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

# Abstract

# Table of Contents

Contents

[Acknowledgements 2](#_Toc511813297)

[Abstract 2](#_Toc511813298)

[Table of Contents 2](#_Toc511813299)

[1 Background, Objectives, & Deliverables 4](#_Toc511813300)

[1.1 Concerning Cryptocurrencies 4](#_Toc511813301)

[1.2 Concerning Blockchain 4](#_Toc511813302)

[1.3 Introduction 4](#_Toc511813303)

[1.3.1 Statement 4](#_Toc511813304)

[1.3.2 Motivation 4](#_Toc511813305)

[1.3.3 Goals 4](#_Toc511813306)

[1.4 Overview of Solution 5](#_Toc511813307)

[1.5 Objectives 5](#_Toc511813308)

[1.6 Requirements 5](#_Toc511813309)

[1.7 Initial Scope 5](#_Toc511813310)

[1.8 Deliverable Parts 5](#_Toc511813311)

[1.8.1 Desktop Application 5](#_Toc511813312)

[1.8.2 Storage & API 5](#_Toc511813313)

[2 Method of Approach 5](#_Toc511813314)

[2.1 Project Management 5](#_Toc511813315)

[2.2 Tools 5](#_Toc511813316)

[2.2.1 Desktop Application 5](#_Toc511813317)

[2.2.2 Storage & API 5](#_Toc511813318)

[2.3 Version Control 5](#_Toc511813319)

[3 Development Stages 6](#_Toc511813320)

[3.1 Stage 0 – Initial Design 6](#_Toc511813321)

[3.2 Stage 1 – Data Harvester 7](#_Toc511813322)

[3.3 Stage 2 – Data Storage 8](#_Toc511813323)

[3.4 Stage 3 – API & Integration 9](#_Toc511813324)

[3.5 Stage 4 – GOFAI Algorithms 9](#_Toc511813325)

[3.6 Stage 5 – GUI 11](#_Toc511813326)

[3.7 Stage 6 – Benchmarking 11](#_Toc511813327)

[3.8 Stage 7 – User Trading 11](#_Toc511813328)

[4 Evaluation & Testing 11](#_Toc511813329)

[4.1 MoSCoW 11](#_Toc511813330)

[4.2 JUnit Testing 11](#_Toc511813331)

[4.3 User Testing 11](#_Toc511813332)

[5 Legal, Social, Ethical, & Professional Issues 11](#_Toc511813333)

[5.1 Ethics 11](#_Toc511813334)

[5.2 Security 12](#_Toc511813335)

[6 End-Project Report 12](#_Toc511813336)

[7 Project Post-Mortem 12](#_Toc511813337)

[8 Conclusions 12](#_Toc511813338)

[9 References 12](#_Toc511813339)

[10 Appendices 12](#_Toc511813340)

[10.1 Appendix 1 – User Guide 12](#_Toc511813341)

[10.2 Appendix 2 – Project Management Artefacts 12](#_Toc511813342)

[10.2.1 PID 12](#_Toc511813343)

[10.2.2 Highlight 1 17](#_Toc511813344)

[10.2.3 Highlight 2 18](#_Toc511813345)

[10.2.4 Highlight 3 18](#_Toc511813346)

[10.2.5 Highlight 4 19](#_Toc511813347)

[10.2.6 Highlight 5 20](#_Toc511813348)

[10.2.7 Highlight 6 20](#_Toc511813349)

[10.2.8 Highlight 7 21](#_Toc511813350)

[10.3 Appendix 3 – Designs 21](#_Toc511813351)

[10.3.1 System Architecture Diagram 21](#_Toc511813352)

[10.3.2 UML Diagram 21](#_Toc511813353)

[10.3.3 Technology Review 22](#_Toc511813354)

[10.3.4 Interface Sketches 22](#_Toc511813355)

[10.3.5 Entity Relationship Diagram 22](#_Toc511813356)

[10.3.6 Database Normalisation 22](#_Toc511813357)

[10.3.7 Oracle Create Statements 22](#_Toc511813358)

[10.4 Appendix 4 – Test Results 23](#_Toc511813359)

[10.5 Appendix 5 – User Testing Questionnaire 23](#_Toc511813360)

Word count:

