BalanceBot

Cross-exchange Cryptoasset Portfolio Rebalancing

FINAL-YEAR PROJECT
BEN GRAY - 1877814
BSC COMPUTER SCIENCE W/ DTP (PWC)
SUPERVISOR: DR. EIKE RITTER
WORD COUNT: 9,837

Abstract

Since its birth in 2009, the burgeoning cryptoasset sector currently boasts a total valuation of over two trillion dollars, attracting interest from a wide range of investors. Due to its youth and the nature of its assets, the volatility and resultant market risk of the sector is especially high. The age-old technique of diversification and Modern Portfolio Theory provide a technique to counter this risk. However, applying diversification to cryptoassets can be difficult given the number of crypto exchange websites an investor may choose to use. This is termed the 'Multiple Exchange Problem' in this report. The project produced an iOS app which allows the user to connect their exchange accounts and gathers relevant portfolio data. It also facilitates the input of the user's portfolio preferences, specifically regarding how their portfolio should be diversified. The app uses this information run a specially-developed, randomised, population-based algorithm to determine the transactions a user should perform to rebalance their portfolio with minimum cost. The implemented algorithm successfully produces optimal solutions in a fraction of a second, although further research and development could offer increased efficiency. Overall, the project gave rise to an app that enables a casual investor to utilise diversification in a crypto portfolio spread across multiple exchanges. No current offering provides this capability free of charge or with the same level of accessibility to the lay-investor.

"Ship your grain across the sea;
after many days you may receive a return.

Invest in seven ventures, yes, in eight;
you do not know what disaster may come upon the land."

~ Ecclesiastes 11:1-2 NIV

Table of Contents

Table of Tables

Table of Figures

1. Introduction

1.1. Cryptoassets

Launched in 2009 as a 'purely peer-to-peer version of electronic cash', Bitcoin [1] was the first ever 'cryptocurrency': a digital currency secured by cryptographic signatures. It offers an alternative to government-issued currency that can be transacted digitally without reliance upon, or requiring the permission of, a trusted third party such as a bank. Instead, transactions are verified by the network's consensus, becoming more secure with more participants. Bitcoin's disinflationary maximum supply of 21 million coins, inspired by the Austrian school of economics, seemingly offers a way to protect against rising inflation, and is considered one of the main reasons for its dramatic rise in value.

Bitcoin's launch marked the inauspicious birth of the cryptoasset sector, 'crypto', now worth over two trillion dollars and comprising over ten thousand individual assets. Roughly 16% of US adults are invested in cryptocurrency [2]; leading financial institutions have included them in their activities; the sector has even begun to evidence its independence from global market sentiments [3]. Many governments, including the UK and US, have begun introducing formal regulation for cryptoassets. El Salvador has even begun accepting Bitcoin as legal tender.

Cryptoassets are clearly a desirable investment for a large number and range of investors. However, they have a significantly higher level of risk than most traditional investments for two principal reasons. First are the prevalent security risks, with supposedly 22% of bitcoin users having lost money due to security breaches or 'self-induced errors' [4]. Secondly, the market risk of holding crypto investments is huge. Notoriously, cryptoassets are unbacked by a government or central bank and, unlike stocks, most do not generate any revenue. Consequently, most cryptoassets are considered 'reflexive', suggesting price reflects public perception rather than any underlying value. This results in rapid price fluctuations that make prudent risk management far more difficult, but ultimately crucial.

1.2. Diversification

Exposure to risk is an inherent, unavoidable part of any investment. Diversification is a technique that mitigates this risk by spreading exposure across multiple investments. The use of diversification is almost as old as civilisation itself. The seafaring merchants of the ancient Assyrian city of Ur would split their capital between multiple ships' voyages to prevent the loss of the entire sum should a ship be lost at sea [5].

Modern Portfolio Theory (MPT) [6] offers a contemporary framework for diversification, maximising returns whilst minimising risk. Using MPT, a portfolio is split across asset classes in varying proportions. The proportions are calculated based on the risk of the asset class and the investor's risk tolerance. The risk of an asset class is measured using the standard deviation of its average return and is deemed risky if it's volatile, i.e, its returns are inconsistent and wide-ranging. When implemented optimally, diversification reduces a portfolio's overall volatility to lower than that of its least volatile constituent asset class [7] while maintaining overall expected returns [8]. A classic portfolio allocation is a split between stocks and bonds, for example, 70% bonds, 30% stocks.

As the prices of assets contained in a portfolio fluctuate, the percentage allocated to each asset class varies. Rebalancing a portfolio resets its allocation to the originally defined proportions by selling some of the assets that have grown and adding to the asset classes that have shrunk. It is performed either periodically, or when the proportions of asset classes in a portfolio deviate too far from the originally defined percentages, based on a threshold.

For example, a $6,000 portfolio is originally split evenly between 2 asset classes A and B, each worth $3,000. Due to changes in asset prices, the value of A rises to $4,000 and B to $6,000. The portfolio, now worth a total of $10,000, is split 40/60 instead of the target 50/50. To return to the target allocation, $1,000 of asset class B must be sold and the money used to buy $1,000 more of A so both are equally worth $5,000. This example, shown in Figure 1, is a rebalance of just two asset classes, but the same method is applicable to any number of asset classes.

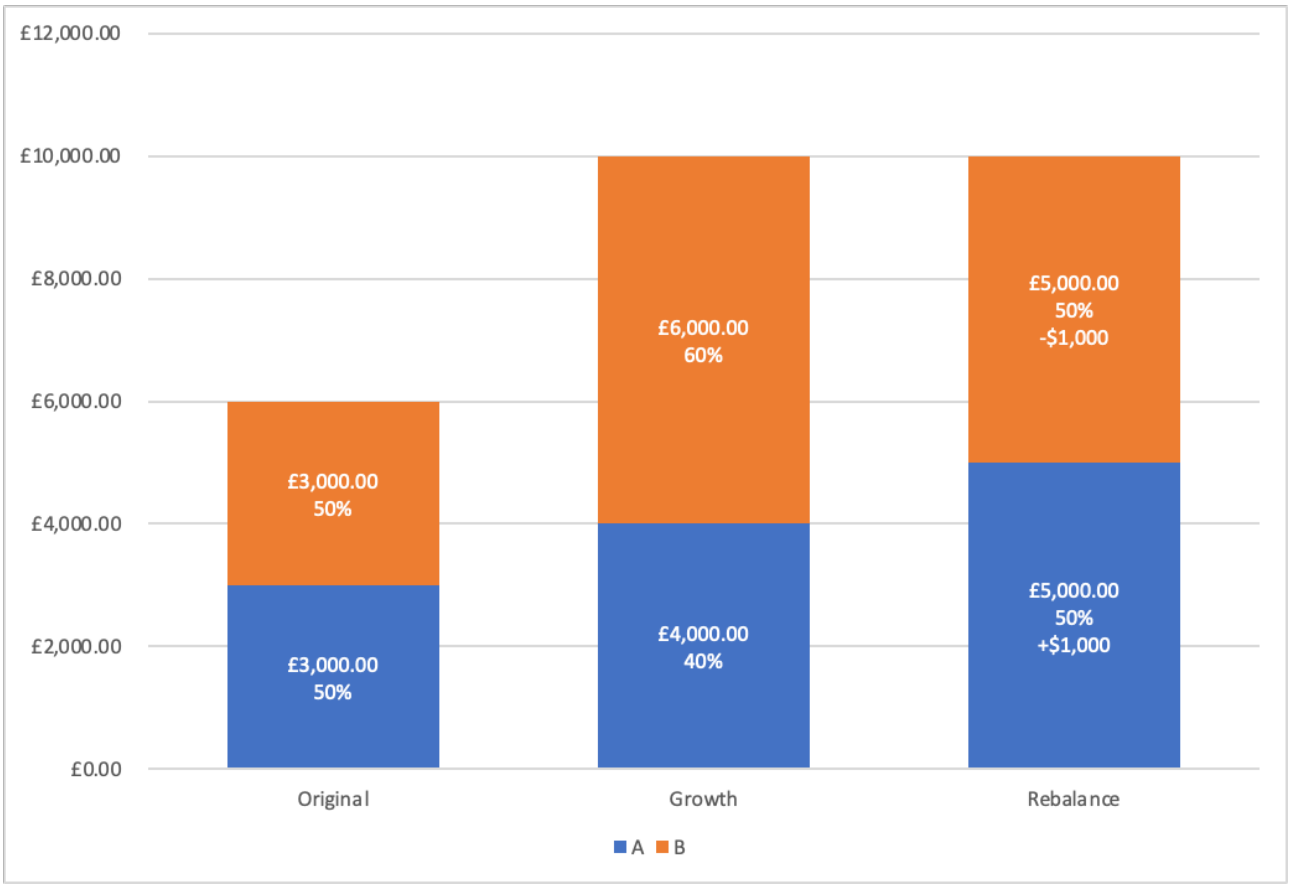


Figure 1: Example portfolio rebalance after growth

1.3. Diversification in Crypto

Crypto has grown to such a size that it has its own emerging sub-classes, which is significant because asset class, rather than individual assets, has been shown to be the main determinant in portfolio performance [9]. Some emerging sub-classes in crypto are decentralised finance (DeFi), ERC-20 tokens (built on Ethereum) and tokens included in games (GameFi). It may be too early to consider any particular sub-class to be entirely uncorrelated from the crypto market at large. However, as the market continues to grow, the independence of sub-classes will become more established and diversification increasingly effective, because a low, or zero, correlation between asset classes is required for successful diversification [10].

There are a couple of distinct challenges facing an investor when looking to diversify within the cryptoasset sector. First, since crypto's sub-classes are still emerging, it can be difficult to define them accurately, let alone represent them in a portfolio. The exchange FTX has been one of the first to begin formalising indices that track certain asset classes. However, among other issues, the investor has no input regarding the index's constituent assets and the range is rather limited. Second is the large number of available crypto exchanges, and the reasons an investor might use more than one. Assets being spread over multiple exchanges majorly increases the complexity of rebalancing; an issue that this report terms the 'multiple exchange problem'.

1.3.1. Multiple Exchange Problem

There are two main reasons for the use of multiple exchanges: asset availability and security. Regarding availability, multiple exchanges are often required to acquire the full range of desired assets due to the different offerings on each exchange. For example, some focus on well-established assets whereas others specialise in lesser-known or newly-created assets which are available on fewer exchanges. Security-wise, storing assets on exchanges poses a 'counterparty' risk, as exchanges are exposed to numerous threats [11]. An investor might mitigate this risk by holding smaller amounts on a larger number of exchanges: another form of diversification itself.

A rebalance consists essentially of the selling of some assets followed by the buying of others. Funds being divided amongst several exchanges makes management of available cash 'liquidity' during the selling and buying a complicated task. Often, an additional transaction (a buy, sell or inter-exchange transfer) is required, which is not the case when funds are on one exchange. In some cases, an investor may not have enough cash or assets on one exchange to complete a portion of selling or buying. Similarly, the transfer of funds between exchanges may be required due to the limited availability of certain assets.

Since each transaction incurs a cost, and the number of transactions is variable, rebalancing across multiple exchanges can be framed as an optimisation problem. A valid solution is a series of transactions that results in a rebalanced portfolio and the best solutions have the lowest number of transactions, especially inter-exchange transfers which are the most costly by a significant margin. The problem can be formulated as a bin-packing problem with several variations, in the NP complexity class [12], and a central focus of this project is the approach taken to its solution.

1.4. Project Intention

Diversification offers a similar benefit when applied to crypto as it does to traditional assets [13]. The digital nature of the assets and the prevalence of API offerings from exchanges means that the sector particularly lends itself to interaction via software. This project looks to facilitate the diversification and rebalancing of cryptoassets across multiple exchanges by way of a mobile application: BalanceBot.

