 2018).

Calculating the growth of a cryptocurrency from one time instance to the next is simply:

https://latex.codecogs.com/gif.latex?%5Cfn_phv%20Growth%20%3D%20Price_%7Bn%7D%20-%20Price_%7Bn-1%7D

With positive results showing an increase in value (positive momentum), a result close to 0 showing low change in value (low momentum), and a negative result showing a decrease in value (negative momentum). Normalising this momentum is required to calculate how much profit would have been made. This is easily done by converting this change into a percentage of the original price:

https://latex.codecogs.com/gif.latex?%5Cfn_phv%20GrowthPercentage%20%3D%20%5Cfrac%7BGrowth%7D%7BPrice_%7Bn-1%7D%7D*100

Assuming that the trend continues from the previous minute then profit will be made, or losses avoided by purchasing a cryptocurrency as it is growing and selling it at the start of any streak of shrinking value.

This method can be averaged over several instances of trading if multiple instances of momentum are required for the prediction.

# 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 prediction algorithms.
5. Create GUI.
6. Benchmark the prediction 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 [(table create statements)](#_10.3.7_Oracle_Create).

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, so the main benefits of using version control were maintaining familiarity with the terminal commands and having an additional backup of the project.

## 2.4 Paperwork

The report was written using Microsoft Office 365’s Word, with diagrams drawn using Lucidchart (2018), Codecogs’ (2018) Latex editor was used for equations, 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.

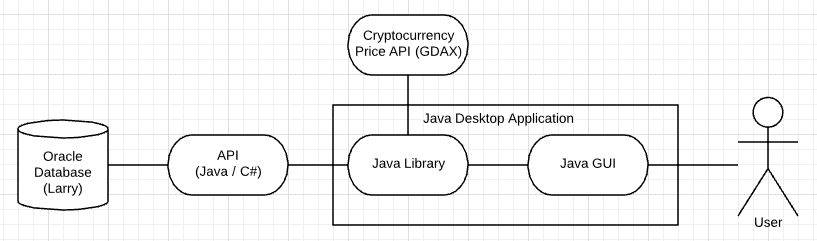


Figure 4: System architecture diagram.

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 instructions on how to trade based on the predictions through the GUI would be illogical. Instead the user would be able to see summarised 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.

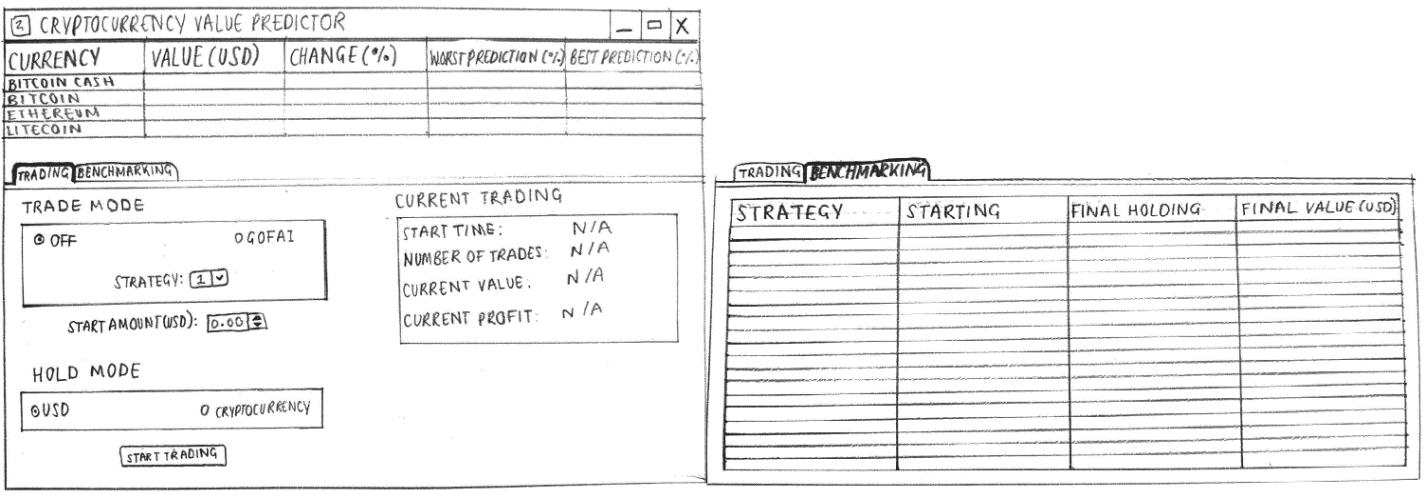


Figure 5: GUI Sketch.

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.

