Cryptolium Application

2018 Computing Coursework OCR

Comment to examiners:

1. This project

Contents

[2 Planning 4](#_Toc509529708)

[2.1 Preparation for interview ✓ 4](#_Toc509529709)

[2.2 Initial Interview ✓ 5](#_Toc509529710)

[2.3 Client Brief 6](#_Toc509529711)

[2.4 Minimum Viable Product (MVP) 6](#_Toc509529712)

[2.5 Additional specification 7](#_Toc509529713)

[2.6 Similar product research 7](#_Toc509529714)

[2.6.1 Coin Ticker iPhone - https://itunes.apple.com/gb/app/coin-ticker-bitcoin-altcoin/id636476147 7](#_Toc509529715)

[2.6.2 Cryptolio - https://github.com/larion/cryptolio 8](#_Toc509529716)

[2.6.3 CryptoCompare - https://www.cryptocompare.com/portfolio/ 8](#_Toc509529717)

[2.7 General development model 9](#_Toc509529718)

[2.8 Technologies needed 9](#_Toc509529719)

[2.8.1 Language Choice 9](#_Toc509529720)

[2.8.2 APIs 10](#_Toc509529721)

[2.8.3 Boilerplate comparison 11](#_Toc509529722)

[2.8.4 Note about FlowJS 11](#_Toc509529723)

[2.8.5 Note about Licenses 12](#_Toc509529724)

[2.8.6 Data visualization framework 13](#_Toc509529725)

[2.8.7 Testing framework 14](#_Toc509529726)

[2.8.8 Hardware and software requirements 14](#_Toc509529727)

[2.8.9 Conclusion 15](#_Toc509529728)

[2.9 Basic Layout design 15](#_Toc509529729)

[2.10 Tests needed for MVP 18](#_Toc509529730)

[2.11 Name Choice 20](#_Toc509529731)

[2.11.1 Considerations and comments 20](#_Toc509529732)

[2.12 Problem splitting/Project Diagram 20](#_Toc509529733)

[2.12.1 UI Flow 20](#_Toc509529734)

[2.12.2 Total project diagram 21](#_Toc509529735)

[2.13 Time management 22](#_Toc509529736)

[2.14 Design transcript 22](#_Toc509529737)

[2.14.1 Style choices 22](#_Toc509529738)

[3 Development 24](#_Toc509529739)

[3.1 Testing 24](#_Toc509529740)

[3.1.1 Testing Needed 24](#_Toc509529741)

[3.2 Setup 24](#_Toc509529742)

[3.2.1 SVN ✓ 24](#_Toc509529743)

[3.2.2 Github Project board ✓ 26](#_Toc509529744)

[3.2.3 Boilerplate ✓ 26](#_Toc509529745)

[3.2.4 Travis CI x Provide advantages in checklist form like Section 2.3 28](#_Toc509529746)

[3.2.5 Security checklist ✓ 29](#_Toc509529747)

[3.2.6 Package choice ✓ 31](#_Toc509529748)

[3.3 Intelligent Error Handling ✓ 33](#_Toc509529749)

[3.3.1 Implementation 33](#_Toc509529750)

[3.4 Project structure at end 34](#_Toc509529751)

[3.5 Programming Features 36](#_Toc509529752)

[4 Evaluation 37](#_Toc509529753)

[4.1 Test Results 37](#_Toc509529754)

[4.2 Security Problems 37](#_Toc509529755)

[5 Conclusion 38](#_Toc509529756)

[5.1 Similar product – Cointracker HN 38](#_Toc509529757)

[5.2 Similar product- https://getdelta.io/ 38](#_Toc509529758)

[5.3 Project Structure 38](#_Toc509529759)

[6 Appendix A - Source code 39](#_Toc509529760)

[7 Appendix B - Running the application 40](#_Toc509529761)

[8 Bibliography 41](#_Toc509529762)

# Planning

## Preparation for interview ✓

A client contacted me with a potential idea for an application concerning cryptocurrencies. I said after a consultation and interview I would be able to evaluate whether the project was feasible and possible costs contained.

I initially prepared for the interview with a few questions:

* What would the product entail?
  + Further probing questions
* What timeframe would be ideal?
* Whether they would mind me using it as a project
* Various questions about cost and payment which I will omit for this writeup

## Initial Interview ✓

[Start transcript 1]

[…]

**Me**

What do you imagine this product entailing?

**Client**

So basically, Crypto Exchanges have APIs.

I was wondering if it would be possible to create a desktop app that collates all of these into one manageable portfolio.

I cannot find a windows PC version of any manager out there

and certainly not one that imports using the APIs provided by the exchanges

**Me**

like information on the current exchange rate?

**Client**

yeah, and pulls the current amount of stock you hold in each coin

bittrex *[ref 1]* currently have one that I can use on an iOS app

**Me**

hmm, interesting - I mean it would need to integrate with wallets which would be more complex. Why not just use a website to look up the data?

**Client**

I have 5 different exchanges

about 10 coins on each,

keeping the value of each and the percentage profit is a nightmare

especially if I'm day trading

I just need a better way of keeping track

**Me**

Definitely sounds possible from the offset but give me some time to look at the APIs and similar products current available.

[…]

[End Transcript 1]

**References:**

1: Bittrex iOS app: <https://itunes.apple.com/us/app/b-trex/id1258071406?mt=8>

At the end of this interview I started to research available similar products (elaborated later in section 1.5 ‘Similar product research’) additionally available APIs for cryptocurrency access (elaborated later in section 1.7.2 ‘APIs’).

## Client Brief

After the initial transcripts the client provided a brief outline of the product to further consolidate my idea of the projects requirements.

A desktop application which allows me to view my current portfolios and balance of bitcoins and various other cryptocurrencies. I would like it to automatically update with the current mean price of the bitcoin to other currencies. I would like it to be customisable, stylish and easy to use. Additionally, I want it integrated with as many different currency exchanges as possible to maximise its usage.

## Minimum Viable Product (MVP)

Final draft of the MVP – the minimum requirements my product must fill for it to be considered ‘complete’.

