

Computing Coursework 2018

1 Planning

1.1 Preparation for interview

A client contacted me with a potential idea for an application within the general topic of 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]

1.2

1.3 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 available APIs for this kind of usage.

[...]

[End Transcript 1]

References:

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

1.4 Design transcript

[...]

1.5 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.

1.6 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
 - a) Able to be installed and run from an applications directory.
 - i) The client is primarily concerned with windows, however cross platform support is preferable going forward.
- 2) Ability to make a portfolio
 - a) Should be intuitive
 - i) Should have introduction on first load
 - b) Ability to add a wallet/exchange/simple amount of coin
 - i) Ability to remove wallet / change simple amount of initial coin
 - c) Ability to watch coin gain / fall relative to the initial input
- 3) Lookup current exchange rates
 - a) Support for multiple exchanges
 - i) Average
 - ii) Binance
 - iii) Bitflyer
 - iv) Bitfinex
 - v) Bithumb
 - vi) Bitsamp
 - vii) Bittrex
 - viii) Coinnest
 - ix) Coinone
 - x) Gdax
 - xi) Geminin
 - xii) Hitbtc
 - xiii) Korbit
 - xiv) Kraken
 - xv) Liqui
 - xvi) Poloniex
 - xvii) WEX
 - b) Allowing changing local currency conversion
 - i) Fetching local currencies exchange rate to interact with exchanges
- 4) Security
 - a) Basic Password on entry

- i) 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.
- b) 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.

1.7 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:

1.7.1 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 1 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.

1.7.2 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'.

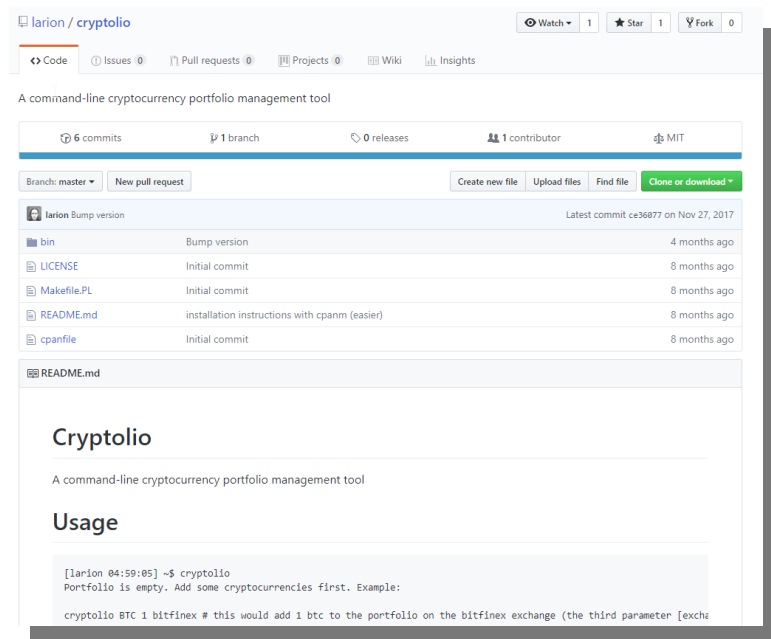


Figure 2 Screen capture of the webpage in which this application is available [CITATION lar17 \l 2057]

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”.

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

1.8 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.

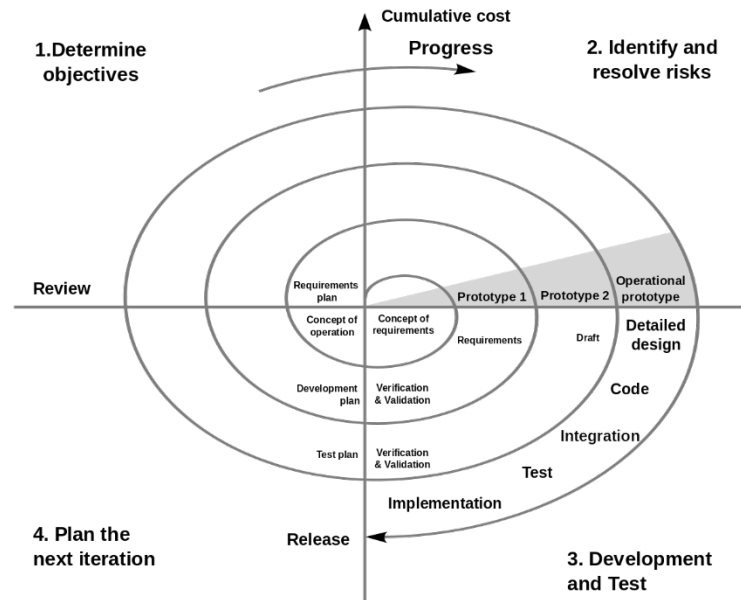


Figure 3 Spiral model development [CITATION Boe04 \l 2057]