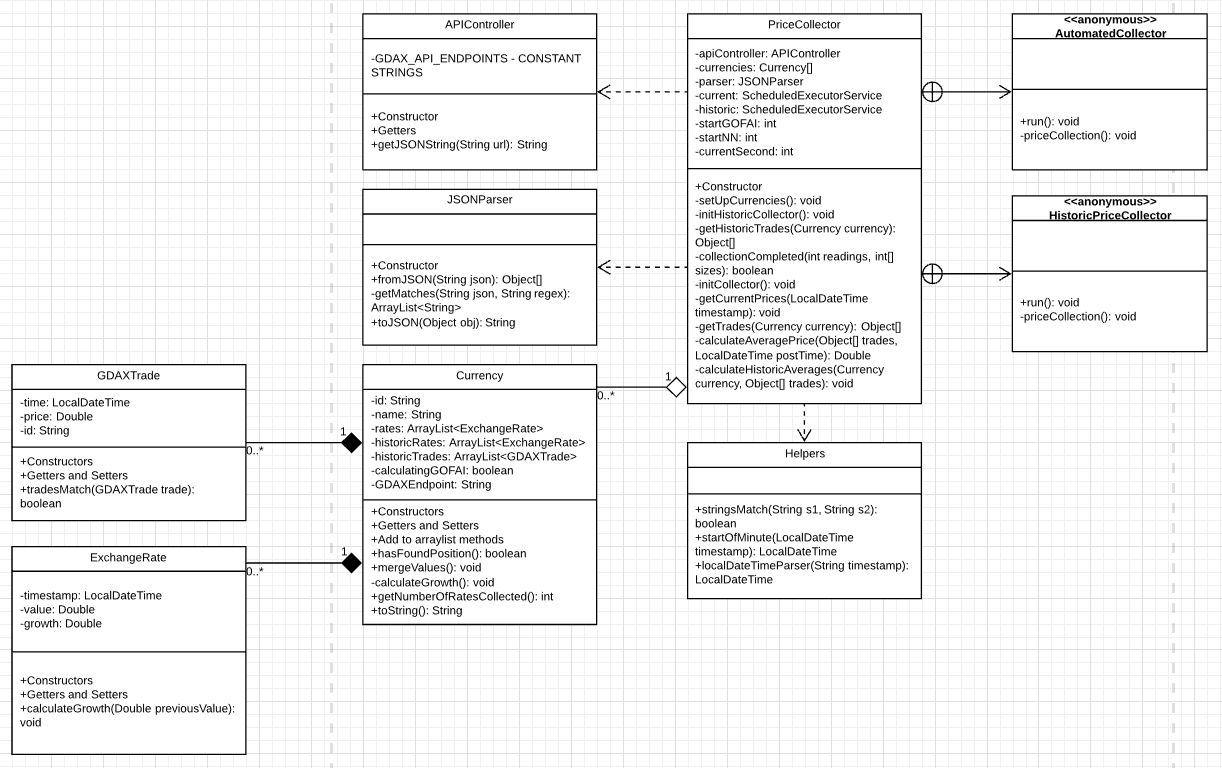


Figure 6: The data harvester’s initial class diagram. [Final class diagram](#_10.3.2_UML_Diagram).

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 prediction 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.

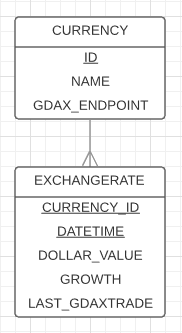


Figure 7: Project entity relationship diagram.

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.

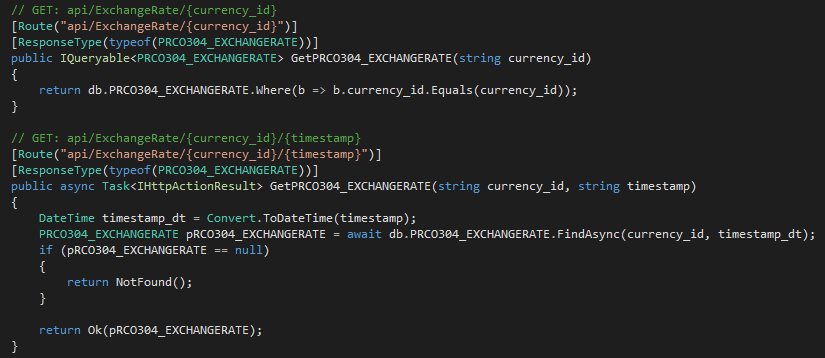


Figure 8: API modifications to allow use of the composite key.

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 – Prediction Algorithms

After data was being harvested and stored it would become possible to analyse prediction 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 prediction algorithm it was given.

The GOFAI predictor would ultimately attempt to find which Currency had the most [momentum](#_1.11_Quantifying_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, the second would be the average of the current and previous growths, 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:

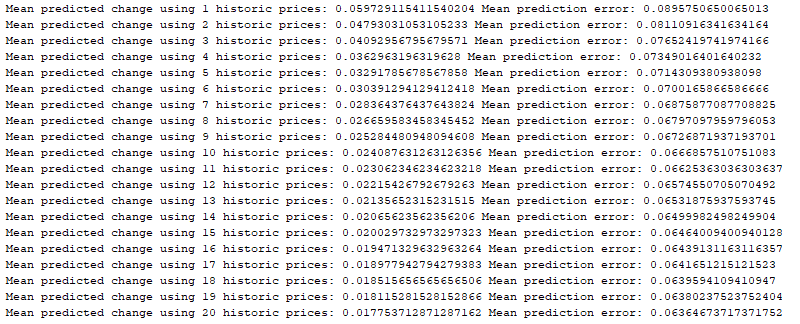


Figure 9: Prediction algorithms means and errors.

Using a larger number of prices meant less error but it also meant a much more conservative prediction, which wasn’t necessarily 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 benefitted the project in a number of 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, reducing power usage and the demands on their machine, 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”.

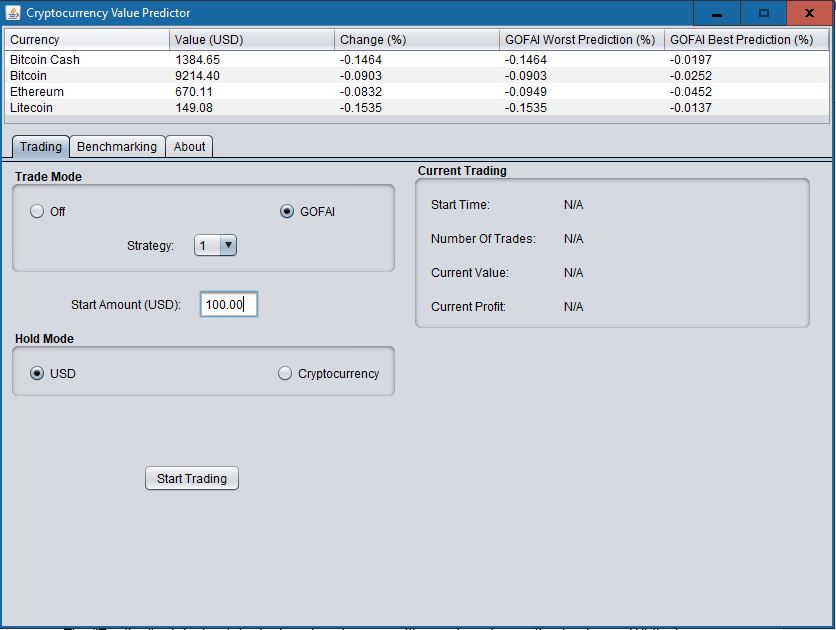


Figure 10: Default GUI.

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).