1. Desktop based application
   1. Able to be installed and run from an applications directory.
      1. The client is primarily concerned with windows and mac, however cross platform support is preferable going forward.
2. Ability to make a portfolio
   1. Should be intuitive
      1. Should have introduction on first load
   2. Ability to add a wallet/exchange/simple amount of coin
      1. Ability to remove wallet / change simple amount of initial coin
   3. Ability to watch coin gain / fall relative to the initial input
3. Persistent storage of user data
   1. Saved to a file somewhere. Not necessarily human readable but reliable.
4. Lookup current exchange rates
   1. Support for multiple exchanges
      1. Average
      2. Binance
      3. Bitflyer
      4. Bitfinex
      5. Bithumb
      6. Bitsamp
      7. Bittrex
      8. Coinnest
      9. Coinone
      10. Gdax
      11. Geminin
      12. Hitbtc
      13. Korbit
      14. Kraken
      15. Liqui
      16. Poloniex
      17. WEX
   2. Allowing changing local currency conversion
      1. Fetching local currencies exchange rate to interact with exchanges
5. Security
   1. Basic Password on entry
      1. This password is not meant to securely protect the product – instead it’s main aim is to prevent anyone physically on the computer just being able to immediately see the data.
   2. This program is not meant to be secure by nature – all the data accessible via exchanges / wallet should be read only

When I had reached a final draft of my MVP. I showed it to my client and asked if they wished to change any of it. My client was able to clarify point 4.a.i. about the non-necessity of security within this product due to its nature; among some other minor changes the finer points of the specification. Once all changes had been made I asked my client to agree that this specification outlined what a MVP would look like and began the project.

## Additional specification

The following is additional ideas suggested by the client and or myself which are not included in the MVP and would be acceptable to not achieve within my application but instead to strive for and possibly implement if time permits.

1. Lazy load of persistent storage
   1. The data from the persistent storage of user data would load ‘lazily’ allowing the user to interact with the app before the user data is loaded – and then when loaded it replaces the current state (to avoid conflicts)
2. Lazy load of cryptocurrency graphs
   1. The graphs should be loaded asynchronously to the main process so is non-blocking to the UI experience and when it has loaded the data it should add it to the UI.

## Similar product research

In the aim of making my application the most relevant and to not reinvent invented products. I looked at many similar products across different platforms.

### Coin Ticker iPhone - <https://itunes.apple.com/gb/app/coin-ticker-bitcoin-altcoin/id636476147>

Coin ticker for iPhone provides many of the features like my specification. It allows the adding of portfolios and connection to read only wallet data, so you can accurately track your worth in the currency you desire. It however is restrictive in its configuration. You can customise what cryptocurrencies you want though the format is list based and hard to analyse accurately. Especially as the graphs used have no scales and instead just notions of increases and decreases.



Figure A graph taken from the app showing Poloneix [a cryptocurrency] data

I suspect this is a symptom of it being a mobile app it is hard to contain all this data in an easy to use screen. This is something that I can improve on through the fact that my application will be desktop based.

### Cryptolio - <https://github.com/larion/cryptolio>

This product is a terminal based crypto currency portfolio released under an MIT license as open source software by a Github user ‘larion’.

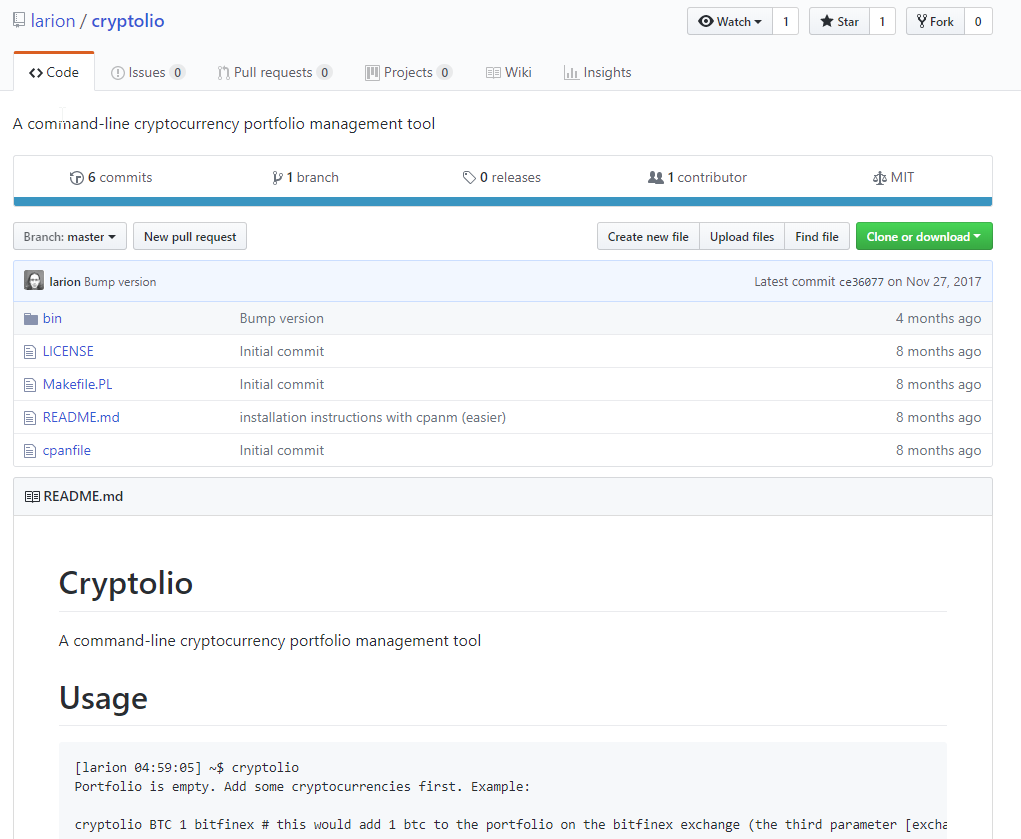


Figure Screen capture of the webpage in which this application is available (larion, 2017)

It has many aspects like my MVP specification:

* It can access data from a variety of different exchanges
* Users are able to add their own crypto ‘holdings’ to it
* It can display this data in a meaningful way to the user

However, it lacks the interface that a GUI based editor or further user settings. That said, it is an important part of my research as it shows the source code behind it as open source software. Therefore, I can examine it and find which APIs it uses. In this case it uses: “<https://api.coinmarketcap.com>”.

### CryptoCompare - <https://www.cryptocompare.com/portfolio/>

CryptoCompare

## General development model

Throughout the development of this application I for a spiral model of development. This allows me to create a very detailed plan to show the work necessary to the coursework requirements and additionally being able to develop the best application possible during the short development window. It also allows me to evaluate my applications performance at the end of the development change.



Figure Spiral model development (Boehm, 2004)

## Technologies needed

### Language Choice

There are many languages available that would adequately fit the requirements of the project and or client. Languages such as C# are well known for being able to cope with desktop GUIs very well and are used for a variety of large projects (Github Inc, 2018). Java additionally is well known especially with its JavaFX framework. There is additionally a relative newcomer to desktop UI design called ElectronJS (Electron JS, 2017). I have discounted a web-based product purely because the MVP specification the client gives wants a **desktop based** client. To decide which one was most applicable to this application I compared the pros and cons of each:

#### C# / WPF - <https://docs.microsoft.com/en-us/dotnet/framework/wpf/getting-started/introduction-to-wpf-in-vs>

This framework is a Windows centric (though cross platform) way of providing enterprise level desktop applications.