# 1 Background, Objectives, & Deliverables

## 1.1 Concerning Cryptocurrencies

## 1.2 Concerning Blockchain

## 1.3 Introduction

### 1.3.1 Statement

### 1.3.2 Motivation

### 1.3.3 Goals

## 1.4 Overview of Solution

## 1.5 Objectives

## 1.6 Requirements

## 1.7 Initial Scope

## 1.8 Deliverable Parts

### 1.8.1 Desktop Application

### 1.8.2 Storage & API

# 2 Method of Approach

## 2.1 Project Management

## 2.2 Tools

### 2.2.1 Desktop Application

### 2.2.2 Storage & API

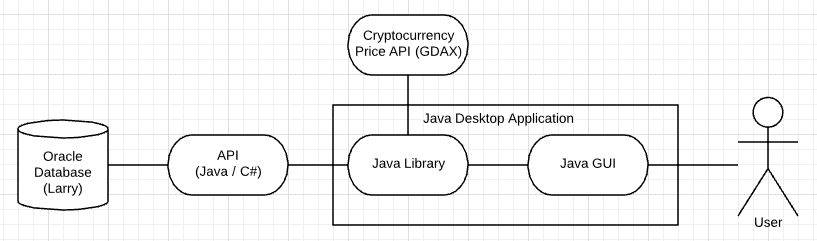
## 2.3 Version Control

# 3 Development Stages

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

## 3.1 Stage 0 – Initial Design

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



The next requirement would be to determine what data would be stored within the database. This meant briefly sketching the GUI. Due to the fact that the application is intended to be used with minimal monitoring displaying the predictions through the GUI would be illogical. Instead the user would be able to see 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.

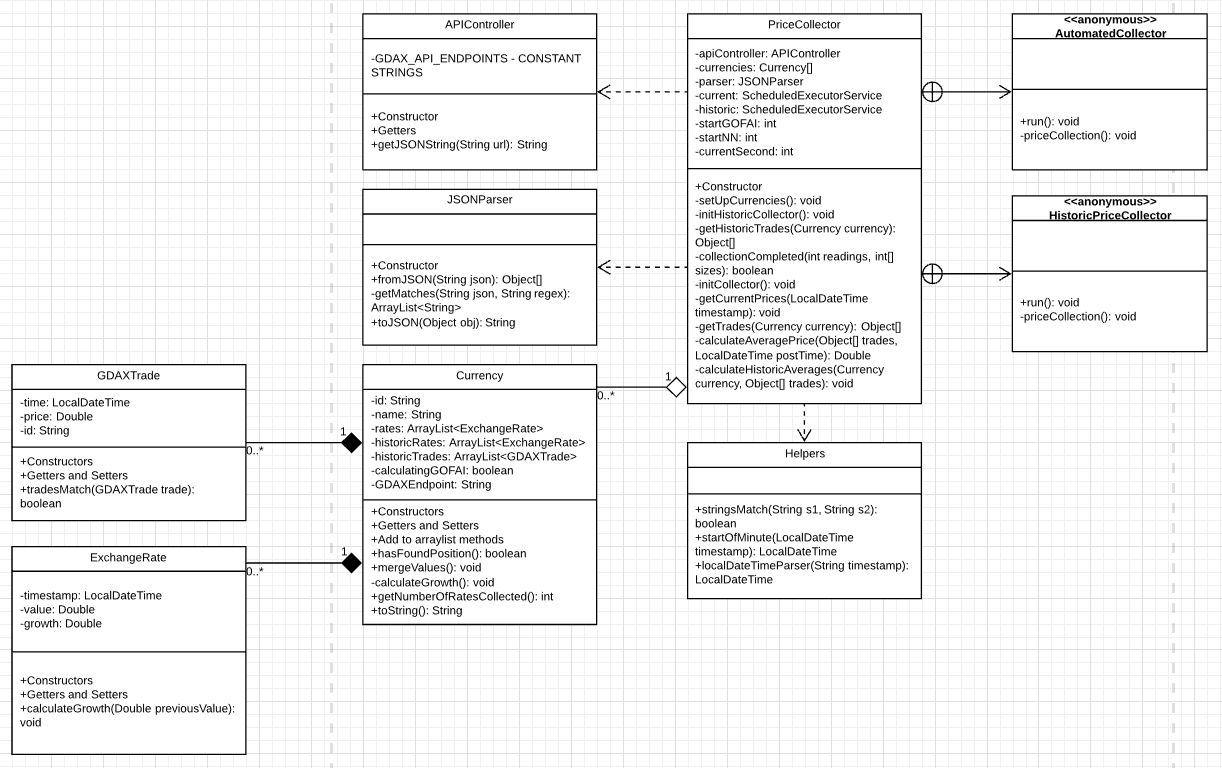
\*\*GUI Sketch

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

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

## 3.2 Stage 1 – Data Harvester

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



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

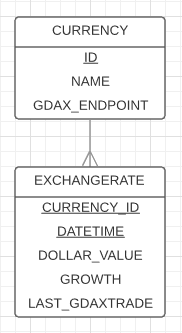
Throughout the project constant monitoring, testing, and improvement of previously implemented systems occurred to ensure that integration didn’t create issues with prior development. This resulted in the PriceCollector’s modification to using only a single ScheduledExecutorService (due to a concurrency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

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

## 3.3 Stage 2 – Data Storage

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

Data to be stored was extracted from the proposed GUI interface \*\*(GUI Sketch), then data required to access prices from the GDAX API endpoint was added and normalisation was performed to create a database with the following ERD.