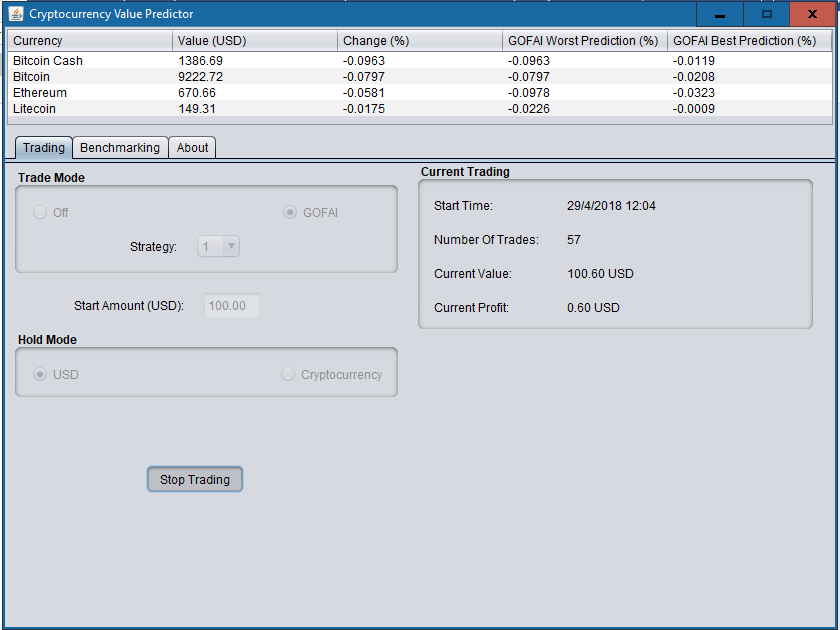


Figure 11: Trading.

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.

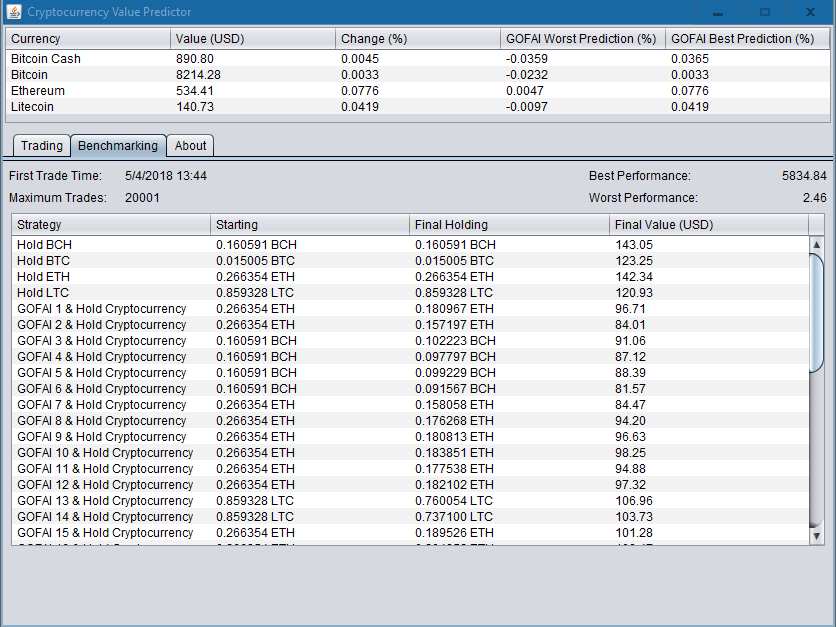


Figure 12: Benchmarking

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.

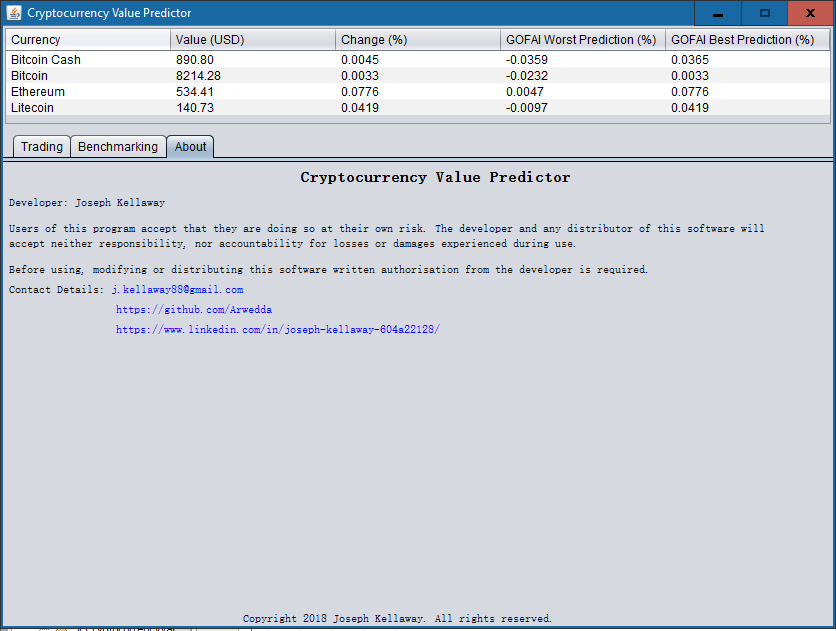


Figure 13: About tab.

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.



Figure 14: Default sorting of numerical Strings.

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, 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.

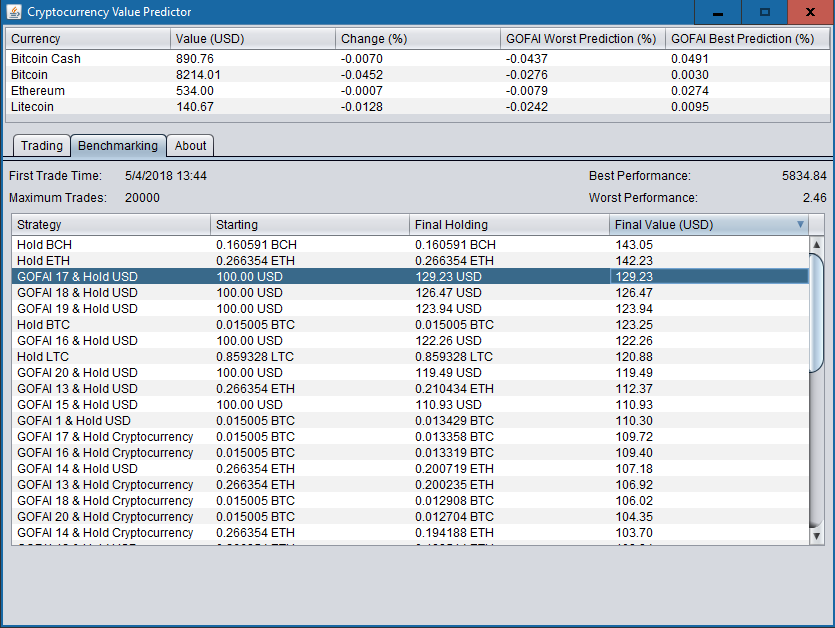


Figure 15: Sorting algorithm.