##### Advantages

* Well supported/Much documentation
* Very well used

##### Disadvantages

* Higher learning overhead
* Closed Source
* Restrictive design / structure

#### Java / JavaFX

This is a cross platform approach of providing desktop applications using their prescriptive xml based markup language.

##### Advantages

* Well-structured language made to fit OOP

##### Disadvantages

* Learning overhead with the xml language
* Harder to make look native (cannot naturally embed native UI elements - *easily*)
* Closed Source
* Notoriously bad editor for the UI (however improved recently)
* Java has long compile times which make rapid development harder even with on the run class swapping

#### Electron - <https://electronjs.org/> - (ElectronJS, 2018)

This framework centres around being completely cross platform and just providing in effect a chromium browser window available to render any modern HTML/CSS/JavaScript. (Electron JS, 2017)

##### Advantages

* Very easy to setup
* Cross platform
* Can still access lower level OS features
* Familiar technologies
* Open Source (MIT License - (Github, Inc, 2018))

##### Disadvantages

* Has large RAM overhead (Various, 2016)
* Larger file size (Various, 2016)
* Harder to make look native (cannot naturally embed native UI elements, *easily*)

#### Conclusion

In the end I believe ElectronJS is the best choice to be able to build the application the client needs. This is due to its low learning overhead and easy cross-platform compatibility. This will be important as a low learning overhead ensures the best code can be written quickly and efficiently. Additionally, in an age with faster and faster computers, the so-called ‘bloat’ we get from embedding effectively a chrome browser within our application is mitigated. This is especially true as our application’s most intensive task with undoubtedly fetching data from an API – which is unlikely to slow down the whole computer.

### APIs

Researching the APIs, I wish to use to get each bit of data such as currency rates/cryptocurrency exchange rates etc. Here’s some I have found during the planning stage:

* <http://fixer.io/>
* <https://github.com/ccxt/ccxt>

### Boilerplate comparison

When creating desktop applications with electron there can be a lot of setup such as setting up the electron build process, Hot-module-reloading for fast development and other components. Additionally, it is helpful to use a MVC framework such as ReactJS or Angular to improve development time and prevent bulk in the html codebase. This in turn presents a problem of managing state in large programs which is generally done through libraries like redux (or MobX) which have direct bindings into Angular or React i.e. react-redux (reactjs, 2018).

One well known resource for electron boilerplates is the “awesome-electron” repository which lists tools that use electron, tools for electron, as well as boilerplates: <https://github.com/sindresorhus/awesome-electron#boilerplates>

It shows a few such as electron-vue, electron-react-boilerplate and others. Though vue and angular both have their own unique boiler plates I am most familiar with ReactJS so I opted for the electron-react-boilerplate ( <https://github.com/chentsulin/electron-react-boilerplate> ). It comes with many advantages such as hot module reloading (allowing modules to be swapped out during development). Additionally, FlowJS to prevent static type errors, it also has a built-in electron packager to easily produce my app as an installing item.

### Note about FlowJS

FlowJS is a static type checker for JavaScript built by Facebook (Facebook Inc., 2018). It allows me to augment my JavaScript code with type blocks as shown:



Figure An example of a flow type block

This code defines a type ‘actionType’ as an object literal which takes two read only keys: ‘type’ which has a string value and ‘payload’ which can have any value and is optional.

What should be noted about the introduction of flow into my code is that it is still valid JavaScript code when the flow types are removed. In all senses and purpose, they can just be treated as additional comments to the code. Though to make it extra clear what the actual JavaScript looks like I have automatically generated a `\_no-flow-src` folder in my final application’s code. This contains all the same files, however all JavaScript files have had the flow notation removed, so just the runnable JavaScript is left.

### Note about Licenses

Throughout my project I will make use of various open source software (OSS). This is commonplace within enterprise software; for example, here is a section on Third party software within the Spotify desktop application (a well-known music streaming software).

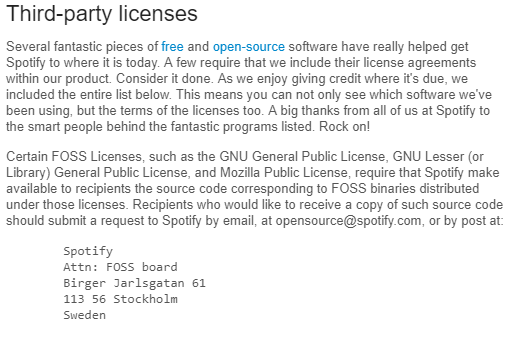


Figure An excerpt from Spotify’s desktop application about Third Party Software

However, care must still be taking concerning licenses. Most open source software imposes conditions, though normally light.

For example, one of the most popular licenses: MIT (Github, Inc, 2018) requires the license and copyright notice to be distributed with it in any software. To correspond with these conditions, when building my project, I installed a package called `electron-license` and included it in my build process as so.



Figure Part of my final package.json – this script is run on building of the project. It compiles all the licenses within all the projects I use and then puts it all in one file – the `LICENSE` file within the release folder.

This then generated a nice license file listing each of the OSS licenses/projects used in the release folder of the project:

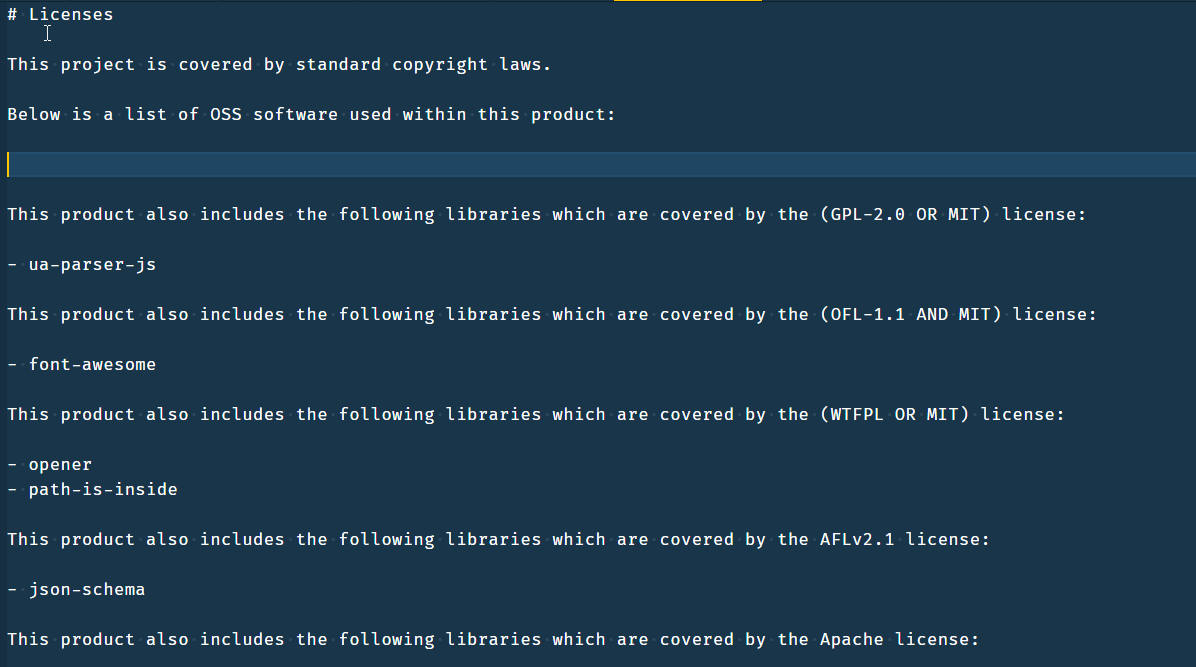


Figure An excerpt from the LICENSE file generated - in total it is more than 1500 lines!

### Data visualization framework

In my application my client has requested various data visualizations. These include candle-stick charts (Wikipedia, 2018). To visualize these properly in my application without spending a needless amount of time generating my own visualization framework I needed to conclude which framework was best for my use case. It came down to two options in the end.

#### D3.js - <https://d3js.org/>

Advantages:

* Well supported (by open source community)
* Many built in graph types
* Easy to add remove data while the program is running

Disadvantages:

* Large learning curve
* Exposes SVG APIs (would be good but I’m not as familiar with them, therefore again serve to add to the large learning curve).
* Doesn’t come with easy export features of the graph

#### Plot.ly - <https://plot.ly/>

Advantages:

* Many built in graph types
* Well supported (by Plotly Ltd and open source community (Plotly Ltd, 2018))
* Easy built in zoom features/export etc.
* Almost no learning curve

Disadvantages:

* Less customisability
* Harder to style

#### Conclusion

I decided to go with Plot.ly based on a simple advantage/disadvantage analysis. The built in features and almost no learning curve are very important due to the time constraints of the project, and well worth a sacrifice in customisability.

### Testing framework

To allow me to get real time indications of the products parity with the original specification I had to introduce a testing framework into my project. The choice of it was made easy by the boilerplate I had chosen (see 1.8.3).

The testing framework I chose was Jest - <https://github.com/facebook/jest> (with additions such as Enzyme for React testing - <https://github.com/airbnb/enzyme>).

### Hardware and software requirements

The hardware and software requirements are important to analyse especially relative to the client’s requirements. From private consultation with the client they have stated how they are using a relative modern computer with Windows 10. Many those investing in new cryptocurrencies are likely to have more modern computers.

The base requirements for electron are as below:



Figure Supported systems (ElectronJS, 2018)

My application would not require any special additional requirements on top of ElectronJS’s ones except for possibly an internet connection to fetch the data. However, it would be able to run without it and would have graceful degradation of content (W3C, 2015) as needed by the MVP.

### Conclusion

## How my MVP and general product is solvable using the technologies I have chosen

## Problems that will be hard to solve

## Basic Layout design

I designed a basic overview of what I wanted the app to look like which is shown below.



Figure – A basic design of what the application might look like

Colours used for mockup:

|  |  |
| --- | --- |
| Area | Colour (#Hex) |
| Left side bar background | #1C1745 |
| Up arrow left sidebar forecolour | #4ABF40 |
| Down arrow left sidebar forecolour | #BF4240 |
| Padlock left side colour forecolour | #FFE37F |
| Text colour left sidebar | #D7CDF2 |
| Background colour main area blocks | #D7D7DB |

This design is heavily subject to change as the app is pushed through development.

Additionally, I modelled an icon for the application based on the Wikimedia cryptocurrency logo as shown below:



Figure Retouched cryptocurrency logo / New Application logo

Colour Specification for logo:

|  |  |
| --- | --- |
| Area | Colour (#Hex) |
| Top right side gradient stop | #FF52E5 |
| Bottom left side gradient stop | #F6D242 |

## Tests needed for MVP

|  |  |  |  |
| --- | --- | --- | --- |
| **Test ID** | **Test name** | **Test Description** | **MVP Spec** |
|  | Basic Load | The application loads up | 1 |
|  | A UI Exists | The UI is present in the rendered application | 1 |
|  | Installation (windows) | The UI can be installed to an applications directory (windows) | 1.a.i |
|  | Installation (mac) | The UI can be installed to an applications directory (mac) | 1.a.i |
|  | On first load displays the portfolio creator | Displays portfolio | 2.a.i |
|  | Portfolio Creator: Add Base Coin | Allows the user to add a base coin in the portfolio adder | 2.a.i |
|  |  |  |  |
|  | Portfolio Creator: Add Wallet | The user can interact with the portfolio creator (probably through a click) to add a wallet id. | 2.a.i |
|  | Portfolio Creator: Add Exchange | The user is able to interact with the portfolio creator (through clicking) to add an exchange id. | 2.a.i |
|  | Has data from multiple exchanges |  | 2.a.i |
|  | Main Graph Adder: Exchanges loaded | A list of exchanges are loaded into the main graph adder and displayed to the user (via a select menu etc.) |  |
|  | Main Graph Adder: User can select and exchange | The user is able to select an exchange using a drop down box from the main graph adder |  |
|  | Main Graph Adder: Symbols loaded | A list of symbols are loaded into the main graph adder and displayed to user (via select menu etc.) |  |
|  |  |  |  |
|  |  |  |  |
| E.1. | Lazy load of configs |  | E.1.a |
| E.2. | Lazy load data from URLs |  | E.2.a |

Tests beginning with ‘E’ are tests which are part of the additional specification (see Section 2.5 ‘Additional Specification’)

## Name Choice

Choosing a name may seem like a trivial task for anyone. However, it could be argued that the name has an impact on the clients view on the final product. (Reynolds, 2016)

Considered names need to reflect the nature of the application:

* Modern
* Cryptocurrency
* Portfolio
* Sleek
* Easy to use
* Secure
* Safe

### Considerations and comments

* Cryptolio
  + Portmantuas are cliché and non-modern but effective
  + Has a name clash with <https://github.com/larion/cryptolio>
* Crypto Buddy
  + Overly friendly, doesn’t seem secure?
  + Has a name clash with <http://www.mycryptobuddy.com/>
* BitPortfolio
  + Implies only for bitcoin – or best serves bitcoin.