1.9 Technologies needed

1.9.1 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 [CITATION Git18 \l 2057]. Java additionally is well known especially with its JavaFX framework. There is additionally a relative newcomer to desktop UI design called ElectronJS [CITATION Ele17 \l 2057]. 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:

1.9.1.1 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.

1.9.1.1.1 Advantages

- + Well supported/Much documentation
- + Very well used

1.9.1.1.2 Disadvantages

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

1.9.1.2 Java / JavaFX

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

1.9.1.2.1 Advantages

- + Well-structured language made to fit OOP

1.9.1.2.2 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

1.9.1.3 Electron - <https://electronjs.org/> - [CITATION Ele18 \l 2057]

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. [CITATION Ele17 \l 2057]

1.9.1.3.1 Advantages

- + Very easy to setup
- + Cross platform
- + Can still access lower level OS features
- + Familiar technologies
- + Open Source (MIT License - [CITATION Git17 \l 2057])

1.9.1.3.2 Disadvantages

- Has large RAM overhead [CITATION Var16 \l 2057]
- Larger file size [CITATION Var16 \l 2057]
- Harder to make look native (cannot naturally embed native UI elements, *easily*)

1.9.1.4 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.

1.9.2 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>

1.9.3 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 [CITATION rea18 \l 2057].

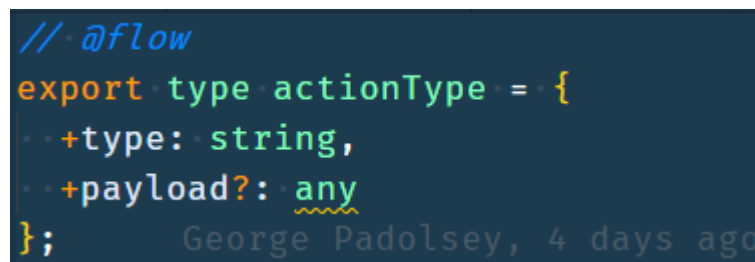
One well known resource for electron boilerplates is the “awesome-electron” repository which lists tools that use electron, tools for electron, as well as boilerplates:

<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.

1.9.4 Note about FlowJS

FlowJS is a static type checker for JavaScript built by Facebook [CITATION Fac18 \l 2057]. It allows me to augment my JavaScript code with type blocks as shown:



```
// @flow
export type ActionType = {
  +type: string,
  +payload?: any
};
```

Figure 4 An example of a flow type block

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 with flow notation in have had it removed so just the runnable JavaScript is left.

1.9.5 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).

Third-party licenses

Several fantastic pieces of [free](#) and [open-source](#) software have really helped get Spotify to where it is today. A few require that we include their license agreements within our product. Consider it done. As we enjoy giving credit where it's due, we included the entire list below. This means you can not only see which software we've been using, but the terms of the licenses too. A big thanks from all of us at Spotify to the smart people behind the fantastic programs listed. Rock on!

Certain FOSS Licenses, such as the GNU General Public License, GNU Lesser (or Library) General Public License, and Mozilla Public License, require that Spotify make available to recipients the source code corresponding to FOSS binaries distributed under those licenses. Recipients who would like to receive a copy of such source code should submit a request to Spotify by email, at opensource@spotify.com, or by post at:

Spotify
Attn: FOSS board
Birger Jarlsgatan 61
113 56 Stockholm
Sweden

Figure 5 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 [CITATION Git17 \l 2057] 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.

```
"generate-licenses": "electron-license build > release/LICENSE",
```

Figure 6 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:

```
# Licenses
|
This project is covered by standard copyright laws.

Below is a list of OSS software used within this product:

This product also includes the following libraries which are covered by the (GPL-2.0 OR MIT) license:
- ua-parser-js

This product also includes the following libraries which are covered by the (OFL-1.1 AND MIT) license:
- font-awesome

This product also includes the following libraries which are covered by the (WTFPL OR MIT) license:
- opener
- path-is-inside

This product also includes the following libraries which are covered by the AFLv2.1 license:
- json-schema

This product also includes the following libraries which are covered by the Apache license:
```

Figure 7 An excerpt from the LICENSE file generated - in total it is more than 1500 lines!