The app requires a foundational set of features to collect the necessary data before attempting to rebalance. It must use exchange APIs to fetch the user's asset balances and corresponding prices to calculate an overall valuation of their portfolio. The app must also allow the user to specify asset 'groups', as asset class is more important for diversification than the individual assets. The user should then specify their desired portfolio percentage allocations to asset groups or, if they choose, individual assets. Once these features are built, the project's next objective is calculating the transactions needed for the rebalance of the user's assets to meet the inputted target allocation, i.e, producing a strong solution to the multiple exchange problem.

1.5. Related Work

1.5.1. Portfolio Trackers

Portfolio trackers are lightweight mobile apps that read balance and price data from exchanges to display the user's overall portfolio valuation, much the same as part of the functionality needed in the proposed application. However, they stop short of allowing the user to input data or performing any action such as calculating a portfolio rebalance. Two examples of such apps are Blockfolio and Delta, almost identical in design and function.

When using these apps, the user first connects to an account on at least one exchange using a generated API key pair. Once the users have inputted their API keys, the app reads their balances and populates the dashboard. The main page, nearly identical in both apps (Figure 2), shows the total portfolio balance and a breakdown of its constituent assets.

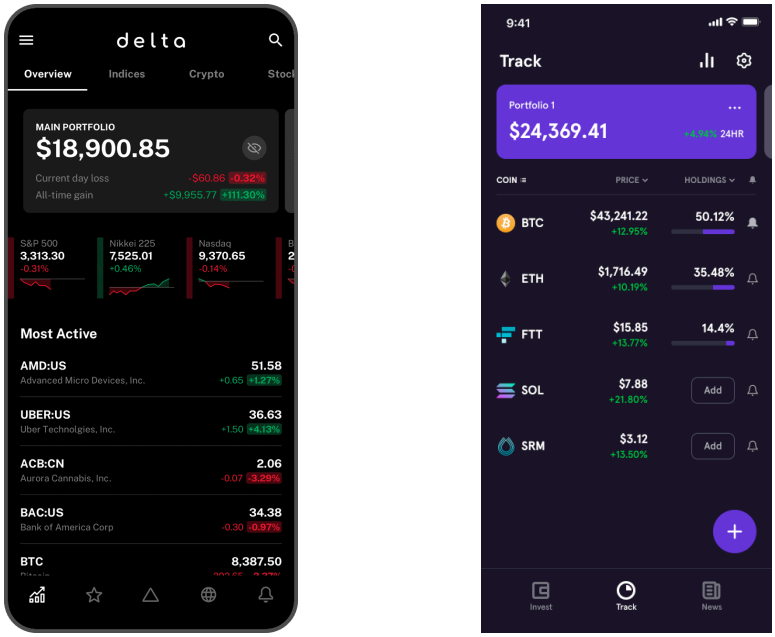


Figure 2: Delta and Blockfolio main pages

Some additional features are included by one or both apps, such as the inclusion of non-crypto assets on Delta. However, it is this core functionality of authenticating with multiple exchange APIs and subsequently requesting the relevant data that is pertinent to the proposed app. Not only is the aggregated data useful to the user for the monitoring of their portfolio, it is also a prerequisite to performing the rebalance calculation.

1.5.2. Portfolio Managers

Where portfolio trackers only read data from a user's multiple exchange accounts, portfolio managers extend this functionality to include placing orders to buy and sell assets on the user's behalf. Two examples of these apps are Shrimpy and 3Commas. Like Delta and Blockfolio, their features overlap significantly.

These examples both include a portfolio rebalancing feature but there are several aspects that this project seeks to improve. The principal issue with both 3Commas and Shrimpy is their aim towards more sophisticated investors. They offer tools such as ongoing trade automation and 'slippage management' which are only desirable to investors with high trading volume and experience implementing advanced strategies. As a result they are complex and largely inaccessible to the lay-investor. Appendix A shows the complexity of the 3Commas interface. Rebalancing is also inaccessible on either platform without a premium subscription.

As crypto grows as an investment sector it is important that it is accessible to regular 'retail'. This project intends to contribute to this end and improve on issues with the aforementioned apps by making a free, single-purpose tool. Although there is benefit in having numerous features, many are superfluous to the casual investor. Focus on a single function will make the app simple to use and reduce the difficulty of applying diversification to crypto. The two existing examples are also designed primarily for a desktop web browser. A native mobile app removes the necessity of a desktop computer for a full experience.

2. Mobile Application

2.1. Specification

The features required in the app can be reverse-engineered from the data needed to produce a solution for the multiple exchange problem: the user's asset balances on each connected exchange, their dollar values, the user-defined asset groups, and the user's specified target allocation. It is simplest to view the features as a direct progression towards the rebalancing implementation, each feature contributing to gathering the required data. The key features of the app are broken down into several stages in Table 1.

Stage		Requirements
1	Authentication and Database	Authentication of the user on the device
		Ability to read and write data to database if authenticated
		Store data at differing levels of privacy e.g. API keys fully private, portfolio strategy accessible from server
2	Connecting to Exchanges	API key input and private storage
		Read account balances from each exchange
		Fetch prices of assets with a non-zero balance
		Aggregate balances and prices to display portfolio summary
3	Strategy Input	Aggregate a list of tickers that are available on the user's connected exchanges
		Allow the user to create 'asset groups' of multiple assets to be treated as one
		Allow the user to allocate percentages of their portfolio to asset groups or individual assets
4	Rebalancing Implementation	Option to calculate the rebalance if all the required data has been fetched
		Implement algorithm to calculate rebalance transactions
		Clear display of the transactions
5	Notifications	Ability in the mobile app to input which strategy is to be followed
		Facility to receive remote notifications within the app
		Companion server that monitors portfolios and sends push notifications to mobile app

Table 1: Required Features

The stages defined in the table are sequential and each builds upon the prior. The authentication and database are foundational to the app, facilitating private, persistent storage of data. Then the app can begin connecting to exchanges and reading relevant data. At this point the app has functionality equivalent to the portfolio tracker applications mentioned previously. The next two stages regard the user's specified target portfolio and calculating the rebalance required to reach it. This completes the core functionality as it can now successfully calculate a rebalance. In stage

five however, notifications are included so the user is sent a reminder at the appropriate time to rebalance according to their strategy. Each stage of the app is detailed later in its own section.

A critical aspect of the app is being highly accessible to the casual investor, achieved by making all the data retrieval and input as straightforward and seamless as possible. The app should also feel natural and fit in with the ecosystem that surrounds it. The app will be built for Apple iOS and should feel natural and in-tune with the ecosystem that surrounds it. A reference for this will be Apple's iOS Human Interface Guidelines [14].

2.2. Architecture

The overall solution consists of three main parts: a mobile application, a companion server for notifications and a system for authenticating and storing data. The app is responsible for receiving user input, connecting to exchanges and reading from and writing to the database. All the essential functionality exists within it and thus it is the centrepoint of the solution. Consequently, it should function without reliance on the notification server, and should therefore be separate from the authentication and database system used by the app.

The chosen database (explored in Section 2.3) enables precisely this, allowing connections from the notification server such that the app and server never need to communicate directly. This not only ensures the app's independence from the server but also reduces the number of integrations needed in the development of the app, which is already large due to the many connections to exchanges. Interactions between different parts of the solution are outlined in Figure 3 and detailed in Table 2.

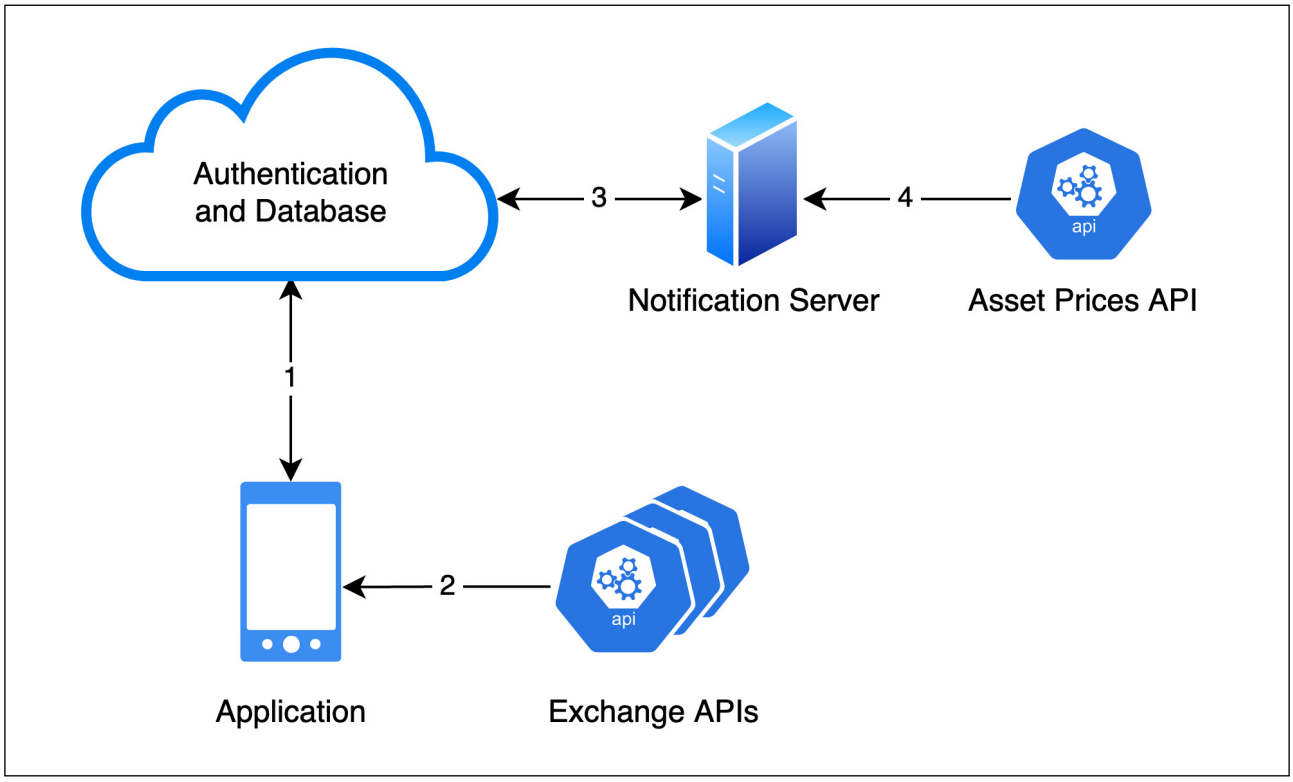


Figure 3: Intra-solution interactions

	Interaction	Purpose
1	Application reads and writes to database	Authentication, storing API keys and portfolio preferences, storing fetched asset balances, receiving notifications
2	Application reads from exchange APIs	Fetching user asset balance data, asset prices and available tickers
3	Notification server reads and writes to database	Monitoring asset balances, writing notifications
4	Notification server reads from asset prices API	Fetching asset prices to calculate current portfolio value and allocation percentages

Table 2: Intra-solution interaction details

2.2.1. iOS Application

The app will be written for iOS devices in Swift using Apple's Xcode, primarily because of existing personal proficiency. Since the second part of the project (implementing a rebalancing algorithm) is a significant task, it proved time efficient to avoid the need to become familiar with a new language during the development of the app. Developing solely for iOS devices has the additional benefit of drastically reducing the number of target devices and screen sizes, resulting in less time spent developing undue cross-compatibility. This does, however, mean the resulting product will run on a smaller range of devices. Given the time constraints and scope of this project this was deemed acceptable but is noted as a point for future improvement.

Apps are currently developed for iOS using SwiftUI, a declarative programming paradigm. In SwiftUI the app's interface responds to changes in the app's 'state' i.e. the data it currently has. When programming in this manner it is useful to have clear architecture and methodology to follow to prevent the codebase from bloating. Clean Architecture (CA) for Swift [15] has been implemented for the iOS portion of this project. Four standalone, generic helper classes from the example repository [16] were directly implemented to aid development. The architecture is split into four main classes across three layers, shown in Figure 4.