In the end I decided Cryptolio sounded the best however it had a name clash with a terminal based crypto currency portfolio. So, I decided to change it slightly into **Cryptolium**. Which I believed made it sound more professional.

## Problem splitting/Project Diagram

### UI Flow

It was essential to decide the programs flow before any UI creation was made. Therefore, I made a basic UI flowchart showing the flow of the user interface.

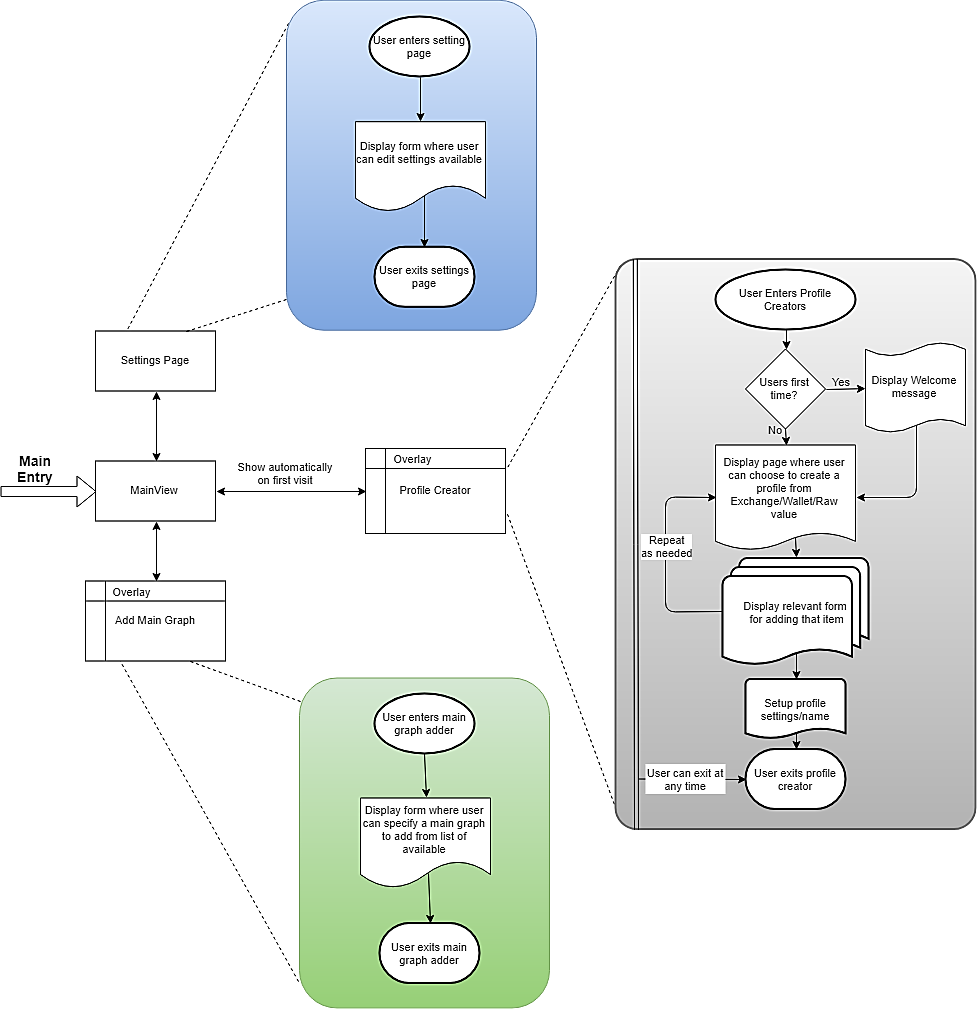


Figure Flowchart showing the basic UI flow for the application

This flow chart shows the intended flow of the UI. The MainView is intended to contain the main graph data / portfolio display. Through triggering via a button or link the overlays, “Add Main Graph,” and “Profile Creator”. The Add Main Graph overlay will allow the user to add a graph to the main view screen. They will get a selection of options on a simple form

### Total project diagram



Figure Complete project diagram

#Evaluation of splitting of problem

## Time management

To plan out the timing of the entire application so it fits to the schedule I prepared a Gantt chart showing the different elements:

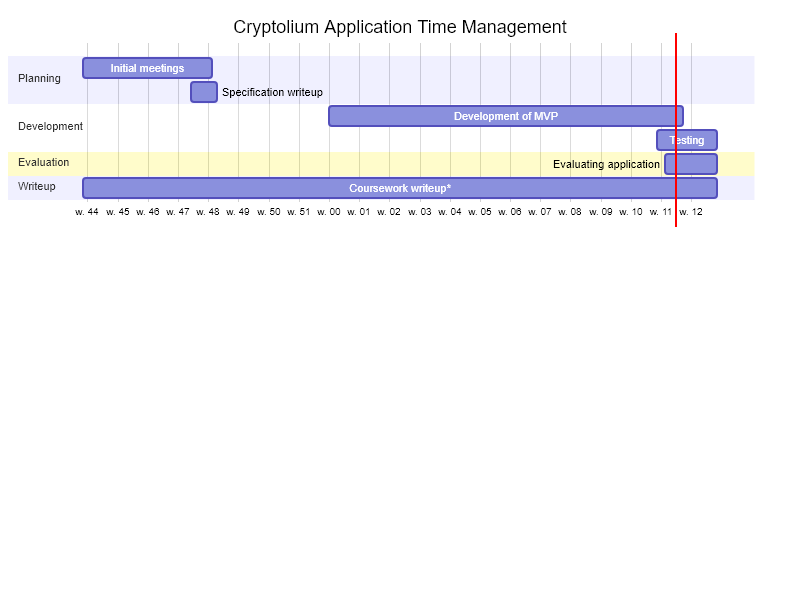


Figure Time management of the application’s entire process

## Design transcript

[…]

### Style choices

Designing an interface which is both effective as well as stylish can be a very hard choice. It is made harder by the…

## Reference clients needs

# Development

## Testing

### Testing Needed

## Setup

Directory layout:



Figure My basic directory layout

### SVN ✓

Early in the project, I introduced a versioning system to better track the progress of the applications development. I created a private (eventually public) GitHub repository to hold the project:



Figure GitHub repository for the application

This also required me to set up a git client on my computer to upload (commit) to the repository. I chose GitKraken due to my familiarity with it:



Figure Setting up GitKraken as a version manager

While making the repository I had to setup various metadata files such as a .gitignore file. This file controls which files are committed to the online repository and which are not. For example, we would not want temporary files or library files to be committed to the online repository.