1.9.6 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

1.9.7 Data visualization framework

In my application my client has requested various data visualizations. These include candle-stick charts [CITATION Wik18 \l 2057]. 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.

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

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

1.9.8 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>).

1.9.9 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:

Supported Platforms

Following platforms are supported by Electron:

macOS

Only 64bit binaries are provided for macOS, and the minimum macOS version supported is macOS 10.9.

Windows

Windows 7 and later are supported, older operating systems are not supported (and do not work).

Both `ia32` (`x86`) and `x64` (`amd64`) binaries are provided for Windows. Please note, the `ARM` version of Windows is not supported for now.

Linux

The prebuilt `ia32` (`i686`) and `x64` (`amd64`) binaries of Electron are built on Ubuntu 12.04, the `arm` binary is built against ARM v7 with hard-float ABI and NEON for Debian Wheezy.

Whether the prebuilt binary can run on a distribution depends on whether the distribution includes the libraries that Electron is linked to on the building platform, so only Ubuntu 12.04 is guaranteed to work, but following platforms are also verified to be able to run the prebuilt binaries of Electron:

- Ubuntu 12.04 and later
- Fedora 21
- Debian 8

Figure 8 Supported systems [CITATION Ele18 \l 2057]

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 [CITATION W3C15 \l 2057] as needed by the MVP.

1.9.10 Conclusion

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

Problems that will be hard to solve

1.10 Basic Layout design

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

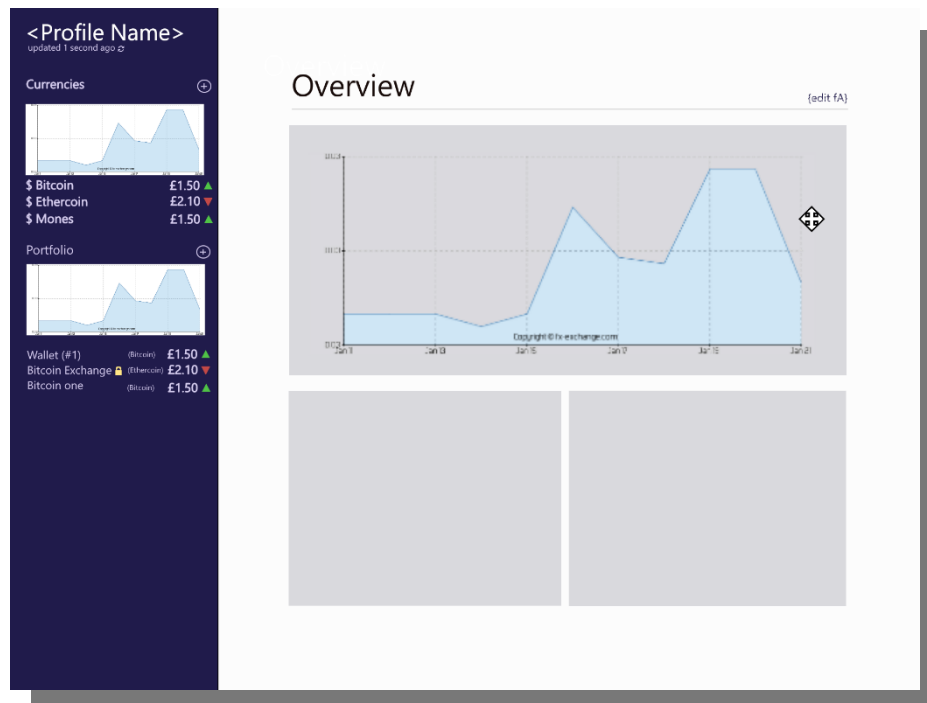


Figure 9 – A basic design of what the application might look like

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:

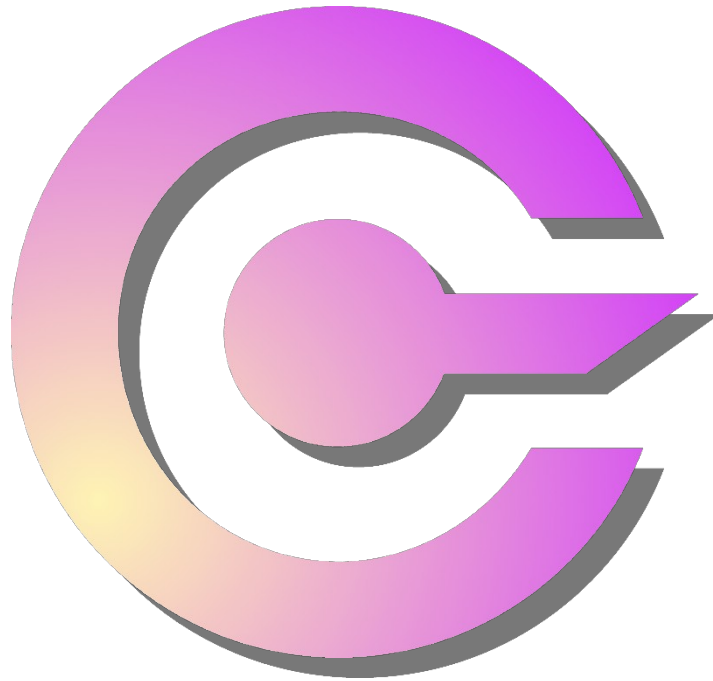


Figure 10 Retouched cryptocurrency logo / New Application logo

Colour Specification for logo:

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

1.11 Tests needed for MVP

Test ID	Test name	Test Description	MVP Spec
1	Basic Load	The application loads up	1
2	A UI Exists	The UI is present in the rendered application	1
3	Installation (windows)	The UI can be installed to an applications directory (windows)	1.a.i
4	On first load displays the portfolio creator	Displays portfolio	2.a.i
5	Portfolio Creator: Add Base Coin	Allows the user to add a base coin in the portfolio adder	2.a.i
6	Portfolio Creator: Add Wallet	The user can interact with the portfolio creator (probably through a click) to add a wallet id.	2.a.i
7	Portfolio Creator: Add Exchange		2.a.i
8	Has data from multiple exchanges		2.a.i

1.12 Name Choice

This may seem like a trivial task for an application. However, it could be argued that the name has an impact on the clients view on the final product.

Considered names need to reflect the nature of the application being:

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

Considered names and thoughts:

- Cryptolio
 - Portmantuas are cliché and non-modern but effective
 - NAME CLASH: <https://github.com/larion/cryptolio>
- Crypto Buddy
 - Overly friendly, doesn't seem secure?
 - NAME CLASH: <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 believes make it sound more professional.

1.13 Problem splitting/Project Diagram

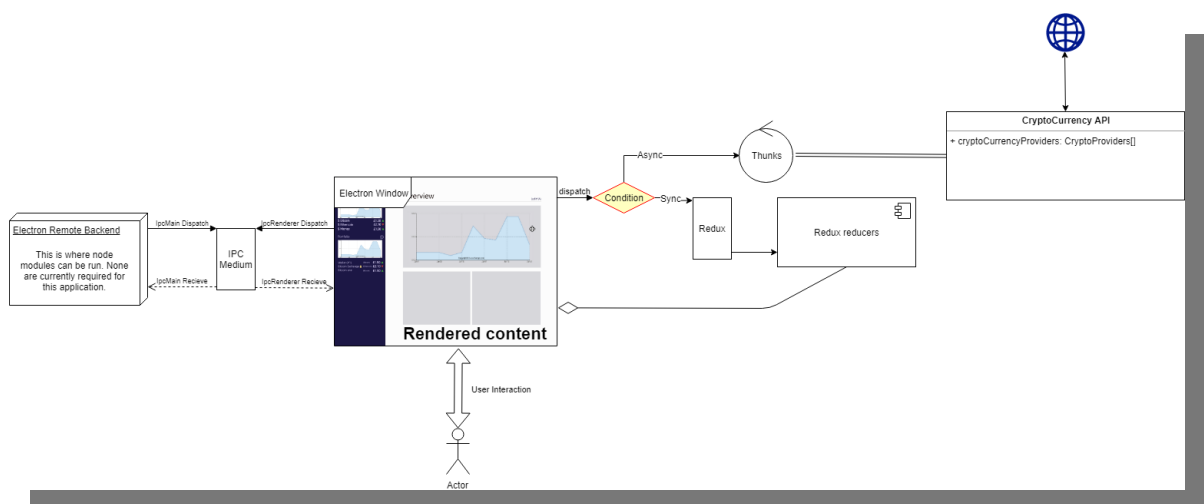


Figure 11 Complete project diagram

#Evaluation of splitting of problem

2 Development

2.1 Testing

2.2 Testing Needed

2.3 Setup

Directory layout:

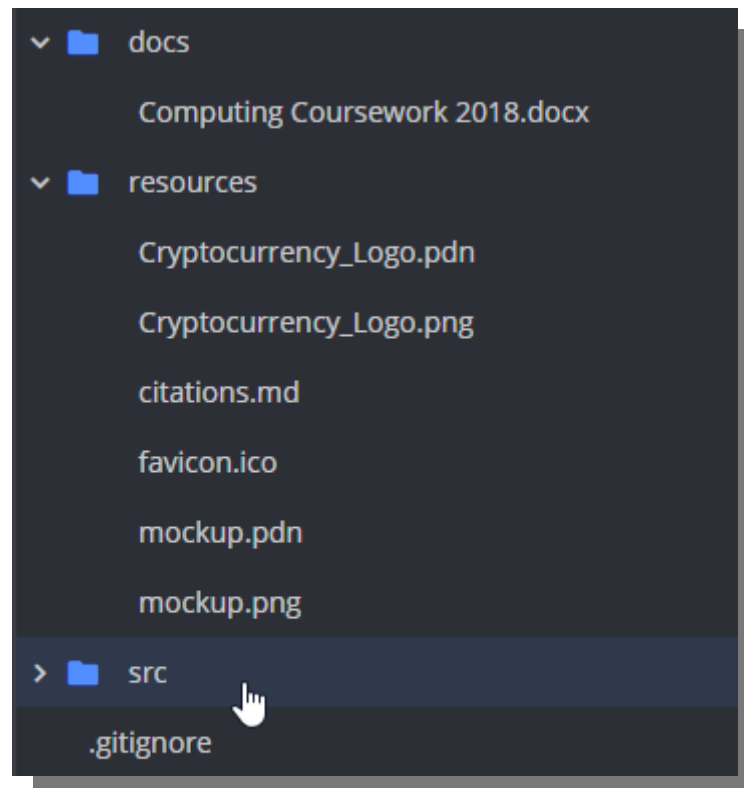


Figure 12 My basic directory layout

2.3.1 SVN

In the pursuit of this project I thought it best I introduce a versioning system to better track the progress of the applications development. This start with me starting a private github repository to hold the project:

[illegible]

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.



2.3.2 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 easily 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 'To do', 'In progress' or 'Done' depending on their progress which is reflected easily on a nice progress bar.

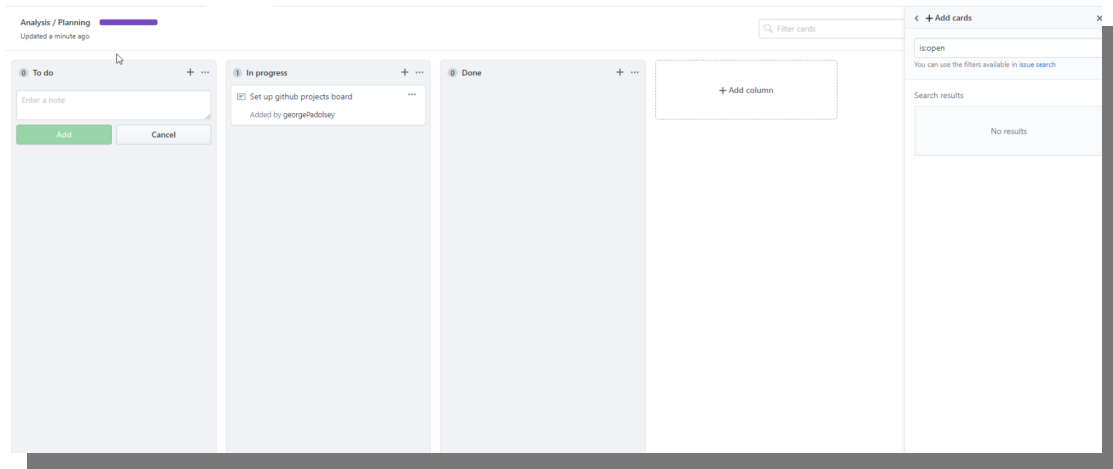


Figure 16 My Github project board for the planning part of the project

2.3.3 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.

```
georg@MegaBiscuitv2 MINGW64 ~/src/Computer Science 2018/src (master)
$ git clone --depth=1 https://github.com/chentsulin/electron-react-boilerplate.git .
```