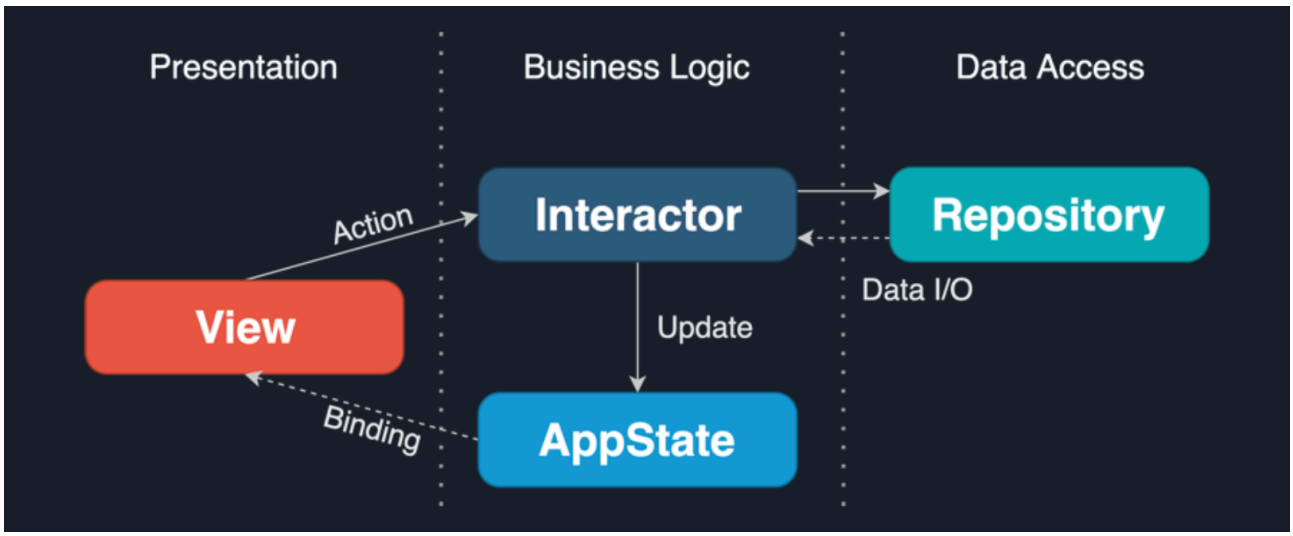


Figure 4: Clean Architecture for Swift
https://nalexn.github.io/clean-architecture-swiftui

The ‘presentation’ layer provides the app’s user interface. Objects within it are all of the class ‘View’, displaying information to the user and receiving input. In the ‘business logic’ layer, ‘Interactor’ objects perform user-initiated actions and handle fetched data, e.g. calculating the total portfolio value. ‘AppState’ is a singleton class which stores the app’s data and is updated by various interactor objects. Finally, the app connects to data providers such as the CloudKit database and exchange APIs in the ‘data access’ layer using ‘Repository’ objects. The distinction between interactors and repositories is that the latter do not perform any processing or logic while the former do not connect to any external services.

This architecture has clear separation of concerns between layers and classes, something the originator of the architecture has written about [17], which is essential for high quality software development. There are alternative architectures that are appropriate for SwiftUI, such as MVVM [18]. However, CA was chosen for its flexibility in connecting the presentation and business logic layer. MVVM requires one ‘ViewModel’ for each view whereas CA allows the use of the same interactor for multiple views, which is useful for the application in this project.

2.3. Authentication and Database

The authentication and database system chosen for the app was required to authenticate users and store data such that it is inaccessible to any user other than the data owner. The focus of the database is therefore privacy and security instead of scalability, as there is not a large quantity of stored data, nor should there be a high volume of requests. The ‘login’ component is also as seamless as possible because user accounts are not a feature of the app other than as a means to store data privately. This stage of the app’s development constitutes the infrastructure upon which the remaining features are built.

The app in this project uses Apple’s CloudKit service for both authentication and data storage. Made by Apple, it is designed to be easy utilised in iOS apps using Xcode, making it especially appropriate for this project. As well as native integration with Xcode, CloudKit also offers a Javascript API, allowing database access from the notification server. Finally, CloudKit also offers the simplest method by far of sending remote push notifications to the app, again due to the close integration of services in the Apple ecosystem.

Authentication with CloudKit is completely automatic using the iCloud account on the iOS device running the application, saving the need for a ‘traditional’ account with a username and password. Unless they have specifically sought otherwise all iOS users have a corresponding iCloud account, meaning all they must do to authenticate is launch the app.

CloudKit offers another highly useful feature when storing data: a separation of what it calls ‘public’ and ‘private’ data. Public data is technically readable to any user of the application. However, this is not the case in practice as the user has no way of querying the data other than those presented in the app, and the app in this project never accesses the database in such a manner (e.g. search). The data can however be read by an administrator account, meaning the notification server can monitor portfolio data without it being accessible to any unauthorised users. Private data has an entirely different level of access. It is stored in the user’s own enclave of the database and is viewable to no one else, not even the database owner, which is perfect for sensitive data such as the user’s API keys.

Data in the CloudKit database is divided into two models: Account and Portfolio. Account contains sensitive API keys and is therefore stored in the private database. Portfolio records are stored in the ‘public’ database so they can be read by the server. Although there is no way for an entity other than the notification server to query portfolio records, they are anonymised with a random ID to eliminate any risk to privacy. The two records are connected by the portfolio ID, used as foreign key in the account model. Figure 5 shows the class diagram.

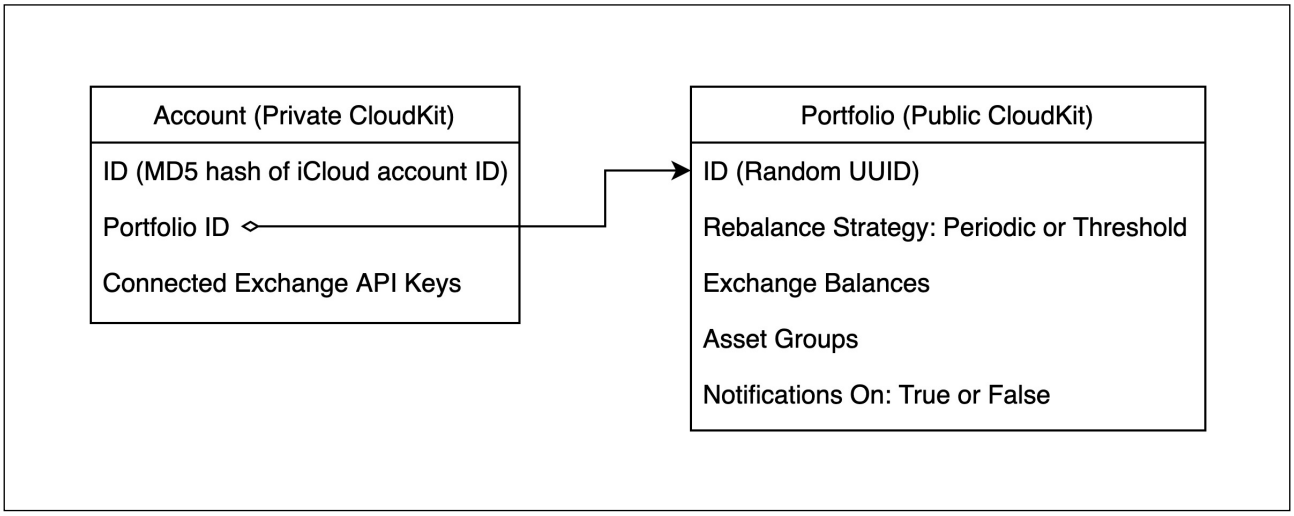


Figure 5: App and CloudKit data models

Note: to fit CloudKit schema, complex fields such as 'Exchange Balances' are strings of JSON data

When the app launches it immediately determines the authentication state. If the user for some reason doesn't have an iCloud account, it shows a prompt to enable one. Although currently a dead-end, proper handling of this scenario is another noted future improvement. Otherwise, the app attempts to retrieve the account record from the private database using the MD5 of the user's iCloud account ID. If it exists, the app uses the portfolio ID field from the account record to fetch the portfolio record from the public database. If no data is in CloudKit under that hash, the app generates account (with hash as ID) and portfolio records, connected via foreign key, and saves them in the respective databases.

2.4. Connecting to Exchanges

Once a user is authenticated and able to store data, the next step is connecting at least one exchange account and fetching their portfolio data. On each supported exchange, they are able to generate an API key pair consisting of a key and secret. The app enables the input of these values and stores them in the user's private CloudKit database. The user can also remove the API key for an exchange, in which case the key is also removed from CloudKit. The list of exchanges and API key input is shown in Figures 6 and 7.

Some exchanges also produce a QR code in addition to the normal pair of text strings when the API key pair is created. In these cases the app offers a scanning interface, Figure 8, so the key and secret do not need to be copied manually. This is implemented in the app using the CodeScanner library [19]. When a QR code is scanned using this library, a string of the QR code's contents is returned. The app decodes this into key and secret strings using regular expression, accounting for each exchange's unique format. It then auto-populates the respective text fields and submits the key pairs to be saved in the private database.

After one or more API key pair has been inputted, the app retrieves the user's asset balances using the authenticated API for each connected exchange. Requests are verified using an HMAC which is generated using a hash function, the user's API key and secret, and a message containing details of the request: the body, method, path and a nonce timestamp. Due to differing implementations between exchanges, there are multiple methods of generating the HMAC signature. The main differences are the hashing function, the construction of the message and the encoding of the input and output strings. Table 3 details the HMAC variations handled by the app, which supports four exchanges: Bitfinex, Kraken, Coinbase and FTX.

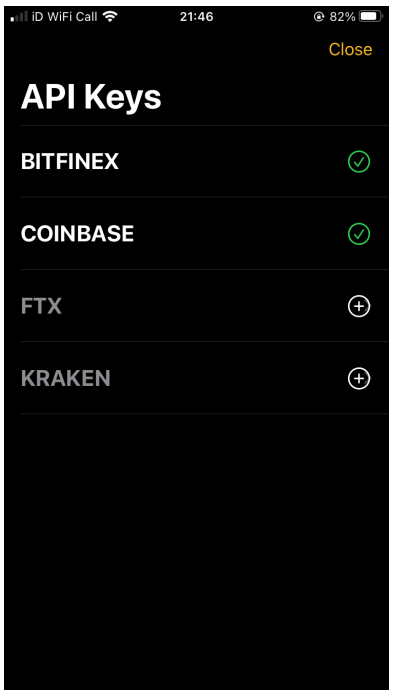


Figure 6: Exchange lists

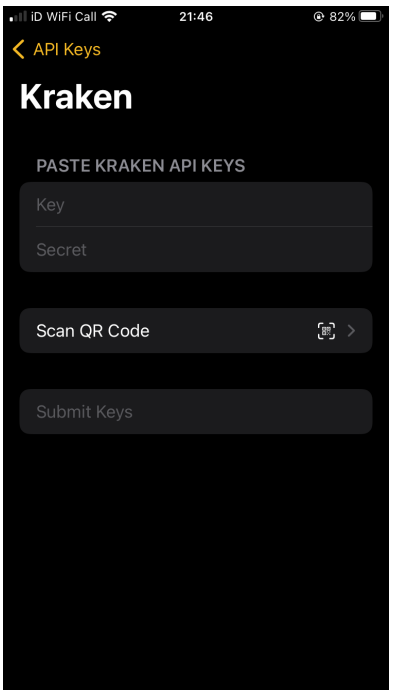


Figure 7: API key input

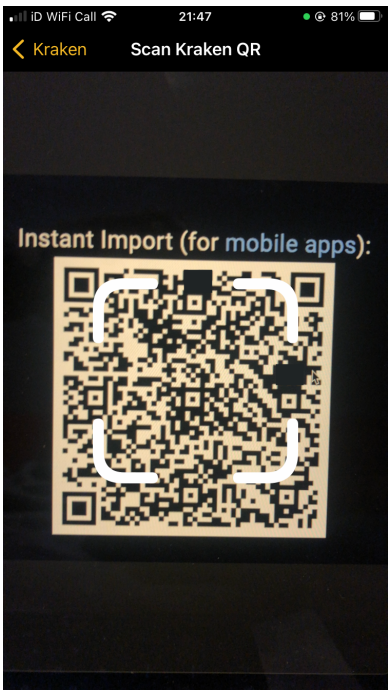


Figure 8: QR scanner

Component		Exchange			
		Bitfinex	**Kraken**	**Coinbase**	**FTX**
Hash Version		SHA384	SHA512	SHA256	SHA256
Message Construction		{path} + {nonce} + {body}	$_{\text{UTF8}}$({path}) + $_{\text{SHA256}}$({nonce} + {body})	{nonce} + {method} + {path} + {body}	{nonce} + {method} + {path}
Signature Encoding		Hexadecimal	Base64	Hexadecimal	Hexadecimal
Parameter Names	**Nonce**	bfx-nonce	N/A	CB-ACCESS-TIMESTAMP	FTX-TS
	API Key	bfx-apikey	API-Key	CB-ACCESS-KEY	FTX-KEY
	Signature	bfx-signature	API-Sign	CB-ACCESS-SIGN	FTX-SIGN