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

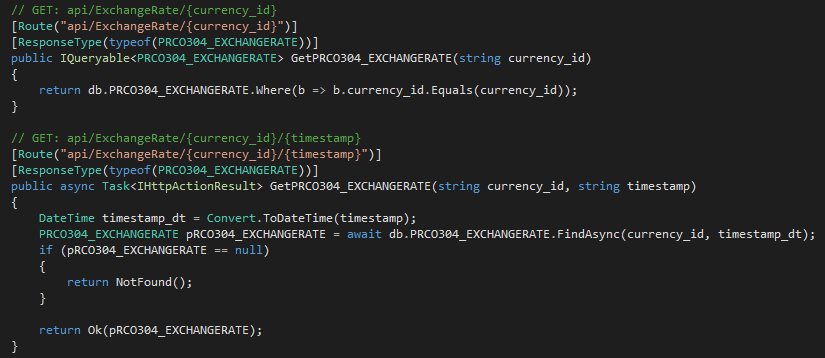
The storage medium used was Oracle as described \*\*earlier and it was created using the create statements (in Appendix 3).

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



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

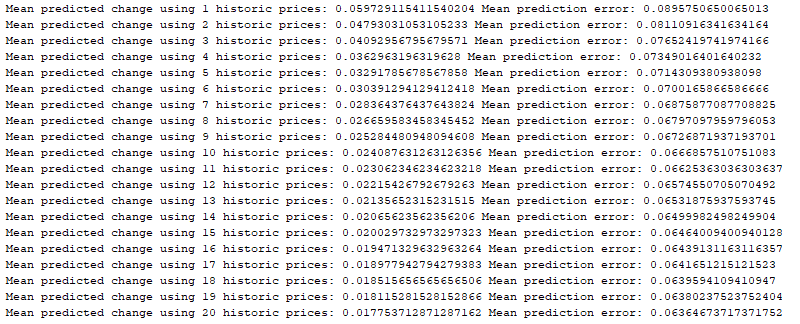
## 3.5 Stage 4 – GOFAI Algorithms

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

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

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

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

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

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

## 3.6 Stage 5 – GUI

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

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

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

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

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

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

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

Each of the JTables used has been configured so that clicking on the content selects a whole single row only. This is to allow the user to clearly see the details of a single item. Clicking a header of a column sorts the column in descending order. Descending order was chosen due to the nature of the data displayed – the user would likely want to quickly single out the best performing algorithm and select it to start trading. User testing did not notice an issue with descending order when using this feature.

## 3.7 Stage 6 – Benchmarking

## 3.8 Stage 7 – User Trading

# 4 Evaluation & Testing

## 4.1 MoSCoW

## 4.2 JUnit Testing

## 4.3 User Testing

# 5 Legal, Social, Ethical, & Professional Issues

## 5.1 Ethics

## 5.2 Security

# 6 End-Project Report

# 7 Project Post-Mortem

# 8 Conclusions

# 9 References

# 10 Appendices

## 10.1 Appendix 1 – User Guide

## 10.2 Appendix 2 – Project Management Artefacts

### 10.2.1 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.

### 10.2.2 Highlight 1

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

### 10.2.3 Highlight 2

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

### 10.2.4 Highlight 3

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

### 10.2.5 Highlight 4

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

### 10.2.6 Highlight 5

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

### 10.2.7 Highlight 6

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

### 10.2.8 Highlight 7

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

## 10.3 Appendix 3 – Designs

### 10.3.1 System Architecture Diagram

### 10.3.2 UML Diagram

### 10.3.3 Technology Review

### 10.3.4 Interface Sketches

### 10.3.5 Entity Relationship Diagram

### 10.3.6 Database Normalisation

### 10.3.7 Oracle Create Statements

#### PRCO304\_CURRENCY

CREATE TABLE "JOSEPHKELLAWAY"."PRCO304\_CURRENCY"

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

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

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

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

USING INDEX PCTFREE 10 INITRANS 2 MAXTRANS 255 COMPUTE STATISTICS

STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645

PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1

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

TABLESPACE "USERS" ENABLE

) SEGMENT CREATION IMMEDIATE

PCTFREE 10 PCTUSED 40 INITRANS 1 MAXTRANS 255

NOCOMPRESS LOGGING

STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645

PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1

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

TABLESPACE "USERS";

#### PRCO304\_EXCHANGERATE

CREATE TABLE "JOSEPHKELLAWAY"."PRCO304\_EXCHANGERATE"

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

"DATETIME" TIMESTAMP (0) NOT NULL ENABLE,

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

"GROWTH" NUMBER(7,4),

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

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

USING INDEX PCTFREE 10 INITRANS 2 MAXTRANS 255 COMPUTE STATISTICS

STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645

PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1

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

TABLESPACE "USERS" ENABLE,

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

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

) SEGMENT CREATION IMMEDIATE

PCTFREE 10 PCTUSED 40 INITRANS 1 MAXTRANS 255

NOCOMPRESS LOGGING

STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645

PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1

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

TABLESPACE "USERS";

## 10.4 Appendix 4 – Test Results

## 10.5 Appendix 5 – User Testing Questionnaire