Figure An example .gitignore <https://gist.github.com/andreasonny83/b24e38b7772a3ea362d8e8d238d5a7bc>

### Github Project board ✓

It is important to be able to easily see the progress you are making through the development of an app to better inform the client of your deadlines and for the developer to easy see what work needs to be done. To make this easier I employed GitHub recently added project boards which allow me to add ‘notes’ which I can then mark as in ‘To do’, ‘In progress’ or ‘Done’ depending on their progress which is reflected easily on a nice progress bar.

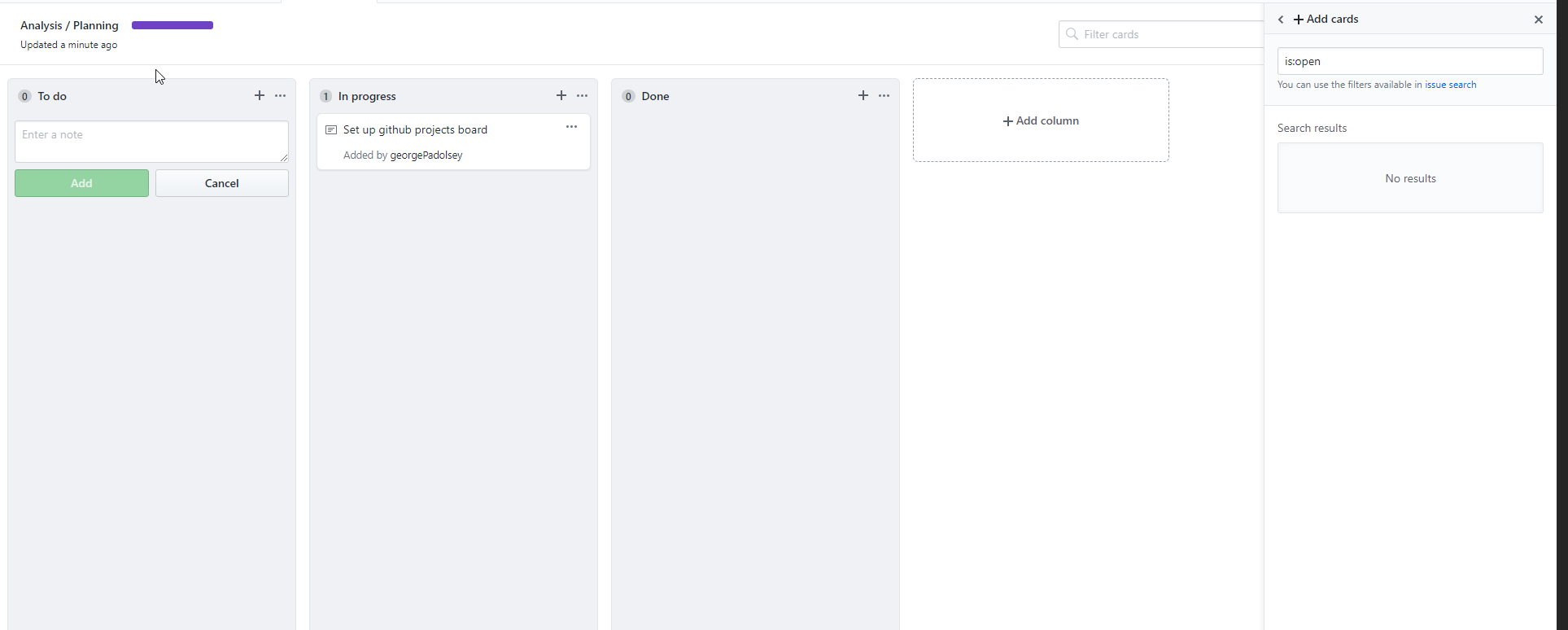


Figure My Github project board for the planning part of the project

### Boilerplate ✓

I realised I made an error by making the .gitignore before cloning my boilerplate into the repository. When I tried to clone the boilerplate into the folder, it caused an error saying the directory had items in. The resolution to this problem was just deleting the .gitignore file I had made.

file:///C:/Users/georg/OneDrive/Documents/ShareX/Screenshots/2018-02/Code_2018-02-04_14-09-31.png

Figure My original attempt at cloning the repository



Figure The .gitignore file



Figure The .gitignore file being deleted.

Finally, I had a fully cloned boilerplate:



Figure Fully cloned boilerplate

### Travis CI x Provide advantages in checklist form like Section 2.3

I decided it might be worth setting up continuous integration that would continuously build and test my application after every commit. I was lucky as the boilerplate library had a prebuilt .travis.yml configuration for Travis CI, a CI I had a private plan for allowing me to use it with the repository.

Unfortunately, when I tried setting it up I got this error:



Figure Travis CI error

I quickly identified based on the error message that this was because the .travis.yml was in the src/ folder with the rest of the boilerplate. I moved the .travis.yml to the root directory of the repository and rewrote the scripts within to change directory to the /src directory where the rest of the code is.file:///C:/Users/georg/OneDrive/Documents/ShareX/Screenshots/2018-02/Code_2018-02-04_14-34-11.png

Figure Part of the rewritten .travis.yml

### Security checklist ✓

In preparation for making the application I read up on how to ensure the electron application is made secure. A well-known document on this topic was released by Doyensec, an independent security agency:

<https://www.blackhat.com/docs/us-17/thursday/us-17-Carettoni-Electronegativity-A-Study-Of-Electron-Security-wp.pdf>



Figure Security Checklist - <https://www.blackhat.com/docs/us-17/thursday/us-17-Carettoni-Electronegativity-A-Study-Of-Electron-Security-wp.pdf>

I implemented each of the changes relevant to my application:



Figure Documentation vs implementation of the checklist



Figure Another example of securing the application – in this case making the build scripts run in sandbox mode (Electron contributors, 2018)