## 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 to hold the ID of the current currency being held and thej 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 data 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 [(Figure 12)](#_3.6_Stage_5).

As [figure 9](#_3.5_Stage_4) indicates, investigation into the performance of algorithms proved it is impossible to single out a single algorithm as the one with the best [performance](#_10.4_Appendix_4) 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 than the 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, coupled with new European Union laws coming in to force regarding the storage of data (General Data Protection Regulation, EU 2016/679), led to a variety of decisions with regards to data storage:

Avoid unnecessary storage of data – 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.

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 an 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 prediction 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 today’s 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.

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)*.

Btcrobot.com. (2018). *BTC Robot 2.0 - The World's First Automated Bitcoin Trading Robot!*. [online] Available at: <http://btcrobot.com/> [Accessed 30 Apr. 2018].

Codecogs. (2018). *LaTeX Editor*. [online] Available at: <https://www.codecogs.com/eqnedit.php> [Accessed 1 May 2018].

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

Cryptoworldevolution.trade. (2018). *Crypto World Evolution*. [online] Available at: <https://cryptoworldevolution.trade/> [Accessed 30 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].

Gdax.com. (2018). *Global Digital Asset Exchange - GDAX*. [online] Available at: <https://www.gdax.com/trade/ETH-USD> [Accessed 1 May 2018].

Gekko.wizb.it. (2018). *Gekko - Open source bitcoin trading bot platform*. [online] Available at: <https://gekko.wizb.it/> [Accessed 30 Apr. 2018].

*General Data Protection Regulation. EU 2016/679*.

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].

Investopedia. (2018). *Momentum Investing*. [online] Available at: <https://www.investopedia.com/terms/m/momentum_investing.asp> [Accessed 1 May 2018].

Koolen, W. and Vovk, V. (2014). Buy low, sell high. *Theoretical Computer Science*, 558, pp.144-158.

Lucidchart. (2018). *Online Diagram Software & Visual Solution | Lucidchart*. [online] Available at: <https://www.lucidchart.com/> [Accessed 1 May 2018].

Lui, W., Strong, N. and Xu, X. (1999). The Profitability of Momentum Investing. *Journal of Business Finance & Accounting*, 26(9-10), pp.1043-1091.

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

### 10.1.1 Requirements

Minimum requirements haven’t been tested, however, the entire system has been run on a system with the following:

|  |  |
| --- | --- |
| Operating System | 64-bit Windows or equivalent |
| CPU | Intel I7-3537U Dual Core Processor |
| GPU | Intel Integrated HD Graphics 3000 |
| RAM | 8GB |
| HDD Space | 111MB |

### 10.1.2 Database Connection

A connection to the database may only be made from within the Plymouth University network (or through using a VPN).

Launch “CryptocurrencyPredictorAPI.sln” located in the “CryptocurrencyPredictorAPI” folder and click “run”.

[Entity Framework](#_2.2.2_Storage_&) v6.1.3 or newer may be required for this step, as may Oracle Developer Tools for Visual Studio.

### 10.1.3 Launching the Application

Launch “CryptocurrencyValuePredictorGUI-1.0.jar” located in the “CryptocurrencyValuePredictor\CryptocurrencyValuePredictorGUI\target” folder. When the application finishes the default application view is shown as in [Figure 10](#_3.6_Stage_5).

### 10.1.4 Trading

To start trading select a trader configuration and click “Start Trading” button as shown in [Figure 10](#_3.6_Stage_5).

To finish trading click the “Stop Trading” button as shown in [Figure 11](#_3.6_Stage_5).

Please note that the “Start Trading” button may say “Collecting Data…”, this is because the database lacks the price data to perform price predictions.

Please also note that it is also advisable to inspect the recent trading performance of algorithms based on the “Benchmarking” tab as shown in [Figure 12](#_3.6_Stage_5) before selecting a trader configuration.

## 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 Highlights

***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. |

***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) |

***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 |

***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 |

***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 |

***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. |

***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. |

### 10.2.4 End-Stage Reports

The following end-stage reports were written to compare implementation plans with work carried out.

**Stage: Initiation, Technology Review & Design**

***Plan:***

*Time allocation:* 6 days.

*Description:*

Complete the project initiation document.

Complete the following design sketches:

* GUI.
* ERD.
* System architecture diagram.

Normalisation of data extracted from GUI sketch.

Review technologies to be used.

*Deliverables:*

1. PID.
2. GUI sketches.
3. ERD.
4. Normalisation.
5. System architecture diagram.
6. Technology review.

*Risks*:

1. Schedule overrun.
2. The plans change for some reason and therefore designs become obsolete.

***Actual:***

*Time required*: 6 days.

The time allocated for the task was sufficient to complete it.

*Deliverables*:

1. [PID](#_10.2.2_PID).
2. [GUI sketches](#_10.3.4_Interface_Sketches).
3. [ERD](#_10.3.5_Entity_Relationship).
4. [Normalisation](#_10.3.6_Database_Normalisation).
5. [System architecture diagram](#_10.3.1_System_Architecture).
6. Half of a [technology review](#_10.3.3_Technology_Review).

*Difficulties:*

Unsure on future difficulties, it was decided that the storage and API technologies would be reviewed at a later date due to some requiring more time investment than others.

*Changes:*

Only half of the technology review was completed.

**Stage: Data Harvester**

***Plan:***

*Time allocation:* 9 days.