Table 3: Authenticated API request signing variations

Note: the '+' symbol denotes concatenation

This first request returns balances without corresponding prices. To calculate the value of the assets it requires the price of each asset in the user's portfolio, which can be fetched from the exchange's public API, as opposed to authenticated one used for requesting balances. The price data from the second request is combined with the asset balance data to form a summary of the user's holdings and their values. This summary is displayed to the user, Figure 9 and pushed to the 'public' CloudKit database so it can be read from the notification server. Some exchanges, such as FTX, provide the dollar value along with the balances, in which case the app omits the additional, separate request. Once aggregated, the summary is presented to the user in a design akin to the Delta or Blockfolio apps in the Related Work section.

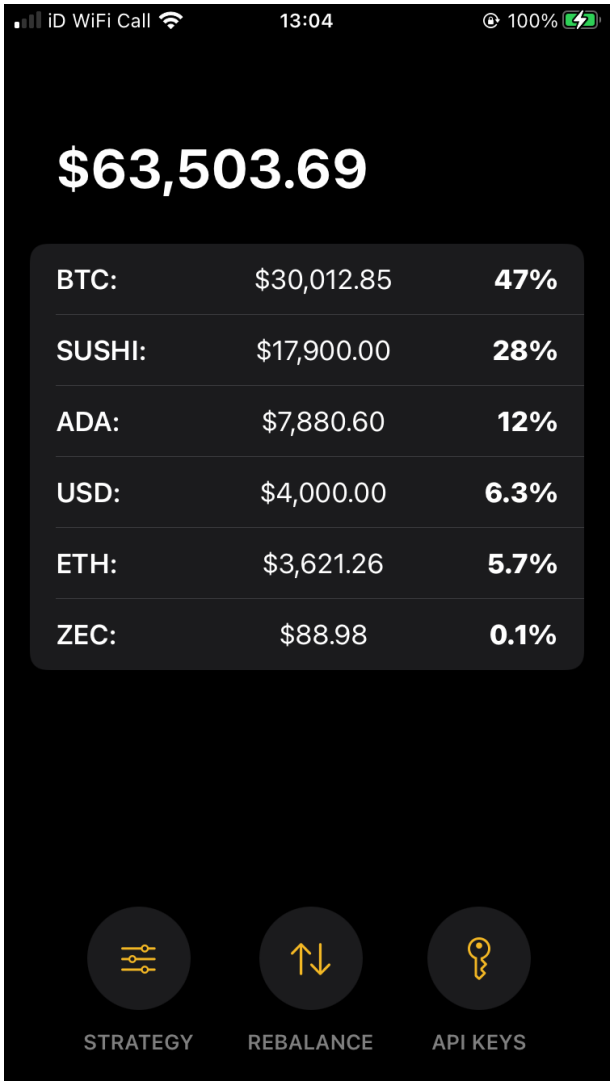


Figure 9: Portfolio dashboard

There are two caveats the app overcomes when handling with data fetched from exchanges. First, the name of the assets, ‘tickers’, are formatted slightly differently on each exchange. The app uses regular expression and string interpolation to transform tickers to and from the format of each exchange. On some exchanges some tickers have entirely different names, for example, “BTC” is “XBT” on Kraken. For this the app maintains a map of asset renames for each exchange. Secondly, as a result of transactions, minuscule quantities of assets (‘dust’ balances) are often undesirably included in a portfolio breakdown. The app uses a simple filter to remove balances under $5 but a customisable threshold is a noted future improvement.

2.4.1. JSON Decoding

The format of the API’s JSON response is differs between exchanges. The app employs a custom solution to decode the received JSON to an array of initialised objects of internal classes. The decoding is done in the app using a recursive, ’indirect’, enumeration to describe the schema of the JSON and select relevant information. It allows value selection via dictionary key or array index, as well ‘repeat unit’ which is used to produce the array of objects of the local class. For example, the first decode the app performs is decoding from JSON, Figure 10, to an internal class named ‘ExchangeBalance’, shown in Figure 11

```
{
  "pagination": { ... },
  "data": [
    {
      "id": "58542935-67b5-56e1-a3f9-42686e07fa40",
      "name": "My Vault",
      "balance": {
        "amount": "4.00000000",
        "currency": "BTC"
      },
      "created_at": "2015-01-31T20:49:02Z",
      ...,
      ...
    }, { ... }, ...
  ]
}
```


Figure 10: Example JSON response from Coinbase API

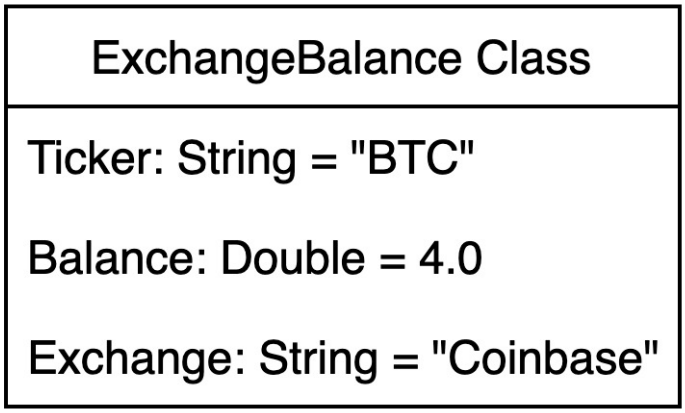


Figure 11: Internal ‘ExchangeBalance’ class

Note: values based on example Coinbase JSON

The ‘repeat unit’ of the JSON in this example is the “data” array, containing elements that become each ExchangeBalance object. The value for “Ticker” (“BTC”) is selected by unwrapping the “balance” object and selecting the value for key “currency”, the value for the “Balance” attribute is selected in the same way, except with key “amount”. The use of a recursive enumeration allows the mapping of any JSON structure to an array of any object used within the app as long as it conforms to the ‘ArrayInitialised’ protocol. This custom protocol adds initialisers so that the conforming object can be initialised using an array of attributes. This powerful feature could be packaged in its own right as a useful library for Swift JSON decoding.

2.5. Strategy Input

Building towards the objective of calculating a rebalance, the app has a ‘Strategy’ section to collect the user’s portfolio preferences. This includes the rebalance trigger selection (periodic or threshold), asset group definition, portfolio percentage allocation and a toggle for notifications.

When the user opens this section of the app, Figure 12, they can allocate a percentage of their portfolio to any asset that is available on one or more of the connect exchanges. For example, 60% Bitcoin, 20% Ethereum, 20% Litecoin. The app verifies that the total of the user’s allocation is exactly equal 100%, in which case offering the option to save the allocation. Upon saving, the target allocation is stored in the public CloudKit database alongside the summary of their holdings so it can be accessed by the notification server. The user can also select the portfolio rebalance trigger, with different value options depending on the selected type, subsequently updated in the public database. Figure 13 shows the options when selecting a periodic trigger.

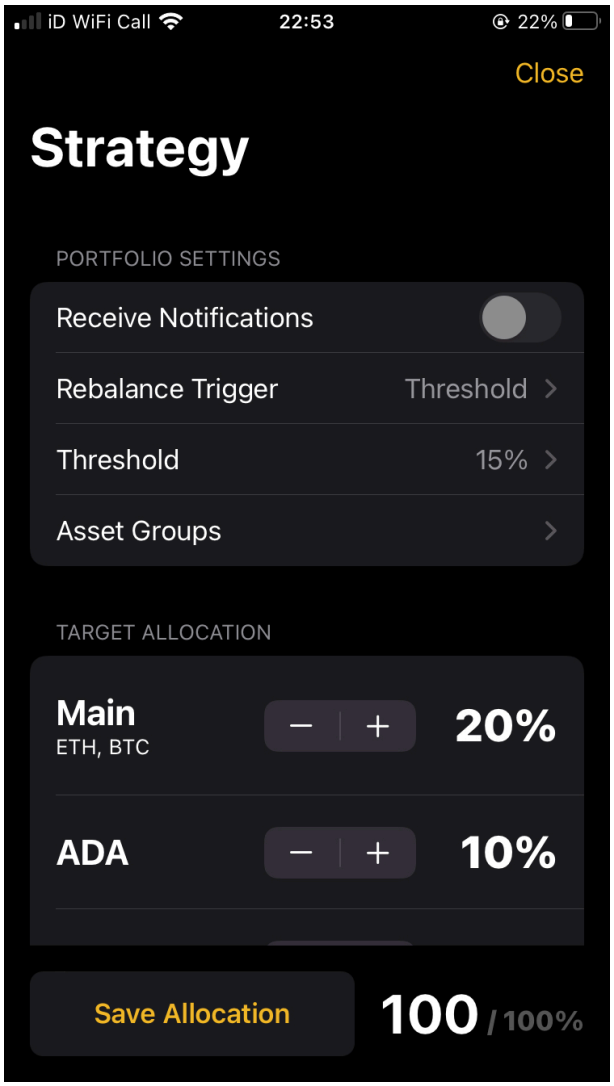


Figure 12: Strategy dashboard

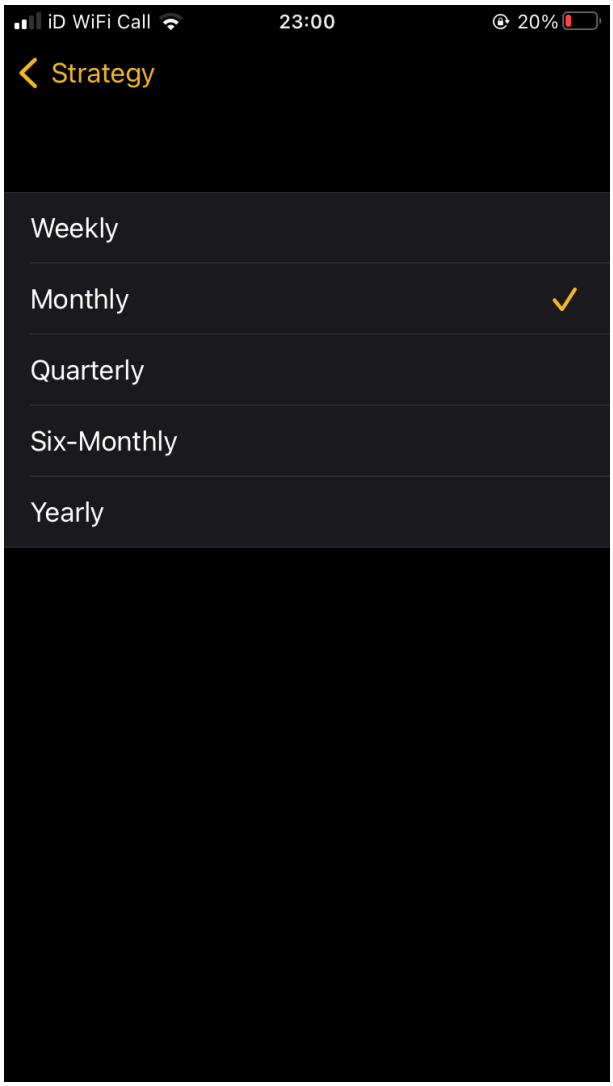


Figure 13: Options for periodic rebalance trigger

In the introduction to this report, it was mentioned that that asset sectors, as opposed to individual assets, were the main determinant in portfolio performance. Consequently, given the same list of available assets, the user can define ‘asset groups’ to which percentages of their portfolio can be allocated. For example, an asset group named ‘Blue Chip’, containing Bitcoin, Ethereum and Litecoin. Once defined, the asset groups are displayed at the top of the list of assets when the user chooses their allocation. The user could now choose an allocation of 40% ‘Blue Chip’, 30% Zcash and 30% Monero. This allocation contains one asset group (‘Blue Chip’, which contains three assets) and two individual assets (Zcash and Monero). At this point, the app is able to gather and store all the data necessary for the implementation of an algorithm to calculate a rebalance. Figures 14 an 15 show the interface for creating and editing asset groups.