Figure 17 My original attempt at cloning the repository

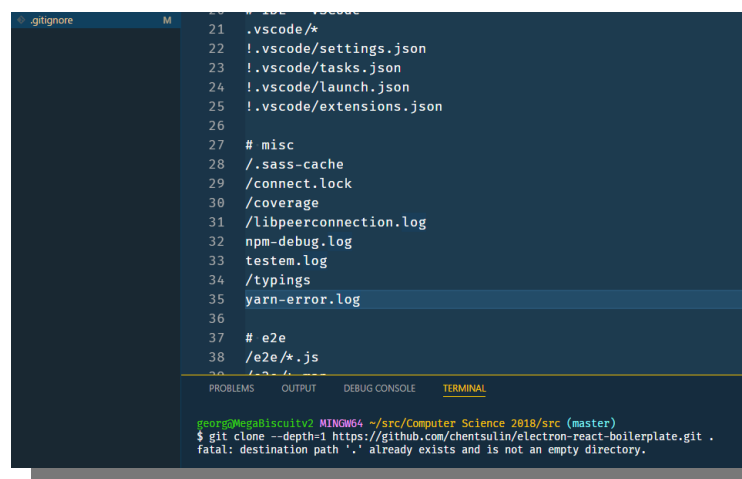


Figure 18 The .gitignore file

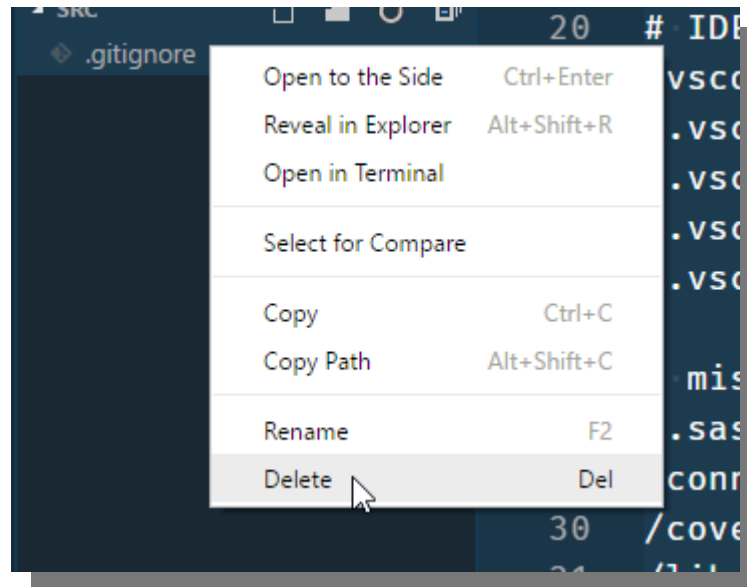


Figure 19 The .gitignore file being deleted.

Finally, I had a fully cloned boilerplate:

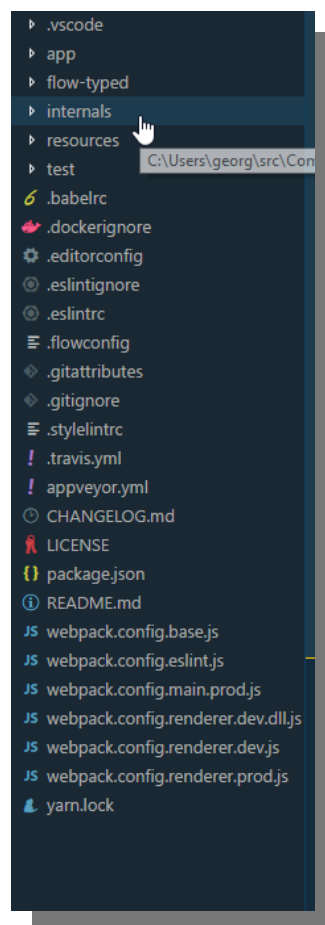


Figure 20 Fully cloned boilerplate

2.3.4 Travis CI

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:

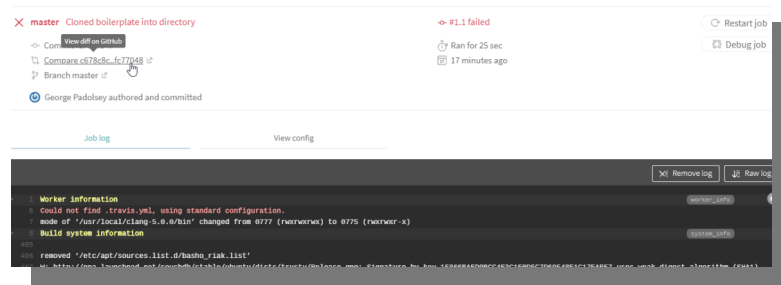


Figure 21 Travis CI error

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.