*Description:*

This stage requires research into the GDAX API endpoint and the creation of a Java application that utilises it to collect cryptocurrency price data.

*Deliverables:*

1. A Java application that consumes data from the GDAX API endpoint and turns it into minute average price data.

*Risks*:

1. Schedule overrun.
2. Inability to use the API endpoint in this way.
3. GDAX and/or the API endpoint discontinues.

***Actual:***

*Time required*: 11 days.

Time allocated was insufficient in part due to extra research that was done into the solution.

*Description*:

The application performs API get requests to collect current and historic price data and merges them into a chronological price list of exchange rates for each cryptocurrency. The API limits requests to minimise the risk of DDOS attack, though this has the added affect of increasing the time required to collect the relevant data. To attempt to bypass this, the current collection was going to utilise a Websocket listener, allowing the maximum amount of API requests to be available to the historic collector.

*Deliverables*:

1. A Java application that consumes data from the GDAX API endpoint and turns it into minute average price data. It is able to collect both live and historic data simultaneously.

*Difficulties*:

1. The inability to use the GDAX API endpoint fully in the way that was required led to the research into alternative methods of collecting the data.
2. This task overran its allotted time due to difficulty 1.
3. A variety of bugs have been fixed (either through JUnit testing or directly attempting to fix them) and several other issues have been noticed to rarely occur. Investigation at a later date will be required.

*Changes*:

Adding historic collection was an unforeseen requirement that allows the hastening of data collection.

**Stage: Data Storage**

***Plan:***

Time allocation: 3 days.

Description:

Complete the data storage technology review and create the database.

Deliverables:

* Finished technology review.
* Database to store currency and price data.

Risks:

1. Schedule overrun.

***Actual:***

Time required: 3 days.

Description:

Oracle was selected for the database as per the [technology review](#_10.3.3_Technology_Review).

Deliverables:

* Finished technology review.
* Database with Currency and ExchangeRate tables.

Difficulties:

There were no difficulties with this stage, lost time on the previous stage did however encourage use of familiar technologies.

Changes:

There were no changes to the plan.

**Stage: API & Integration**

***Plan:***

Time allocation: 4 days.

Description:

Expose the Oracle database created in the previous stage via an API and integrate its use with the Java application made in the stage before.

Deliverables:

1. A C# API exposing the Oracle database.
2. Data Harvester functionality to load data from and save it to the database.

Risks:

1. Schedule overrun.

***Actual:***

Time required: 5 days.

Description:

Entity Framework was used to create the majority of the functionality; however, some code was [altered](#_3.4_Stage_3) due to the database’s use of a compound key.

Deliverables:

1. C# API.
2. Data Harvester modified to utilise data storage.

Difficulties:

1. Unexpected need to alter Entity Framework autogenerated code to use compound key.
2. Unforeseen installation time requirement.

Changes:

Extra development to API.

**Stage: Prediction Algorithms**

***Plan:***

Time allocation: 10 days.

Description:

Research into stock market [prediction techniques](#_1.10_Achieving_Profitability) as well as [competitor applications](#_1.4_Competitor_Applications) to understand how price prediction can be achieved. Implement prediction functionality, integrating it with the data harvester, and find a way to benchmark it.

Deliverables:

1. Java application now predicts the price of the following minute.

Risks:

1. Schedule overrun.
2. Research unable to provide prediction guidance.

***Actual:***

Time required: 8 days.

Description:

Research and implementation of the algorithms went smoothly. Benchmarking was possible from the console, but it would be significantly improved using a JTable in a GUI.

Deliverables:

1. Java application able to output predictions to console.

Difficulties:

1. Need to implement GUI to benchmark more intuitively.

Changes:

Benchmarking will commence after GUI is implemented.

**Stage: GUI**

***Plan:***

Time allocation: 10 days.

Description:

Implement the GUI that the end user will interact with to use the application. Add in a benchmarking section (that may later be removed) to benchmark algorithms during development.

Deliverables:

1. Java application that uses the functionality in the Java library to display meaningful information to the user.
2. Benchmarking section for the developer to compare prediction algorithm success levels.

Risks:

1. Schedule overrun.

***Actual:***

Time required: 8 days.

Description:

Implementation faced no serious issues. A sorting algorithm was required in addition to what was anticipated.

Deliverables:

1. Java GUI application to utilise Java library.
2. Benchmarking section for developer.

Difficulties:

1. A sorting algorithm was written to accommodate the JTable storing numbers as strings.

Changes:

Sorting algorithm required.

**Stage: Benchmarking**

***Plan:***

Time allocation: 5 days.

Description:

Develop benchmarking algorithm to compare prediction algorithms based on the difference between output and actual growth. Outputting the average predicted change over a set time period may also be useful.

Deliverables:

1. Algorithm to benchmark prediction algorithms.
2. Output integrated with GUI for more intuitive viewing.

Risks:

1. Schedule overrun.
2. Prediction algorithms unsuccessful and need refinement.
3. Benchmarking algorithms show very little.

***Actual:***

Time required: 8 days.

Description:

Initial benchmarking using error showed little, alternative based on investment profit or loss was used instead.

Deliverables:

1. Two separate benchmarking algorithms.
2. JTable GUI output for benchmarking.

Difficulties:

1. Unsuccessful benchmarking method used initially.
2. Schedule overran.

Changes:

As anticipated by the risks, the benchmarking algorithm needed to be modified.

**Stage: User Trading**

***Plan:***

Time allocation: 10 days.

Description:

Implement user trading functionality that allows the end user to select a trading strategy using the GUI input and start trading cryptocurrency based on that strategy.

Deliverables:

1. GUI modification to include user trading section.
2. Trading functionality in Java library.

Risks:

1. Schedule overrun.

***Actual:***

Time required: 11 days.

Description:

Implementation was straight forward with no huge issues.