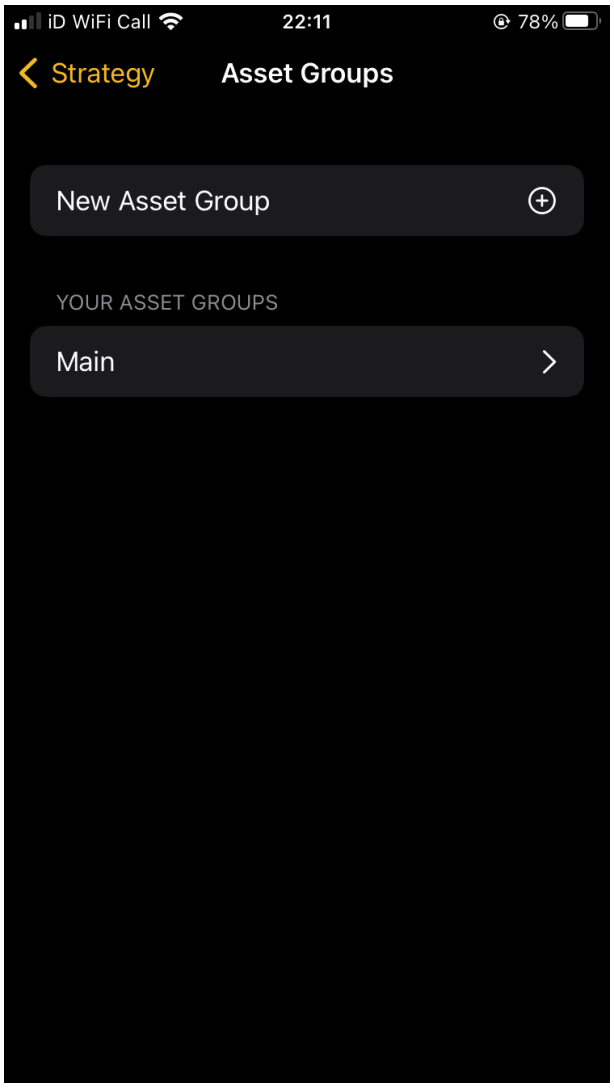


Figure 14: List of asset groups

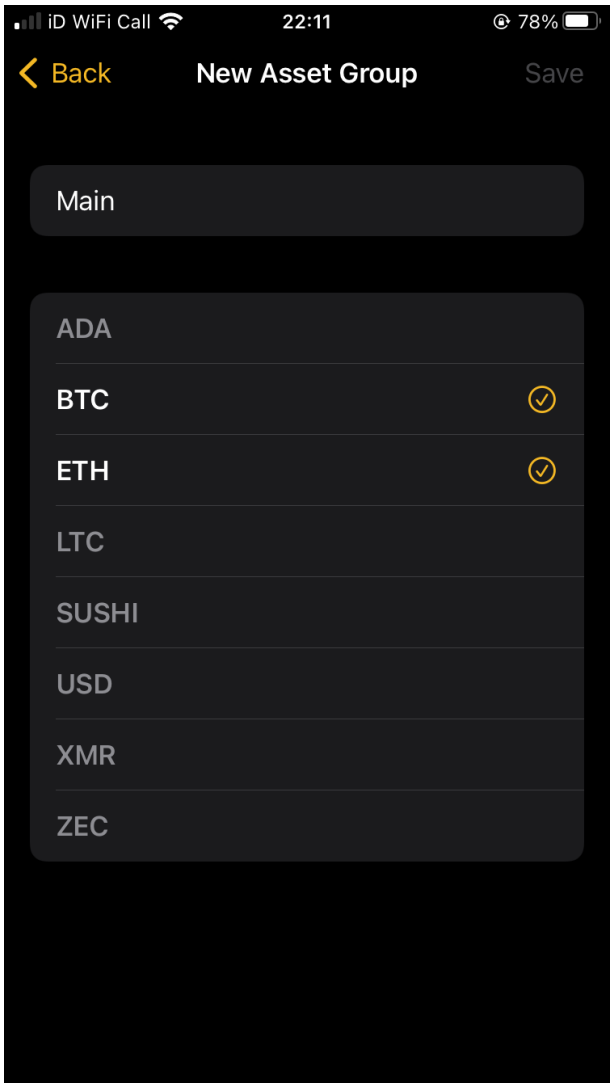


Figure 15: Asset group constituent selection

2.6. Rebalancing Implementation

If the user has connected at least one exchange and has inputted their target portfolio allocation the app offers the option to calculate a portfolio rebalance. When initiated by the user, the algorithm runs and calculates the necessary steps to rebalance the portfolio. Once the steps have been determined they are shown to the user, as in Figure 16. This completes the app's core functionality, from onboarding a user to calculating a rebalance across multiple exchanges. The details of the algorithm used to produce the rebalancing steps is detailed later in its own section.

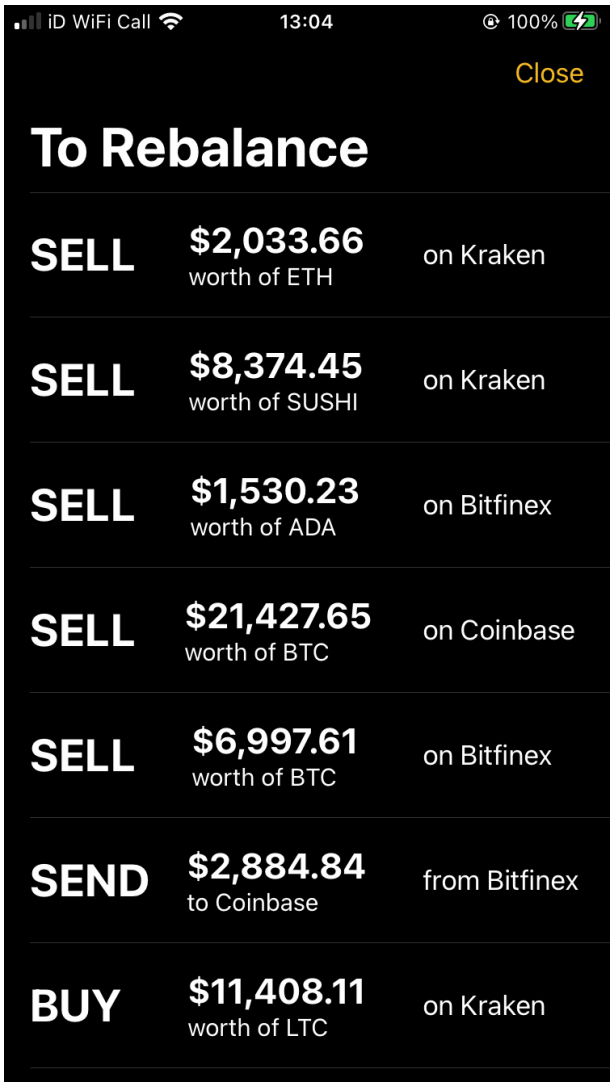


Figure 16: Transactions for rebalance

2.7. Notifications

The ‘Strategy’ section of the app allows the user to select which rebalance trigger to use, and whether they would like to receive reminder notifications. When creating the user and portfolio models, in Authentication and Database, the app also creates a notification subscription to a third kind of model named ‘Notification’. These records simply contain the ID of the associated portfolio so notifications are sent to the correct user. A separately-built notification server reads portfolio data from CloudKit, performs calculations and logic, then, if required, writes a notification record to be automatically forwarded to the app.

Since CloudKit’s offers a web API written in Javascript (JS), the server is written using NodeJS, a popular, general-purpose JS environment. It integrates easily with the CloudKit JS API and has an extensive library of packages that can be installed to support many other necessary functions. Due to the simplicity of both the server’s function and NodeJS itself the server does not require any specific architectural structure beyond a single file. The server’s access to the public CloudKit database is authenticated with a ‘server-to-server key’. These keys can only be generated by the database owner on the CloudKit web dashboard and do not grant access to any ‘private’ data.

Depending on the strategy, the server should either be regularly calculating whether a portfolio’s allocation has deviated beyond the specified threshold or keeping track of the periods between rebalances. When a rebalance is required by either strategy, and if the user has opted for notifications, the server writes a notification record to CloudKit which is received by the mobile app, prompting the user to launch the app and calculate a rebalance. To calculate whether a portfolio is beyond the threshold the server requires asset price data. Instead of drawing it from several different exchanges (as the app does) it uses the CoinGecko API [20] NodeJS package to get all the information from one source. CoinGecko is a crypto market tracker website with almost every asset in the sector listed.

In its current implementation, the server’s main script does the following. First, it reads all portfolios in the public CloudKit database, filtered to those that have opted for notifications. It then performs the threshold calculation for portfolios with the threshold rebalance trigger. For each of these portfolios, if the threshold has been violated, the script writes a notification to CloudKit with the corresponding portfolio ID. The server does not currently implement ongoing portfolio monitoring, although notifications have been successfully received by the iOS device associated with that portfolio ID, showing that the server is complete in the rest of its functionality.

3. Multiple Exchange Problem

3.1. Problem Formulation

3.1.1. Supermarket Analogy

The second portion of this project focuses on determining and minimising the steps needed to rebalance a portfolio when the user's assets are spread across several exchanges, i.e. producing strong solutions to multiple exchange problem (MEP) defined in the introduction of this report. Visualising the problem is quite difficult due to the number of constraints that must be observed and the number of possibilities. It can be understood more intuitively using the analogy of supermarkets, products and gift cards, based on the following scenario.

- There are two supermarkets, A and B.

- Both stock certain products
  - Many of the products are stocked in both supermarkets, if they are they have the same price
  - Some may only be available at supermarket A and others at B.

- A small cost is incurred for repeating purchases of products in both supermarkets.
  - e.g. buying a single loaf of bread at supermarket A and another at B is less efficient than buying two loaves at one supermarket.
  - This cost is smaller than the cost incurred for transferring credit between supermarkets.

- Both offer gift cards which can only be spent at the respective supermarket.
  - The supermarket only allows purchases using its gift card i.e. there is no cash or card.
  - It is possible to transfer credit from one supermarket to another but at significant cost.

With this context, the optimisation can be formulated:

Given that:
- the shopper has a list of required products, all of which are available at supermarket A and/or B.
- the shopper has enough total credit between supermarkets to buy all the required products.
- the shopper knows the amount of credit on their card for each supermarket.
- the shopper knows the availability of the products at each supermarket.
- a cost is incurred for credit transfers between supermarkets.
- a small cost is incurred for repeating the purchase of products between exchanges.