### Package choice ✓

Throughout the development process decisions had to be made which could not be delegated to the client. These decisions would not impact the client in anyway though would impact the developer and development time. For example, the choice of the boilerplate initially was one of those decisions. Repeatedly through the project I decided what was the best way to implement a certain function. For example, I needed a way for user data to persist such as profiles for the app and other configurations. I could roll out my own system for it, however it is such a common problem there are a plethora of opensource packages to choose from. Therefore, I came up with a list and measured each of these advantages and disadvantages between each other:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Package Name | URL | Advantages | Disadvantages | License |
| Cosmiconfig | <https://www.npmjs.com/package/cosmiconfig> | * Multiple formats: can read from a package.json, JSON, YAML, .config.js | * Would need to develop way of getting from electron renderer process to main process to save. * Not electron specific | MIT (Github, Inc, 2018) |
| Properties | <https://www.npmjs.com/package/properties> | * Built in sections * Supports .properties files | * Only supports .properties files (older format) * Would need to develop way of getting from electron renderer process to main process to save. * Not electron specific * API is full of callbacks rather than newer promises (MDN, 2018) – would complicate my codebase. | MIT (Github, Inc, 2018) |
| rc | <https://www.npmjs.com/package/rc> | * Multiple formats: can read from a package.json, JSON, YAML, .config.js * Built in sections | * Would need to develop way of getting from electron renderer process to main process to save. * Not electron specific | MIT (Github, Inc, 2018) and others |
| Configstore | <https://www.npmjs.com/package/configstore> | * Multiple files * Allows encryption * JSON | * Recommends electron-store * Only JSON format | BSD 2-clause (Github, Inc, 2018) |
| preferences | <https://www.npmjs.com/package/preferences> | * Allows encryption * Multiple files * YAML+JSON | * Would need to develop way of getting from electron renderer process to main process to save. * Not electron specific | MIT (Github, Inc, 2018) |
| config | <https://www.npmjs.com/package/config> | * “Simple to setup” | * Would need to develop way of getting from electron renderer process to main process to save. * Not electron specific * Only JSON | MIT (Github, Inc, 2018) |
| Electron-store | <https://www.npmjs.com/package/electron-store> | * Can use from renderer / main – no need for ipc transport * Electron Specific * Multiple files * Intuitive API | * Only JSON format * Uses electron remote – so may impact security configuration. | MIT (Github, Inc, 2018) |
| Electron-settings | <https://github.com/nathanbuchar/electron-settings> | * Electron specific | * Single file | ISC (Github, Inc, 2018) |

To see why licenses the packages are under is important in this process please see Section 1.8.4 (“Note about Licenses”).

N.B. This is meant to serve as an example to the type of process I would go through when choosing each of my packages. However, this one will be more documented to show the process in higher detail.

#### Conclusion

In the end I decided to go with ‘Electron-store’ due to it seeming to be the best for my specific electron-based use case. It had an intuitive API and above all seemed to contain everything I needed in my application.

Possibly add more to this section though overwise is done

## Intelligent Error Handling ✓

I realised during the creation of my application that every time I received an error, it would just show a blank screen to the user in the main window and I’d have to open the developer console to view the error. To make it clearer when an error happened I used a new feature of React 16 called ‘error boundaries’:

Error boundaries are React components that **catch JavaScript errors anywhere in their child component tree, log those errors, and display a fallback UI** instead of the component tree that crashed. Error boundaries catch errors during rendering, in lifecycle methods, and in constructors of the whole tree below them.

Figure A quote from the official React dev block article concerning Error Boundaries (Facebook Inc., 2018)

This provided clear advantages:

* Allowed the user to instantly see an error has happened (instead of a blank screen)
* It informs them to contact the developer with a brief description of it which encourages bug fixing
* It provides a better user experience

### Implementation

I implemented one error boundary as shown:

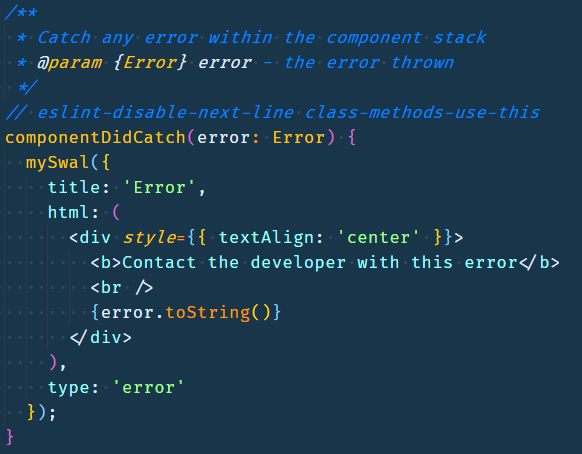


Figure An excerpt of my code implementing the error boundary

This error boundary was implemented on my main component – the `Home` component. The mySwal function is Sweet Alert 2 with react content (<https://sweetalert2.github.io/> + <https://github.com/sweetalert2/sweetalert2-react-content>).

It displays an error as shown:

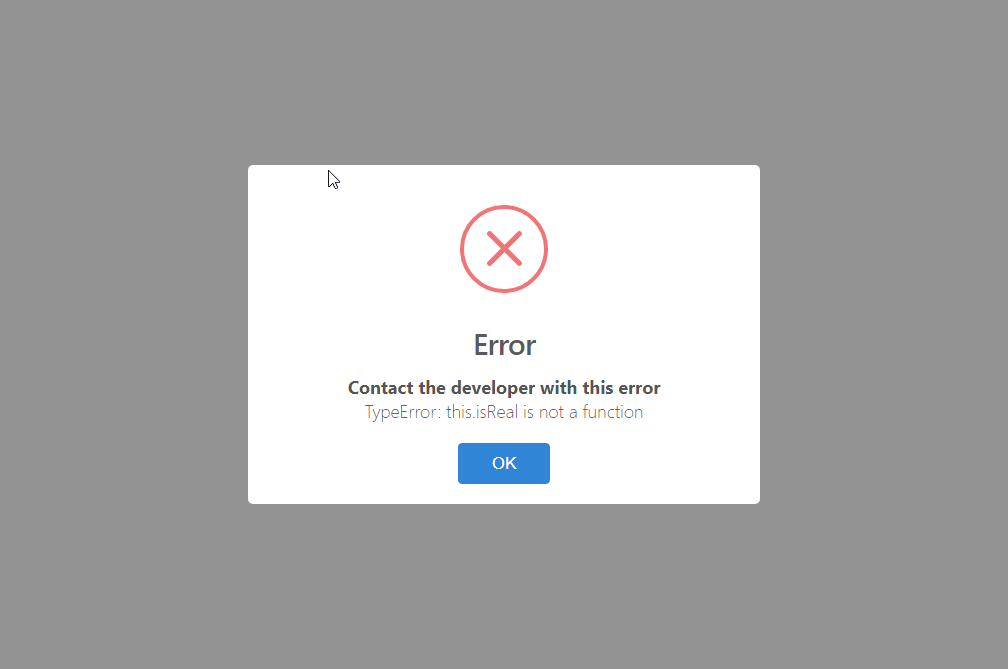


Figure Example error presented using error boundary

This helped me locate an error in my code:



Figure The error in my code

## Project structure at end

* Computer Science 2018
  + .git
  + assets
    - screenshots
  + docs
  + resources
  + src
    - .vscode
    - app
      * actions
        + types
      * components
      * containers
      * dist
      * enc\_keys
      * node\_modules
      * reducers
        + ipc
      * store
      * utils
      * \_app\_node
      * \_types
    - dll
    - flow-typed
      * npm
        + @fortawesome
    - internals
      * flow
      * img
      * mocks
      * scripts
    - node\_modules
    - release
      * win-unpacked
        + locales
        + resources
    - resources
      * icons
    - test
      * actions
        + \_\_snapshots\_\_
      * components
        + \_\_snapshots\_\_
      * containers
      * e2e
      * reducers
      * utils
  + \_no-flow-src
    - .vscode
    - app
      * actions
        + types
      * components
      * containers
      * dist
      * enc\_keys
      * reducers
        + ipc
      * store
      * utils
      * \_app\_node
      * \_types
    - dll
    - resources
      * icons
    - test
      * actions
        + \_\_snapshots\_\_
      * components
        + \_\_snapshots\_\_
      * containers
      * e2e
      * reducers
      * utils
  + \_screen\_dir
    - app
      * actions
      * components
      * containers

Update with `dirsToLi -d . -I node\_modules,.git`

## Programming Features

Throughout this project is was necessary to use a variety of programming features. These included but not limited to:

* Iteration
* Conditionals
* Switching / Case conditions
* Enums (Enumerated Types)
* Inheritance / Polymorphism
* Functional programming

For the purposes of this writeup I will provide an example for each of these features to show the variety of programming technique used within this project. N.B. These are just excerpts so I can show some of my reasoning during development, the full source code is linked in Appendix A.

# Evaluation

## Test Results

## Security Problems

In this section it may be worth referring to Section 2.3.6, ‘Package Choice’, and Section 2.3.5, ‘Security Checklist’. In my Security Checklist section, I showed how I added a variety of security measures to secure my application. Unfortunately, due to the package choice I made – which require the electron ‘remote’ [cite] object to be accessible. Therefore, many of the security measures introduced never made it into the final source code as they stopped the application from working. An example error message caused is shown:

[Example error message]

# Conclusion

## Similar product – Cointracker HN

Through the creation of this product it came to my attention that a similar product was just realesed by the name of “Cointracker” (Nin Finance, Inc, 2018). I believe my project is significantly different however I contacted my client concerning it. They assured me that they still wished the project to be completed as they believe they will still be able to seek a market for the product.

## Similar product- <https://getdelta.io/>

## Project Structure

# Appendix A - Source code

Due to the size of my project, it seems infeasible to include every single program file as a picture or formatted text within this document. Therefore, it seems the best compromise is to place some of the program files, which best demonstrate the style / programming techniques used within the project and light commentary on which and provide a link to an online repository with all the project files on. Additionally, it contains a very overt README specifying how to run the program if one wishes.

Online repository link:

<https://github.com/georgePadolsey/ComputerScience2018>

N.B. I can guarantee this link will be valid till 2023 at the least.

# Appendix B - Running the application

# Bibliography

Boehm, 2004. *File:Spiral model (Boehm, 1988).svg - Wikimedia Commons.* [Online]   
Available at: https://commons.wikimedia.org/wiki/File:Spiral\_model\_(Boehm,\_1988).svg  
[Accessed 04 02 2018].

Electron contributors, 2018. *electron/sandbox-option.md at master · electron/electron.* [Online]   
Available at: https://github.com/electron/electron/blob/master/docs/api/sandbox-option.md  
[Accessed 19 March 2018].

Electron JS, 2017. *Electron | Build cross platform desktop apps with JavaScript, HTML, and CSS..* [Online]   
Available at: https://electronjs.org/

ElectronJS, 2018. *Supported Platforms | Electron.* [Online]   
Available at: https://electronjs.org/docs/tutorial/supported-platforms  
[Accessed 04 02 2018].

Facebook Inc., 2018. *Error Handling in React 16 - React Blog.* [Online]   
Available at: https://reactjs.org/blog/2017/07/26/error-handling-in-react-16.html  
[Accessed 14 March 2018].

Facebook Inc., 2018. *Flow: A Static Type Checker for JavaScript.* [Online]   
Available at: https://flow.org/  
[Accessed 13 March 2018].

Github Inc, 2018. *Trending C# repositories on GitHub today.* [Online]   
Available at: https://github.com/trending/c%23  
[Accessed 04 02 2018].

Github, Inc, 2018. *BSD 2-Clause “Simplified” License | Choose a License.* [Online]   
Available at: https://choosealicense.com/licenses/bsd-2-clause/  
[Accessed 7 March 2018].

Github, Inc, 2018. *ISC License | Choose a License.* [Online]   
Available at: https://choosealicense.com/licenses/isc/  
[Accessed 7 March 2018].

Github, Inc, 2018. *MIT License | Choose a License.* [Online]   
Available at: https://choosealicense.com/licenses/mit/  
[Accessed 7 March 2018].

larion, 2017. *larion/cryptolio: A command-line cryptocurrency portfolio management tool.* [Online]   
Available at: https://github.com/larion/cryptolio  
[Accessed 15 March 2018].

MDN, 2018. *Promise - JavaScript | MDN.* [Online]   
Available at: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\_Objects/Promise  
[Accessed 19 March 2018].

Nin Finance, Inc, 2018. *CoinTracker · Cryptocurrency Portfolio & Tax Manager.* [Online]   
Available at: https://www.cointracker.io/  
[Accessed 7 March 2018].

Plotly Ltd, 2018. *plotly.js | JavaScript Graphing Library.* [Online]   
Available at: https://plot.ly/javascript/  
[Accessed 21 March 2018].

reactjs, 2018. *reactjs/react-redux: Official React bindings for Redux.* [Online]   
Available at: https://github.com/reactjs/react-redux  
[Accessed 13 March 2018].

Reynolds, M., 2016. *Why Brand Names Are So Important, According To Science | HuffPost.* [Online]   
Available at: https://www.huffingtonpost.com/molly-reynolds/why-brand-names-are-so-im\_b\_11930994.html  
[Accessed March 21 2018].

Various, 2016. *Ask HN: Why / Why Not Use Electron? | Hacker News.* [Online]   
Available at: https://news.ycombinator.com/item?id=12119278  
[Accessed 04 02 2017].

W3C, 2015. *Graceful degradation versus progressive enhancement - W3C Wiki.* [Online]   
Available at: https://www.w3.org/wiki/Graceful\_degradation\_versus\_progressive\_enhancement  
[Accessed 04 02 2018].

Wikipedia, 2018. *Candlestick chart - Wikipedia.* [Online]   
Available at: https://en.wikipedia.org/wiki/Candlestick\_chart  
[Accessed 13 March 2018].