Deliverables:

1. Trading GUI.
2. Trading functionality.

Difficulties:

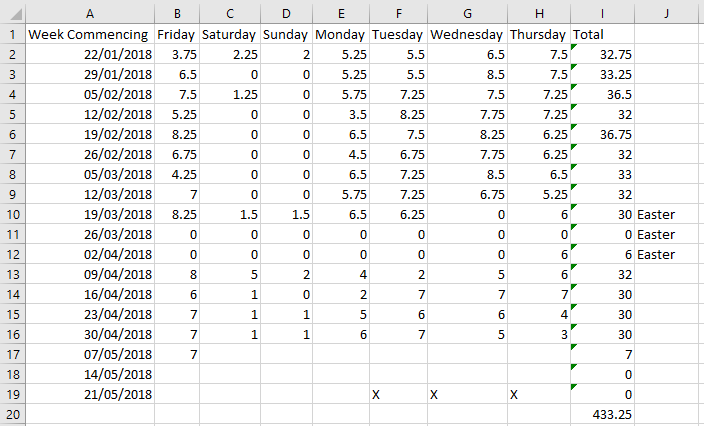
1. None.

Changes:

The benchmarking GUI was left available to the user due to the inconsistency of performance due to changes in market behaviour.

### 10.2.5 Log of Hours

To ensure that the minimum recommended hours for the project were met, a log of hours committed to the project each day was kept.



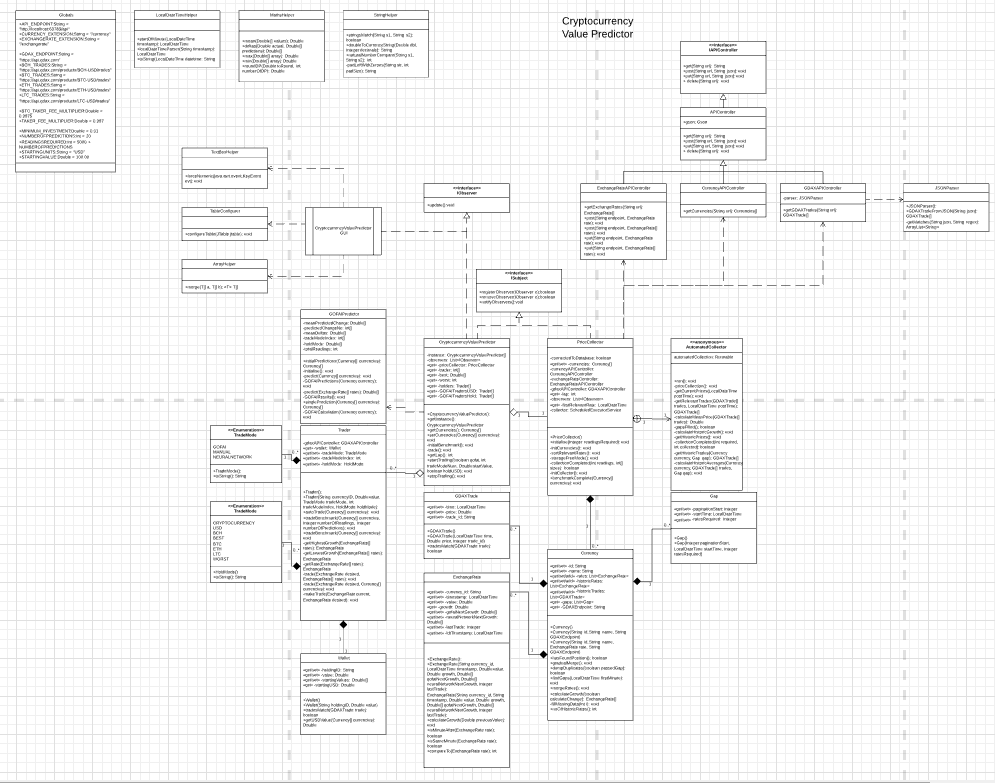
## 10.3 Appendix 3 – Designs

### 10.3.1 System Architecture Diagram

See [Figure 4](#_3.1_Stage_0).

### 10.3.2 UML Class Diagram

The following UML shows how the classes within the CryptocurrencyValuePredictor Java applications interact with each other. Helper classes and global variables that deliver functionality and values that are reused among several classes within the application can be found at the top left of the diagram. Relationships between these classes and others have been omitted to provide clarity to the diagram.



For a more intuitive view of the diagram [click here](https://www.lucidchart.com/documents/view/1dea3807-7403-4bb1-adbb-b0c0b6ccc44f/0).

### 10.3.3 Technology Review

***Core Application***

Java and C# were both highlighted as technologies suitable for the main application. Due to the project’s dependence on an unfamiliar system – the GDAX API – and the need to research into how a price can be predicted from the data available, it was decided that working with a familiar technology was logical. With the fixed time constraints of the project and potential unforeseen challenges with using with the GDAX API it would be illogical to learn a new programming language also. Another issue with using an unfamiliar programming would be the need to investigate licensing terms and conditions, and the need to find a suitable integrated development environment.

Given that Java and C# are very similar in terms of the positive attributes:

* Familiarity for the developer
* Suitable for task
* Mature languages
* Support and documentation available
* Suitable IDEs available
* License covers project

Selecting a language fell on the question “should the application be platform independent, or should it take advantage of C#’s arguable superiority on a Windows system?” Given that approximately 88.8% (Netmarketshare.com, 2018) of the world use Windows, it wouldn’t be illogical to focus on that group if it had a cost to target the other 11.2%, but there isn’t a cost, so development in Java is the logical decision.

***Data Storage & API***

Oracle and Neo4J were both highlighted as technologies for data storage. Due to the challenges discussed when reviewing the application technology, it was also necessary to remain with a familiar technology for data storage. These technologies both shared various positive attributes:

* Familiarity for the developer
* Suitable for task
* Mature languages
* Support and documentation available
* License covers project

Selecting the storage technology fell on the following two attributes:

* Oracle: free storage on campus.
* Neo4J: personal control over storage.

Given financial limitations and the ability to produce an API very quickly with Entity Framework it was decided that Oracle was the correct technology for data storage in this project.