Which combination of products should the shopper purchase at which supermarkets such that cost is minimised?

There are several situations that the shopper could face, of varying complexity. Figure 17 shows the main high-level questions to be considered to get a sense of a specific problem instance and determine how a solution might be attempted.

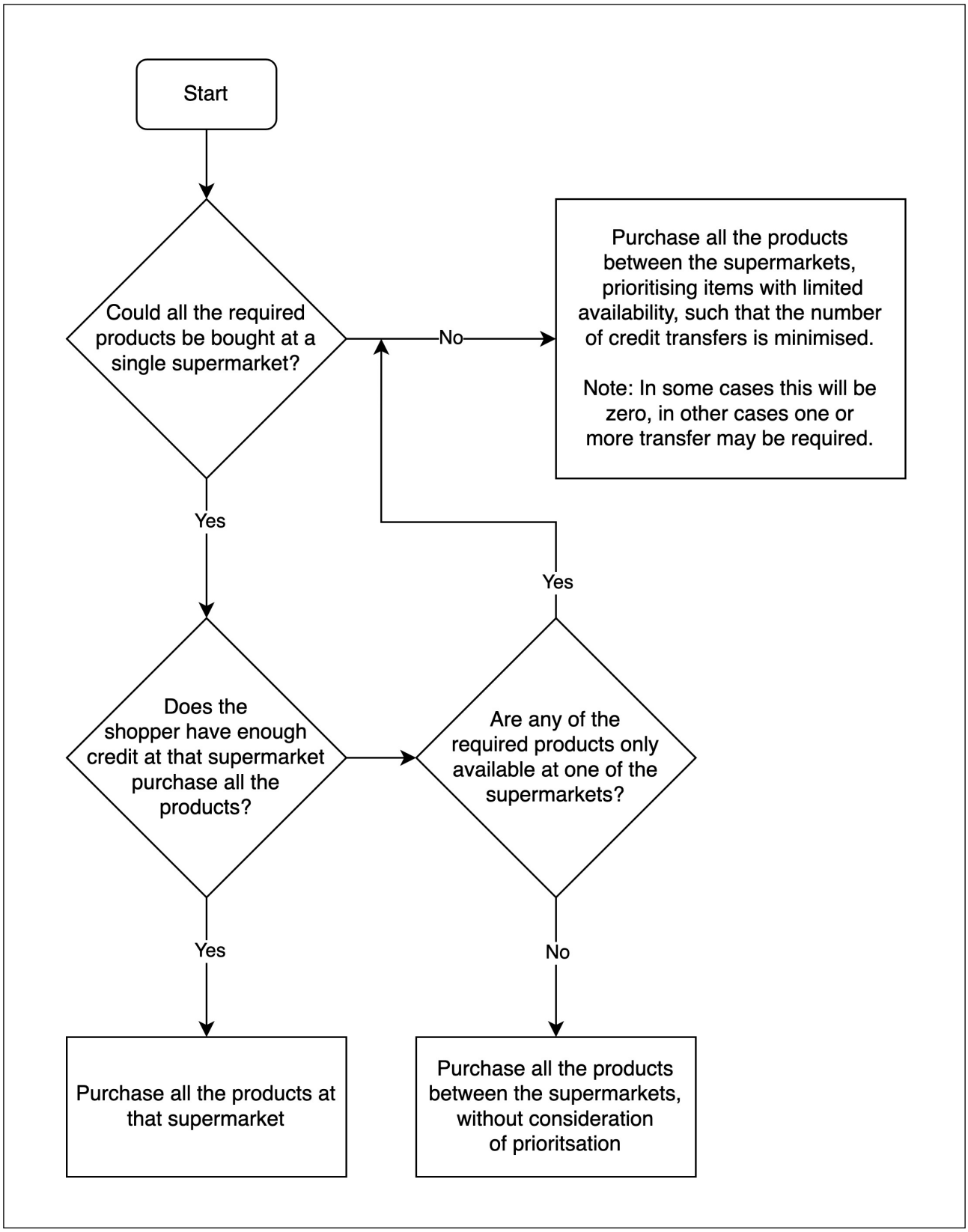


Figure 17: Flow of high-level considerations within supermarket analogy

3.1.2. Bin Packing

In terms of established problems in computer science, the MEP is closest to bin-packing combinatorial optimisation problems, where ‘bins’ are exchanges and ‘items’ are asset. However, several variations must be applied to the general bin packing problem for the comparison to hold.

First, the ‘items’ being considered are unidimensional dollar values as opposed to most traditional bin packing problems which usually feature two or three-dimensional items. The items are fragmentable as dollar values can be broken into smaller amounts. With this variation the minimisation aspect is no longer the number of bins but the number of times items are fragmented [21]. This is applicable, although in the MEP fragmentation is one of several possible transactions to be minimised. Another variation is item-bin compatibility. For example, one example of the bin packing problem features items that have to go in specific refrigerated bins whereas other can go in any [22]. In the context of the MEP, item-bin compatibility is asset-exchange compatibility: exchanges and assets are compatible if the asset is available on the exchange. Next, since assets are already on exchanges in one combination or another, the bin packing problem can be said to have ‘pre-placed’ items, see [23]. Finally, the deltas required for the rebalance must be satisfied, representing an additional constraint that must be upheld, as a solution is only valid if it makes the necessary changes to asset balances.

3.1.3. In Context

Calculating the changes necessary for each asset in a portfolio to be rebalanced is as simple as finding the difference between its current value and the value it would be if it were the correct proportion. The change needed, ‘delta’, to reach this target can be considered the buying or selling of a given asset. For example, a delta of -$100 for bitcoin would mean $100 worth of bitcoin needs to be sold to be rebalanced. When holdings are on one exchange, enacting these changes is trivial. Simply sell the required amount of any asset with a negative delta before buying the required amount of assets with a positive delta; each delta only needs a single buy or sell transaction. However, this is often not the case when holdings are distributed across exchanges. The multiple exchange problem comes down to finding the most efficient way to enact the deltas needed to set the portfolio to its target allocation.

Some deltas are negative, stating that a quantity of the corresponding asset needs to be sold, and some are positive, meaning a quantity of that asset must be bought. It is important to perform the required selling to free up funds before attempting the required buying. With the supermarket analogy for reference, deltas represent two things. Negative deltas represent potential credit (gift cards) in an exchange. It is considered ‘potential’ because the investor may have facility to realise credit on multiple exchanges. Only when the investor sells a quantity of an asset on an exchange does it become realised. On the other hand, positive deltas correspond with the shopping list of products that the shopper has to buy as the investor must buy assets with a positive delta.

Each transaction (buy, sell or transfer between exchanges) incurs a cost, and the overall cost of the rebalance can be optimised. The aim of a solution would be to produce a series of transactions that satisfies the required deltas, whilst minimising the number of transactions, especially transfers between exchanges as these are more costly and time consuming than a buy or sell transaction. The problem can be reformulated in the real-world context:

Given:
- The deltas required for a rebalance and their corresponding asset
- The investor’s current holdings on exchanges
- The availability of assets on exchanges

Calculate a series of transactions such that:

- The deltas are satisfied
- The number of transactions, then inter-exchange transfers, is minimised

Although the minimisation objective in the problem formulation is two-fold, the two minimisations are not inherently in conflict with each other. The nature of the problem means that there are many solutions that share the same optimal number of transactions. It then follows that the next minimisation, number of inter-exchange transfers, can be applied on the subset of solutions that already share that minimum number of transactions.

As with the supermarket analogy, there are multiple scenarios an investor could face. There are cases where a transfer of funds between two exchanges is unavoidable due to insufficient 'potential credit' (assets with a large enough negative delta) on that exchange to buy the asset with a positive delta that is only available there. However, sometimes, in certain situations, the investor can avoid having to transfer funds between exchanges by prioritising purchase of assets with limited availability.

Below is an example of three such cases, where the optimal solutions have different minimum levels of cost due to limited asset availability and differing quantities of assets held on different exchanges. Table 4 shows the current and target portfolio allocations, as well as the deltas for each asset to reach its target value in the example rebalance.

Asset	Current Portfolio (Before Rebalance)		Target Portfolio (After Rebalance)		Deltas
	% Allocation	Dollar Value	% Allocation	Dollar Value	
BTC	70%	$7,000	60%	$6,000	-$1,000
ETH	20%	$2,000	25%	$2,500	+$500
LTC	10%	$1,000	15%	$1,500	+$500
Total	100%	$10,000	100%	$10,000	$0

Table 4: Delta calculation for example portfolio rebalance

The deltas are the same in each scenario. It is the distribution of assets across exchanges which affects the need for a transfer in order to fulfil the deltas. The situations involve two exchanges, Kraken and Bitfinex, and three assets, Bitcoin (BTC), Ethereum (ETH) and Litecoin (LTC). In this unrealistic example, all three assets are available on Kraken whereas only BTC and ETH are available on Bitfinex, not LTC. This limited asset availability, combined with the distribution of assets is what affects the minimum transactional cost of the rebalances in each scenario. For reference, the required deltas are: BTC: -$1,000, ETH: +$500, LTC: +$500.

Scenario 1

In the first scenario, the rebalance can be completed by transacting on a single exchange, despite assets being held on multiple. Table 5 shows how the rebalance is achieved, with all the buying and selling done solely on Kraken. The transactions are done on Kraken because one of the assets that needs to be bought is LTC, which is only available on Kraken. If $1,000 of BTC were to be sold on Bitfinex, the necessary ETH could be bought but not the LTC. Since ETH is also available on Kraken, to minimise the number of transactions, Bitfinex is not used at all.

Exchange	Asset Holdings Before Rebalance			Asset Holdings After Rebalance		
	BTC	ETH	LTC	BTC	ETH	LTC
Kraken	$2,000	$1,000	$1,000	**$1,000**	**$1,500**	**$1,500**
Bitfinex	$5,000	$1,000	Not Available	$5,000	$1,000	Not Available
Total	$7,000 (70%)	$2,000 (20%)	$1,000 (10%)	$6,000 (60%)	$2,500 (25%)	$1,500 (15%)

Table 5: Scenario 1 - Exchange holdings before and after rebalance

Note: **bold** values indicate change, red indicates decrease, green indicates increase

Scenario 2

In the second scenario there is not enough BTC 'potential credit' on Kraken ($500) that can be realised to fully execute the rebalance on just that exchange. If the total delta for bitcoin was sold on Bitfinex only, the purchase of LTC would require a transfer of $500 from Bitfinex to Kraken. However, there is enough Bitcoin to sell on Kraken to facilitate the purchase of the LTC, which has limited exchange availability. If the investor 'fragments' the BTC delta into two separate transactions, $500 on Kraken and $500 on Bitfinex, they are able to fulfil the deltas without needing to transfer money between the exchanges, buying LTC on Kraken and ETH on Bitfinex.

Exchange	Asset Holdings Before Rebalance			Asset Holdings After Rebalance		
	BTC	ETH	LTC	BTC	ETH	LTC
Kraken	$500	$1,000	$1,000	**$0**	$1,000	**$1,500**
Bitfinex	$6,500	$1,000	Not Available	**$6,000**	**$1,500**	Not Available
Total	$7,000 (70%)	$2,000 (20%)	$1,000 (10%)	$6,000 (60%)	$2,500 (25%)	$1,500 (15%)

Table 6: Scenario 2 - Exchange holdings before and after rebalance

Scenario 3

In the final scenario there is not enough BTC on Kraken to sell (only $200) to facilitate the buying of all the LTC ($500). Consequently, the transfer of funds between Bitfinex and Kraken is unavoidable. When this is the case it is preferable to keep the amount of money transferred between exchanges to a minimum. The full $1,000 of BTC is sold on Bitfinex as there is no point selling only $200 on Kraken, adding an extra transaction. $500 is then sent from Bitfinex to Kraken where LTC is bought. The remaining $500 on Bitfinex is used to buy ETH.