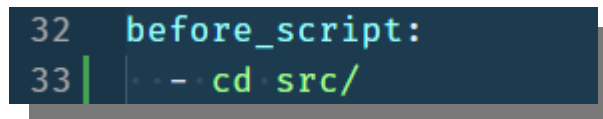


Figure 22 Part of the rewritten `.travis.yml`

2.3.5 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 23 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:

Risk	If enabled, nodeIntegration allows JavaScript to leverage Node.js primitives and modules. This could lead to full remote system compromise if you are rendering untrusted content.
Auditing	<div><p>nodeIntegration and nodeIntegrationInWorker are boolean options that can be used to determine whether node integration is enabled.</p><p>For BrowserWindow, default is true. If the option is not present, or is set to true/1, nodeIntegration is enabled as in the following examples:</p><pre>mainWindow = new BrowserWindow({ "webPreferences": { "nodeIntegration": true, "nodeIntegrationInWorker": 1 } });</pre><p>Or simply:</p><pre>mainWindow = new BrowserWindow()</pre><p>For webview tag, default is false. When this attribute is present, the guest page in webview will have node integration:</p><pre><webview src="https://doyensec.com/" nodeintegration></webview></pre><p>When sandbox is enabled (see below), nodeIntegration is disabled.</p></div> <div><pre>mainWindow = new BrowserWindow({ show: false, width: 1024, height: 728, webPreferences: { // Electron Security Checklist - pg 9 nodeIntegration: false, // not needed as set to false by default but just in case: nodeIntegrationInWorker: false } });</pre></div>

Figure 24 Documentation vs implementation of the checklist

```
webPreferences: {
  //
  contextIsolation: true,
  // Electron Security Checklist - pg 9
  sandbox: true
}

electron --enable-sandbox ./app/",
E_ENV=development electron --enable-sandbox
```

Figure 25 Another example of securing the application – in this case making the build scripts run in sandbox mode [cite]

2.3.6 Package choice

Throughout the development process decisions must be made which cannot be delegated to the client. These decisions will not impact the client in anyway though impact the developer and possible development time. For example, the choosing of the boilerplate initially was one of those decisions. Repeatedly through the project I needed to decide 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. 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 there advantages between eachoter:

Package Name	URL	Advantages	Disadvantages	License
Cosmiconfig	https://www.npmjs.com/package/cosmiconfig	+	–	MIT[CITATION Git17 \I 2057]
Properties	https://www.npmjs.com/package/properties	+	–	MIT[CITATION Git17 \I 2057]
rc	https://www.npmjs.com/package/rc	+	–	MIT[CITATION Git17 \I 2057] and others
Configstore	https://www.npmjs.com/package/configstore	+	–	BSD 2-clause [CITATION Git171 \I 2057]
preferences	https://www.npmjs.com/package/preferences	+ Allows encryption	–	MIT[CITATION Git17 \I 2057]
config	https://www.npmjs.com/package/config	+	–	MIT[CITATION Git17 \I 2057]
Electron-store	https://www.npmjs.com/package/electron-store	+ Can use from renderer / main – no need for ipc transport	–	MIT[CITATION Git17 \I 2057]
Electron-settings	https://github.com/nathanbuchar/electron-settings	+	–	ISC [CITATION Git172 \I 2057]

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.

2.4 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 26 A quote from the official React dev block article concerning Error Boundaries [CITATION Fac181 \l 2057]

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

2.4.1 Implementation

I implemented one error boundary as shown:

```
/**
 * Catch any error within the component stack
 * @param {Error} error - the error thrown
 */
// eslint-disable-next-line class-methods-use-this
componentDidCatch(error: Error) {
  mySwal({
    title: 'Error',
    html: (
      <div style={{ textAlign: 'center' }}>
        <b>Contact the developer with this error</b>
        <br />
        {error.toString()}
      </div>
    ),
    type: 'error'
  });
}
```

Figure 27 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:

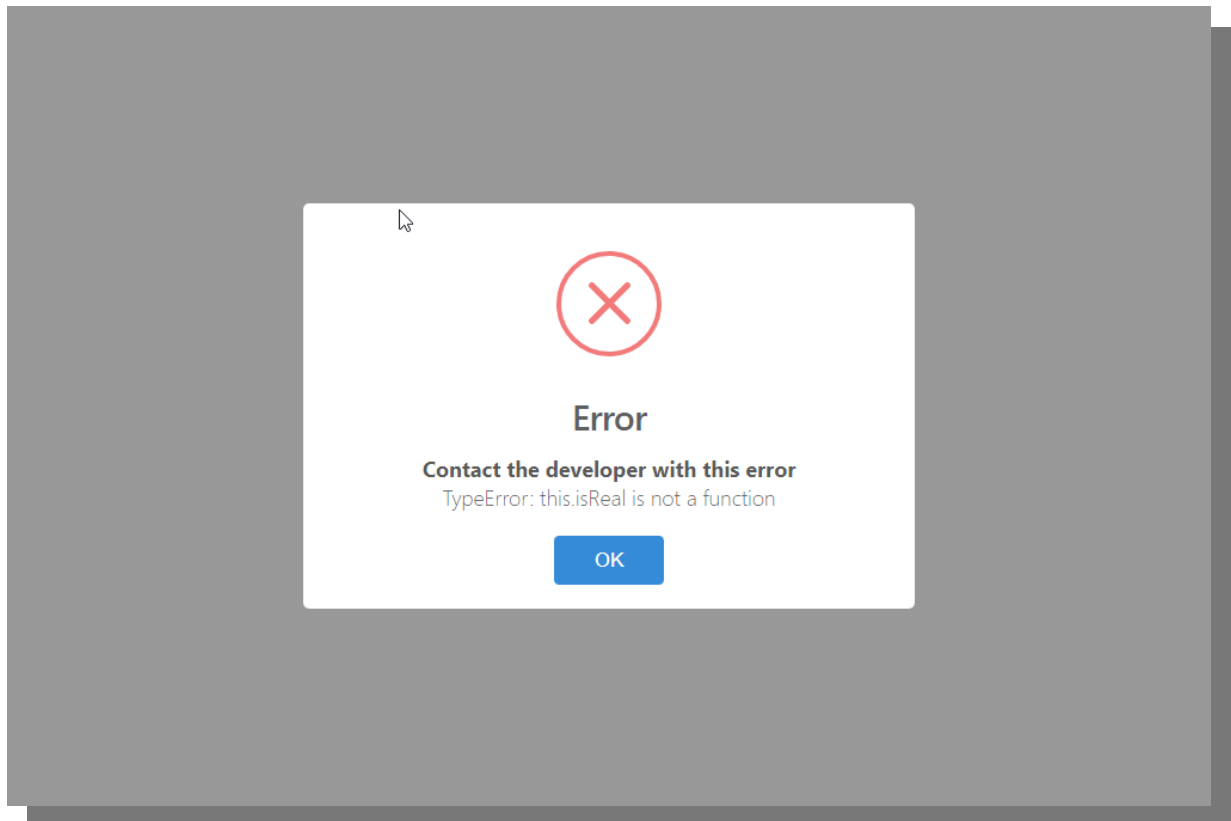


Figure 28 Example error presented using error boundary

This helped me locate an error in my code:



Figure 29 The error in my code

Evaluation

2.5 Testing

3 Conclusion

3.1 Similar product – Cointracker HN

Through the creation of this product it came to my attention that a similar product was just released by the name of “Cointracker” [CITATION Nin18 \l 2057]. 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.

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

3.3 Project Structure / Code Written

3.4 Main program files

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 also 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:

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

4 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](https://commons.wikimedia.org/wiki/File:Spiral_model_(Boehm,_1988).svg)
[Accessed 04 02 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].

Github Inc, 2018. *Trending C# repositories on GitHub today*. [Online]

Available at: <https://github.com/trending/c%23>
[Accessed 04 02 2018].

Github, Inc, 2017. *BSD 2-Clause “Simplified” License | Choose a License*. [Online]

Available at: <https://choosealicense.com/licenses/bsd-2-clause/>
[Accessed 7 March 2017].

Github, Inc, 2017. *ISC License / Choose a License*. [Online]

Available at: <https://choosealicense.com/licenses/isc/>

[Accessed 7 March 2017].

Github, Inc, 2017. *MIT License / Choose a License*. [Online]

Available at: <https://choosealicense.com/licenses/mit/>

[Accessed 7 March 2017].

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].