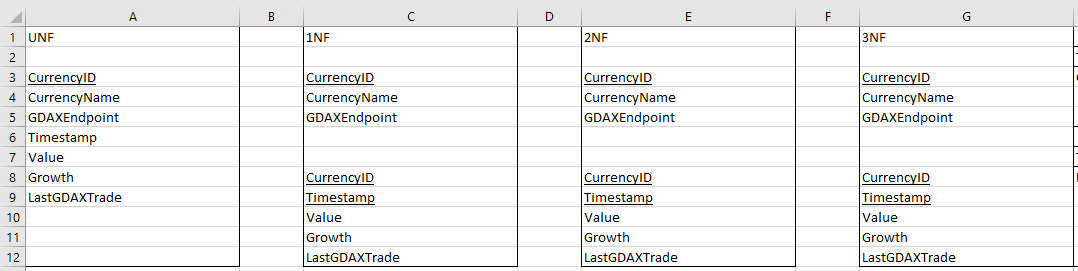
### 10.3.4 Interface Sketches

See [Figure 5](#_3.1_Stage_0).

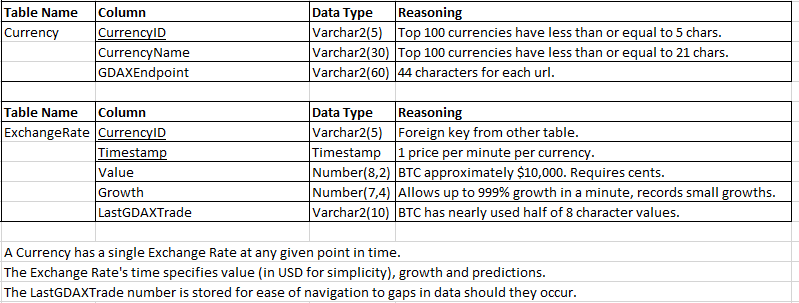
### 10.3.5 Entity Relationship Diagram

See [Figure 7](#_3.3_Stage_2).

### 10.3.6 Database Normalisation

The following database normalisation based on the information displayed in the [GUI sketch](#_3.1_Stage_0) was carried out prior to development:

This normalisation resulted in the following database tables being created:



### 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)

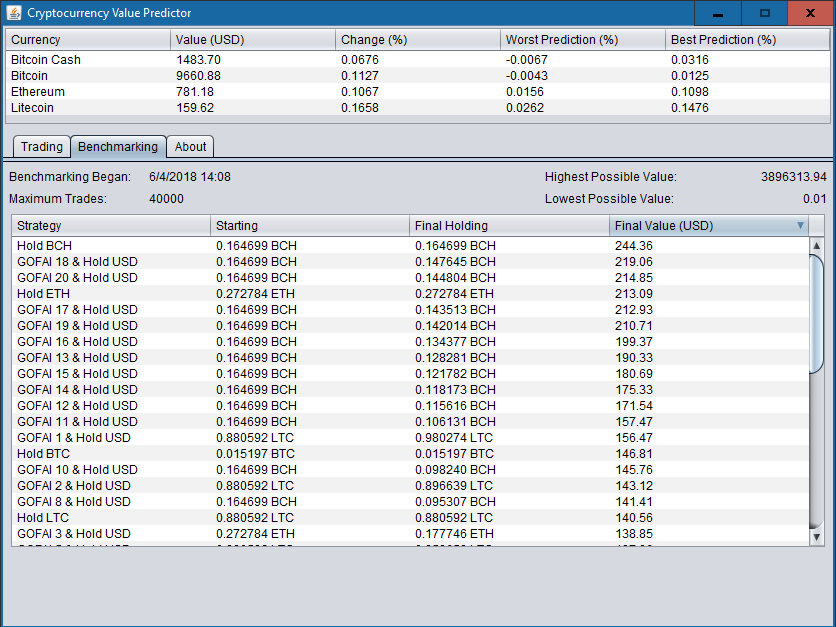
TABLESPACE "USERS";

## 10.4 Appendix 4 – Benchmarking

The following data was collected and run through the benchmarking process to show how the application could perform over an extended time period. The data was collected over several days (due to the frequency constraints on the GDAX API calls) and was run through the benchmarking process at approximately 9:00am on Friday 4th May 2018.

Each trading strategy was allocated an initial investment of 100.00 USD and given the option to trade or hold once per minute for the duration of each benchmark.

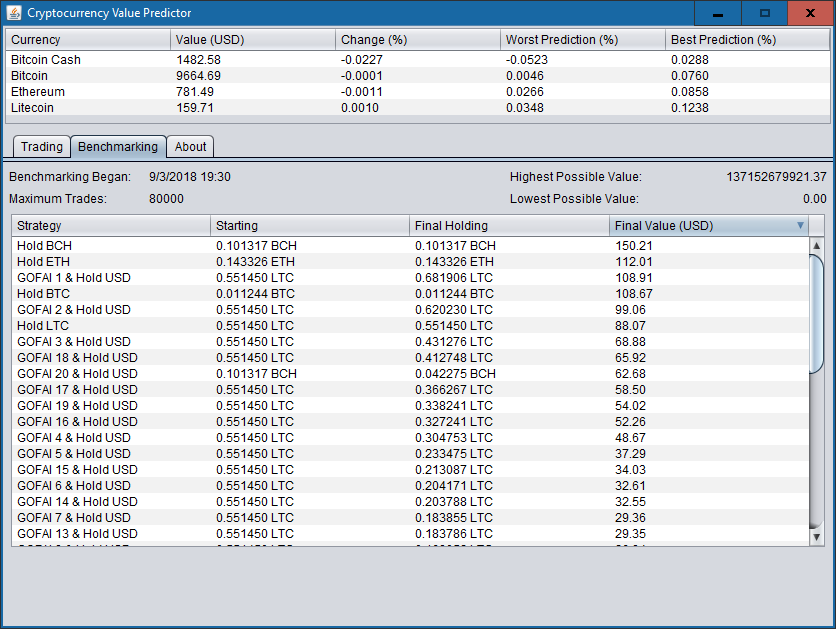
### 10.4.1 Results



40,000 minutes of price data – approximately 4 weeks of trading.

The above benchmarking shows trading for data collected between 6th April and 4th May 2018. As you can see, when comparing the trader with simply holding a single cryptocurrency 35% (14/40) of trading strategies are more successful than holding Litecoin, 27.5% were also better than holding Bitcoin, 5% were superior to holding Ethereum as well, and none were superior to holding Bitcoin Cash.