Exchange	Asset Holdings Before Rebalance			Asset Holdings After Rebalance		
	BTC	ETH	LTC	BTC	ETH	LTC
Kraken	$200	$1,000	$1,000	$200	$1,000	**$1,500***
Bitfinex	$6,800	$1,000	Not Available	**$5,800**	**$1,500**	Not Available
Total	$7,000 (70%)	$2,000 (20%)	$1,000 (10%)	$6,000 (60%)	$2,500 (25%)	$1,500 (15%)

Table 7: Scenario 3 - Exchange holdings before and after rebalance

* Requires inter-exchange transfer

The existence of three distinct possibilities depending on the distribution of assets on exchanges, despite the same rebalance deltas, demonstrates the variability of the MEP. The implemented algorithm is designed to produce optimal solutions regardless of asset distribution.

3.2. Implementation

The developed algorithm draws from the field of Genetic Algorithms (GAs) by generating a population of randomised solutions and selecting for the strongest. Unlike GAs however, the algorithm does not 'evolve' solutions over multiple generations because the optimal solution is almost always contained in the first generation. It can ultimately be considered a sort of 'brute-force' algorithm, as it makes randomised attempts. However, rather than having a specific target at which the algorithm halts once reached, the algorithm generates a certain number of candidate solutions from which the best is selected. The algorithm consists of three steps: calculating the required deltas for the rebalance, producing randomised solutions that fulfil the deltas and selecting the optimal.

3.2.1. Calculating Deltas

As mentioned before, calculating the deltas can be done simply by finding the difference between the target and current percentage allocations of the portfolio. The user's target portfolio allocation percentages are already saved from the user input in the 'Strategy Input' section. However, for the current percentage allocation, the app recalculates the user's portfolio in terms of percentages of the portfolio's total value, rather than asset value, e.g. BTC: 40% instead of BTC: $4,000. The user may have defined one or more asset groups for which there is the additional step. The app iterates through the list of asset groups to and combines the constituent assets of each into a single percentage allocation. Having defined the portfolio in percentage terms, accounting for asset groups, the deltas can be calculated.

For assets that are not in any asset groups, the calculations are the same subtractions of current from target allocations as those in Table 4. The app ignores very small deltas (under 1% of the portfolio's total value) and adjusts the other included deltas slightly to account for their omission.

To calculate the required delta for an asset group, a more complex method is used. The delta for the entire group is calculated in the same way as an individual asset. However, the app calculates the group's target 'equilibrium': the mean asset value if the group were the target value. The app makes adjustments to assets that are the wrong side of this equilibrium, i.e. assets with values beneath the equilibrium if the group's delta is positive, and vice versa for a negative delta. The app calculates a delta for each 'offside' asset that is on the wrong side of the equilibrium. Application of each delta results in the asset group's overall valuation reaching its target, fulfilling the overall delta whilst only adjusting the necessary assets. The list of deltas, adjusted for asset groups, are now used as an input to the algorithm.

3.2.2. Producing Solutions

The part of the algorithm that produces the rebalance transactions is referred to in the app as "Liquidity Matching". This is because it creates a list of prospective purchases and matches it with a list of prospective sells. The algorithm receives the rebalance deltas and the availability of tickers on each exchange as inputs. The steps taken by the algorithm are detailed below:

1. Cash (USD balances) on exchanges is marked as liquidity on the respective exchange that can be used to buy assets, akin to having pre-existing credit on a gift card for a supermarket.

2. The rebalance deltas are split into two lists of positive and negative.

    2.1. Positive deltas are used to create a list of assets to be purchased, along with the quantity to be purchased and the exchanges on which the assets are available.

    2.2. For each negative delta, the list of exchanges on which the asset is available is iterated through in a random order. Assets are marked as being sold on the iterate exchange until the overall delta is satisfied. This is because the investor will have a larger amount of the asset being held than that which needs selling, although often the iteration will only reach the first exchange. This process results in a list of sell transactions used later.

3. The list of buys and sells (created in 2.1 and 2.2 respectively) are matched together by exchange, again in a random order, to create a list of buy transactions. As buys are recorded, the corresponding balance on the respective exchange is deducted by the same amount.

4. Due to exchange-asset availability, the matching process in stage 3 often leaves some required purchases unfulfilled. The remaining purchases are fulfilled in a random order, often requiring a transfer from an exchange with remaining cash. These final buy transactions are added to the existing list. Any transfers are added to a new list of transfer transactions.

5. The output of the algorithm is a list of transactions created by combining the list of sells, transfers and buys consecutively.

The output of this algorithm is a set of transactions which results in a rebalanced portfolio across multiple exchanges. However, the optimisation component is not acknowledged when the solutions are created. This algorithm can be considered a variation of the ‘first-fit’ approach to bin-packing problems, although the randomisation aspect makes it seemingly more ‘random-fit’.

3.2.3. Minimising Transactions

The randomisation in the liquidity matching algorithm was added so that the search space is adequately explored when multiple solutions are produced. Originally, in the several places where random ordering is applied, an ordering by holding size was implemented. Although this seemed to produce strong solutions consistently in a single run, there were concerns that certain possibilities (such as Scenario 2 in section 3.1.3) could be overlooked, i.e. the algorithm found a local minimum rather than the global. The implementation of a randomised algorithm was implemented since the real-world scale of this problem means that this ‘brute-force’ is feasible.

Adding the randomisation was done by replacing any decision made with the size-order heuristic with random ordering. Specifically, these are in the asset selling stage 2.2 and liquidity matching stage 3. Now based on randomness, the algorithm runs for many iterations producing a population of random solutions. Given a random population, the algorithm finds the shortest length of any solution and filters the population to include only those of that length. Using this sub-population, a sort by number of inter-exchange transfers is applied, thereby minimising both the overall number of transactions and inter-exchange transfers. It was found that with an adequate population size an optimal solution could be found consistently. The chosen number of candidate solutions to produce is calculated based on the number of deltas to fulfil and the number of connected exchanges.

4. Evaluation

4.1. App Functionality

Overall the app completes the full extent of its required core functionality. However, there are edge cases and additional features that would add refinement to the app and improve its stability. The design of the app can also be considered to achieve its goal of accessibility and seamlessness. The user interface, Figures 6 to 9 and 12 to 16, uses many standard interface elements from the iOS UI guidelines, giving it a natural appearance in the surrounding ecosystem. Its simple design focuses on the app's core functions, making the app easy to use for an inexperienced investor.

4.1.1. Authentication and Database

The functionality of the authentication and database system is necessarily complete, as it is required for the rest of the app to function. Authentication is fully automatic given that the user has an iCloud account and both private and public storage in CloudKit function without errors.

If the user does not have an iCloud account, or they refused the app's request for permissions, the app currently handles this by presenting a "Not Authenticated" page. Were the app to be developed further, this would be changed into something more constructive: either a prompt to the user to enable their iCloud account in settings or, to take it further, the full capability to authenticate with means other than iCloud. This wasn't implemented in the apps initial version since it is a rare edge case, as almost all iOS devices have an associated iCloud account. Once authenticated, the app goes on either to fetch existing user data or create and save new records.

An originally intended feature of this section was to make use of the keychain storage facility of iOS devices to store the API keys. The keychain uses the 'secure enclave' in the device's processor, offering purpose-built offline storage. The main issue was that, unlike CloudKit, keychain storage is not transferrable between devices and would require complex reconciliation between the on-device storage and the database. Since the app does not transact on the user's behalf and uses only read-only API keys, the utmost security was deprioritised for the sake of implementation simplicity. Similarly, it was intended for the app to make use of TouchID and FaceID biometric capabilities to doubly-authenticate the user. This is simpler to add than keychain storage and would be one of the first to be added if the app's development continues.

4.1.2. Connecting to Exchanges

The app exhibits the full required range of functionality for the four supported exchanges. The functionality of each was manually tested and performed consistently as expected, from accepting API key input (via QR scanner when available) to accessing relevant account data via the exchange's authenticated API and fetching prices from the public API. The app also performs consistently when the user removes an API key pair, successfully deleting the keys from CloudKit.

The validity of an inputted API key is overlooked by the app, which in its current implementation, accepts invalid keys and stores them as if they were valid. Validation was not included primarily because there is no clear way to verify keys without sending API requests. Its implementation was deprioritised given the complication for such a small feature. Another reason is that removing API keys, incorrect or otherwise, can be done easily by the user. The attempted use of incorrect keys does not cause any serious errors such as crashing; it simply means that requests to that API return the 'unauthorised' response.

An investor may not wish to view the full extent of their holdings as one and instead consider particular assets separately from the rest. An unimplemented idea in this section was enabling the

user to select assets to exclude from their portfolio summary. Adding this feature is not straightforward, requiring database updates and additional filtering when handling portfolio data from exchanges. It is also less relevant to the casual investor and, even to those who would use it, provides only marginal utility. Although a useful feature for some, it is not particularly regrettable that this feature was not included.

4.1.3. Strategy Input

This section is highly stable when manually tested against all required functionality. It facilitates the selection of a periodic or threshold rebalance trigger, with different values offered for each. The user can define asset groups and assign percentages of their portfolio to them, along with individual assets.

This section is perhaps the best at handling the edge cases that emerge within it. Several rules which must be upheld regarding asset groups result in edge cases:

1. No asset can be included in more than one asset group.
2. It is not possible to allocate directly to assets that are contained in an asset group.
3. The portfolio is updated when an asset group with a percentage allocation is updated.

For the first rule, a simple filter is used to omit any assets already included in groups from the list of available assets when creating or editing another list. The same filter is used to hide assets in groups from the list of assets seen by the user. The result of the third rule has depends on the situation. If the user removes an asset from an asset group, or deletes the group itself, the target allocation is reset and must be reallocated by the user, given that the group in question was included in the target allocation. However, if the user adds an asset to an asset group it does not require a reset. Instead, any target percentage allocation assigned to the asset is added to the percentage allocation of the group.

4.1.4. Notifications

The notification server achieves the full functionality it would need for ongoing use: reading from CloudKit, fetching price data, performing threshold calculations and writing notifications. However, due to the scope of this project and its time constraints, the ‘always-on’ monitoring aspect wasn’t fully implemented. Had it been, it would have used another NodeJS package, ‘node-cron’, to run the monitoring script regularly after a specified time interval. When run in its current implementation the app consistently receives notifications for its associated portfolio when written by the server.

4.2. Rebalancing Implementation

The liquidity matching algorithm and surrounding population generation procedure has been tested manually with a range of cases. For this testing, the app utilises a dummy repository to allow the specific declaration of asset holdings and availability. Using this, multiple variations of each scenario from Section 3.1.3 were tested and an optimal solution was calculated consistently. The fundamental goal of producing strong solutions to the multiple exchange problem has therefore been categorically accomplished.

Although producing strong solutions, more rigorous prior research of the problem would have clarified and strengthened this project’s approach to its solution. For example, each variation applied to the generic bin-packing problem to formulate the MEP would also benefit from the full exploration of its implications. The brute-force algorithm currently implemented could be optimised via precise study of the required solution population size given the the inputs. This was overlooked in the scope of this project as the population could be ample, without impeding user

experience. Experimentation with ‘evolving’ or producing ‘neighbouring’ solutions could facilitate more efficient approaches than brute force, such as full genetic algorithms or simulated annealing.

There is an obvious unimplemented extension of functionality, however, that would provide a large additional benefit to the user. At present, after it has calculated the series of required transactions, the app merely presents the user with the resulting list. Although this saves the user having to calculate the cross-exchange transactions themselves, they still must initiate the transactions themselves which can be a long process. The obvious extension would be for the app to transact on the user’s behalf and fully execute the rebalance.

The main reason this wasn’t implemented were the surrounding security concerns. Currently the app can perform all its functionality using only read-only API keys provided by the user. When creating API keys, the user is able to select which permissions they intend to grant it. It is common practice to create read-only keys; keys with write permissions are seldom shared with any application unless thoroughly convinced of its security. From the perspective of the application, receiving keys with write permissions leads to higher liability and the need for additional caution. If the offline keychain storage and biometric authentication mentioned in Section 4.1.1 were implemented, then this capability could be added more confidently. However, despite the high level of security, the existing private CloudKit implementation would be deemed insufficient by many prospective users.

When making actual market transactions there is a very high level of variability in outcome and a multiplicity of exceptions that require careful consideration. The resulting complexity of the feature would have made it difficult to implement within the time frame of this project.

4.3. Discussion

In the context of the pre-existing related work, the app produced in this project builds on the functionality of portfolio tracker apps, utilising the risk-reducing power of diversification, whilst maintaining their focus and usability. The app features a purpose-built, randomised liquidity matching algorithm which enables the successful rebalancing of assets across multiple exchanges, creating a population of candidate solutions and selecting the strongest. The report’s research found this capability in only two other apps, in the Related Work section, and neither are free or simple to use. As crypto’s sub-sectors become more established, the power of diversification within it will only continue to grow. This app enables a casual investor to maintain a diversified crypto portfolio in a manner that has never before been so accessible.

The app would perhaps best support the landscape of surrounding work as an open-source proof of concept; a concise tool that utilises exchange APIs to answer a specific challenge. The infrastructure the app contains for storing credentials and connecting to exchanges could serve as the basis for a large range of other functions. Strengthened by its commitment to its architectural pattern, the app’s codebase is accessible to any developer looking to build upon any extent of its functionality. The liquidity matching algorithm could also be adapted to other tasks requiring coordination of capital between exchanges, such as cross-exchange trading strategies.

There are several other future improvements that could provide significant benefit. First would be simply adding more supported exchanges, although this could be expanded to include decentralised exchanges or even crypto wallets. Tracking portfolio performance over time, a standard feature of portfolio trackers, would also provide additional useful information. Finally, inclusion of predefined asset groups representing certain crypto sub-sectors could remove the need for the user to define them. A participant study focusing on the accessibility and usability goal of the project would assist further evaluation of its potential impact on the average investor. Automated testing could also be included in the future to verify the app’s operational stability.

5. Conclusion

This project sought to produce an app that would aid the average investor in applying diversification to the burgeoning crypto sector. Built for iOS using Swift, the resulting app, BalanceBot, efficiently provides the core functionality defined in its specification. It allows the user to connect exchange accounts, gathers relevant portfolio data, accepts input of the user's preferences, and ultimately calculates the rebalance of their assets across multiple exchanges.

Overall, the project can be considered a success, accomplishing its goals in producing a mobile application and generating solutions for a complex optimisation problem. On reflection, this project has been a constructive challenge, testing and improving development skills and exhorting imaginative approaches to problems. Personally, it was the first time adhering to a strict, predefined architecture, requiring additional discipline and forethought when implementing certain features but ultimately educing a genuine appreciation for its value.

There are obvious areas of improvement and extension that would refine the app were it to be developed into a customer-ready product, but a well-designed architecture means that features can be added to the app simply and intuitively. Although the current implementation is adequate, consistently producing optimal results in a near-instant, potential improvements and evolution would be accelerated by further study of the ‘multiple exchange problem’ examined in this report.

6. References

(1) Nakamoto, S. (2008). Bitcoin: A Peer-to-Peer Electronic Cash System. www.bitcoin.org

(2) Perrin, A. (2021). *16% of Americans say they have invested in, traded or used cryptocurrency*. Pew Research Center. https://www.pewresearch.org/fact-tank/2021/11/11/16-of-americans-say-they-have-ever-invested-in-traded-or-used-cryptocurrency/

(3) Sifat, I. (2021). On cryptocurrencies as an independent asset class: Long-horizon and COVID-19 pandemic era decoupling from global sentiments. *Finance Research Letters*, *43*, 102013. https://doi.org/10.1016/J.FRL.2021.102013

(4) Krombholz, K., Judmayer, A., Gusenbauer, M., & Weippl, E. (2017). The other side of the coin: User experiences with bitcoin security and privacy. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *9603 LNCS*, 555–580. https://doi.org/10.1007/978-3-662-54970-4_33

(5) Oppenheim, A. L. (1954). The Seafaring Merchants of Ur. *Journal of the American Oriental Society*, *74*(1), 6. https://doi.org/10.2307/595475

(6) Markowitz, H. (2009). Portfolio selection. *Harry Markowitz: Selected Works*, 15–30. https://doi.org/10.2307/2975974

(7) O'Sullivan, Arthur; Sheffrin, Steven M. (2003). Economics: Principles in Action. Upper Saddle River, New Jersey: Pearson Prentice Hall. p. 273. ISBN 0-13-063085-3.

(8) Goetzmann, W. N. (2000). II. Portfolio of Assets. In *An Introduction to Investment Theory*. Yale School of Management. https://viking.som.yale.edu/an-introduction-to-investment-theory/chapter-ii-the-geography-of-the-efficient-frontier/

(9) Brinson, G. P., Hood, L. R., & Beebower, G. L. (1995). Determinants of Portfolio Performance. *Financial Analysts Journal*, *51*(1), 133–138. https://doi.org/10.2469/FAJ.V51.N1.1869

(10) Krishnan, C. N. V., Petkova, R., & Ritchken, P. (2009). Correlation risk. *Journal of Empirical Finance*, *16*(3), 353–367. https://doi.org/10.1016/J.JEMPFIN.2008.10.005

(11) Feder, A., Gandal, N., Hamrick, J. T., & Moore, T. (2017). The impact of DDoS and other security shocks on Bitcoin currency exchanges: evidence from Mt. Gox. *Journal of Cybersecurity*, *3*(2), 137–144. https://doi.org/10.1093/CYBSEC/TYX012

(12) Mandal, C. A., Chakrabarti, P. P., & Ghose, S. (1998). Complexity of fragmentable object bin packing and an application. *Computers & Mathematics with Applications*, *35*(11), 91–97. https://doi.org/10.1016/S0898-1221(98)00087-X

(13) Liu, W. (2019). Portfolio diversification across cryptocurrencies. *Finance Research Letters*, *29*, 200–205. https://doi.org/10.1016/J.FRL.2018.07.010

(14) Apple. (n.d.). *iOS Deisgn Themes - Human Interface Guidelines*. Retrieved April 14, 2022, from https://developer.apple.com/design/human-interface-guidelines/ios/overview/themes/

(15) Naumov, A. (2019). *Clean Architecture for SwiftUI*. https://nalexn.github.io/clean-architecture-swiftui/

(16) Naumov, A. (2019). *Clean Architecture for SwiftUI + Combine - GitHub* . https://github.com/nalexn/clean-architecture-swiftui

(17) Naumov, A. (2019). *Separation of Concerns in Software Design*. https://nalexn.github.io/separation-of-concerns/

(18) Mishra, A. (2017). The MVVM Architectural Pattern. *IOS Code Testing*, 43–60. https://doi.org/10.1007/978-1-4842-2689-6_3

(19) Hudson, P. (2019). *CodeScanner*. https://github.com/twostraws/CodeScanner

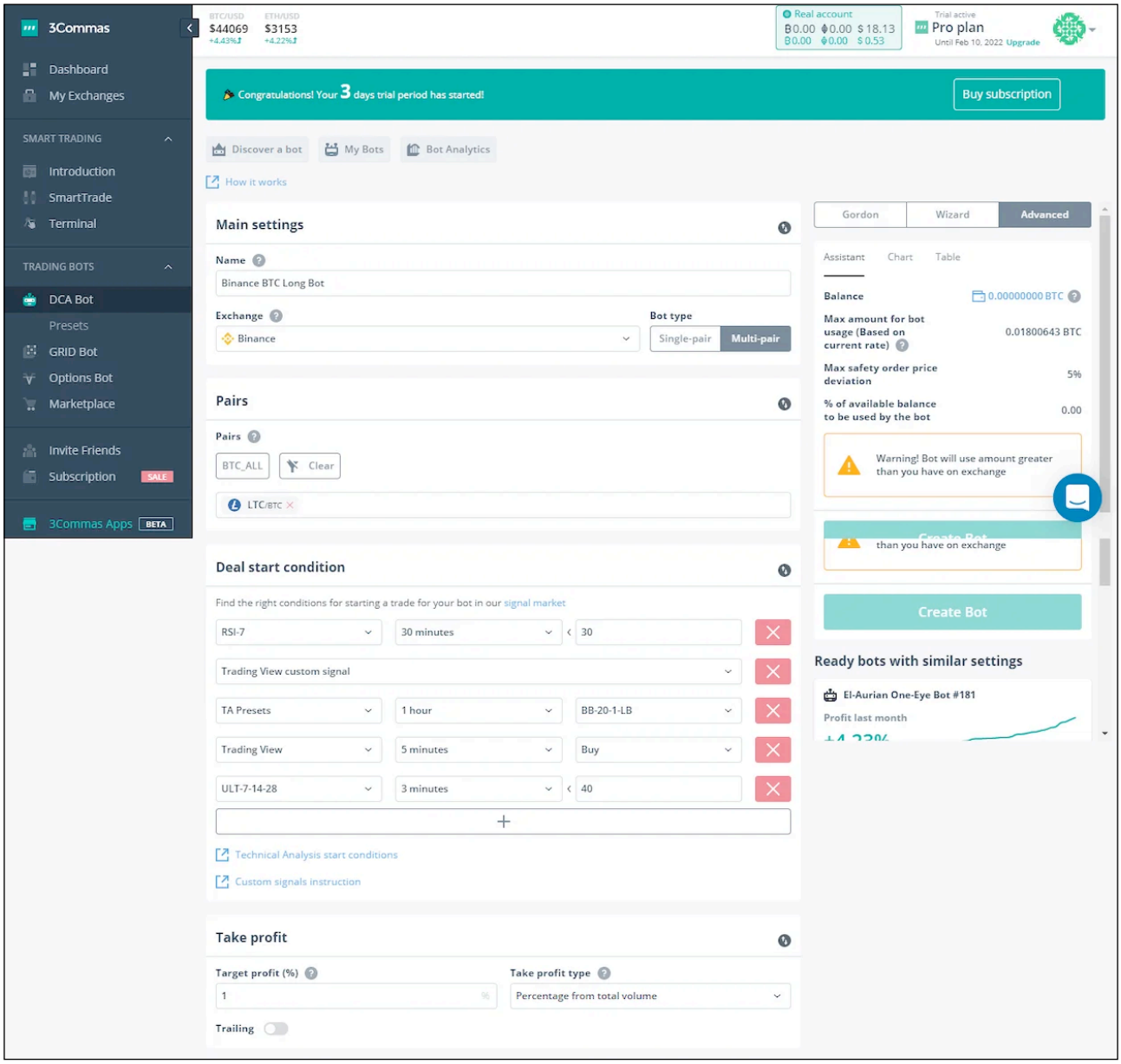
(20) CoinGecko. (n.d.). *Crypto API*. https://www.coingecko.com/en/api/documentation

(21) Lecun, B., Mautor, T., Quessette, F., & Weisser, M.-A. (2013). Bin Packing with Fragmentable Items: Presentation and Approximations. https://hal.archives-ouvertes.fr/hal-00780434

(22) Heßler, K., Irnich, S., Kreiter, T., & Pferschy, U. (2020). Lexicographic Bin-Packing Optimization for Loading Trucks in a Direct-Shipping System (Issue 2009). https://ideas.repec.org/p/jgu/wpaper/2009.html

(23) Ghomi, H. M. (2013). Three-Dimensional Knapsack Problem with Pre-Placed Boxes and Vertical Stability. *Electronic Theses and Dissertations*. https://scholar.uwindsor.ca/etd/4987

7. Appendices



Appendix A: 3Commas Bot Configuration Page