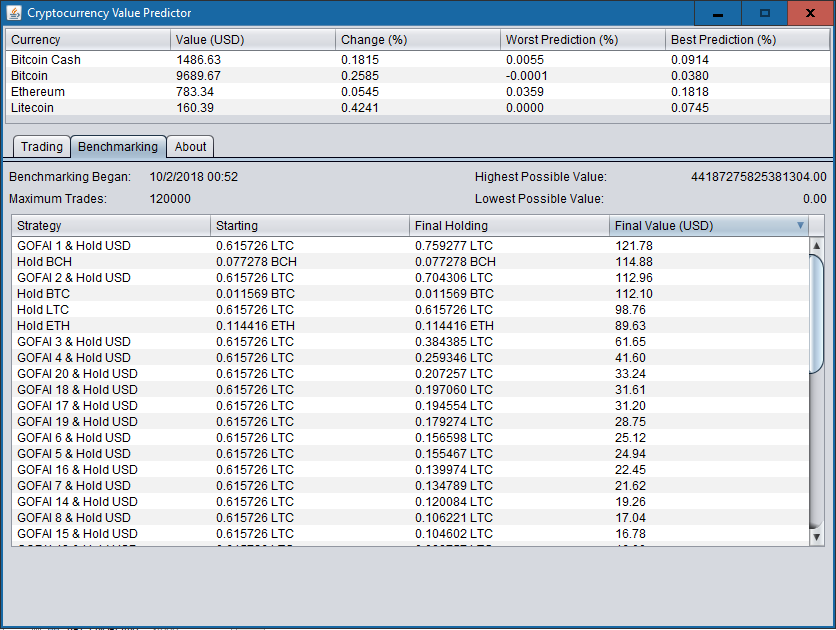
This data suggests that there is some success for traders who don’t have time to research which currencies to invest in due to the large number of strategies that reported a profit on the original investment. It is worth noting that in this trading data, the strategies that involve switching to USD when a loss is predicted for all cryptocurrencies were the only ones that were more successful than holding any cryptocurrency. It is also worth noting that few trading strategies had managed to outperform holding in terms of cryptocurrency amassed, with their final holding almost always lower than their starting amount.



80,000 minutes of price data – approximately 8 weeks of trading.

The above benchmarking shows trading for data collected between 9th March and 4th May 2018. As you can see, when comparing the trader with simply holding a single cryptocurrency 5% (2/40) of trading strategies are more successful than holding Litecoin, 2.5% were also better than holding Bitcoin, and none were superior to holding Ethereum or Bitcoin Cash.

The data suggests that there was much less success for traders over the course of the 8 weeks. It is worth noting that the two traders that did outperform at least one hold strategy did amass more cryptocurrency than they started with, even though one made a small USD loss.



120,000 minutes of price data – approximately 12 weeks of trading.

The above benchmarking shows trading for data collected between 10th February and 4th May 2018. As you can see, when comparing the trader with simply holding a single cryptocurrency 5% (2/40) of trading strategies are more successful than holding Litecoin, Ethereum or Bitcoin and 2.5% were also better than holding Bitcoin Cash.

The data suggests that as with 8 weeks of trading, there was much less success for traders. However, the two successful traders were more successful when compared with the hold strategies. Successful traders were positive in terms of both USD value and cryptocurrency amassed.

### 10.4.2 Conclusions

The above benchmarking of the collected data is somewhat inconclusive regarding selecting a superior trading strategy. It appears that strategies involving selling cryptocurrency when all four currencies are predicted to make a loss are always superior, however, running these benchmarks numerous times would be required to prove this.

The benchmarking does show that making a profit in both the short and long term is at least possible, even if it did not occur for many strategies and even if it were to not occur next time that the benchmarking was run.

The results of testing the trading strategies through benchmarking are largely positive, however, it would be prudent to allow the user to see the recent benchmarking results and select a trading strategy based on those rather than issuing a single trading strategy for them to use.

## 10.5 Appendix 5 – User Testing

### 10.5.1 Instructions

**Cryptocurrency Value Predictor User Testing Instructions**

Thank you for taking part in user testing for this final year university project. Please follow the instructions and then fill out the questionnaire. Please be aware that this process will be kept completely anonymous. This does however mean that it will be impossible to opt out after departing this session.

1. Write down the current value of “Bitcoin”: ……………………………………………………………………..
2. Write down the current growth rate of “Ethereum”: ………………………………………………………
3. Write down the time benchmarking began: …………………………………………………………………….
4. Write down the most successful trading strategy excluding the “Hold” strategies:

………………………………………………………………………………………………………………………………………………

1. Initiate trading $100.00 using the trading strategy you gave as the answer to task 5.
2. Write down the trading start time: ………………………………………………………………………………….
3. Please wait 1 minute.
4. Write down the number of trades: …………………………………………………………………………………
5. Write down the current value of your investment and current profit (if applicable):

……………………………………………………………………………………………………………………………………………..

1. Halt trading.

### 10.5.2 Questionnaire

**Cryptocurrency Value Predictor User Testing Questionnaire**

Thank you for taking part in user testing for this final year university project. Please fill out the questionnaire after following the instruction sheet. Please be aware that this process will be kept completely anonymous. This does however mean that it will be impossible to opt out after departing this session.

With 1 being very hard and 10 being very easy, please rate each task within the instructions:

1. ……………………………………………………………… 2. ………………………………………………………………

3. ……………………………………………………………… 4. ………………………………………………………………

5. ……………………………………………………………… 6. ………………………………………………………………

7. ……………………………………………………………… 8. ………………………………………………………………

9. ……………………………………………………………… 10. ……………………………………………………………

If you found any task difficult, please specify why: …………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………

How would you improve the application to make this task easier?

…………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………

Please give any other feedback on the application:

…………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………

### 10.5.3 Feedback

The following list are issues highlighted by the user testing:

* Dislike the way numbers are displayed – scientific notation is confusing.
* “Trading” tab should be default view, not “Benchmarking”.
* JSpinner not a good tool, prefer text box.
* Dislike the use of the dollar sign ($) since this can mean Canadian/Australian dollars etc.
* Label changes:
  + “Needs Data” to “Collecting Data…”
  + “First Trade Time:” to “Benchmarking Began:”
  + “Best Performance:” to “Highest Possible Value:”
  + “Worst Performance:” to “Lowest Possible Value:”
* Application should start in the centre of the screen.