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.

I initially prepared for the interview with a few questions:

-

1.2 Initial Interview

1.2.1 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

1.2.2 Me

hmm like information on the current exchange rate?

1.2.3 Client

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

bittrex currently have one that I can use on an iOS app

1.2.4 Me

hmmm interesting - I mean it would need to integrate with wallets which would be more complex -> though why not just use a website to look up this stuff?

1.2.5 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

1.2.6 Me

hmmm okay

would be interesting to work on - let me just have a look at the apis out there

1.2.7 Client:

Alrighty

1.2.8 Me:

so I just got bittrix on my phone and I see the market you mean - you sure there's no one of these for windows already?

1.2.9 Client:

They provide an API, which I have found only one app that can use it

There's one company called Delta which could potentially be releasing something

Just wondering what your thoughts on the whole situation were

1.2.10 Me:

I saw hmmm - i'll have a look at making a simple PoC and see how long it takes to integrate stuff together - looks like a fun project - and delta looks pretty good - I'm surprised no ones released a desktop version...

1.2.11 Client:

So am I, I would have thought they would release desktop before iOS or apps

1.2.12 Me:

mhm

I guess mobile is such a big market atm?

[...]

1.3 Design transcript

[...]

1.4 Client Brief

After the initial transcripts the client provided a brief outline of the product:

> *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.*

1.5 MVP Plan

A minimum viable product plan – my interpretation of the client's requirements:

1) Desktop based application
   a) Able to be installed and run from an applications directory.
2) Ability to make a portfolio
   a) Ability to add a wallet/exchange/simple amount of coin
      i) Ability to remove wallet / change simple amount of initial coin
   b) 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) Google Account based
        i) Two factor auth
        ii) Password
5) Analytics
    a) This is to analyse what actions have been taken in the application.
6) Licensing
    a) The client has suggested he only wishes the application to exist. He would be willing to spend money for it. Though has additionally indicated that it would be fine to sell on. For this requirement I would need to introduce a license server, so I can control who is authorised / has paid for the application and who hasn't.

1.6 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:

1.6.1 Coin Ticker iPhone - https://itunes.apple.com/gb/app/coin-ticker-bitcoin-altcoin/id636476147?mt=8

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.

1.6.2 Cryptolio - https://github.com/larion/cryptolio

Terminal based crypto currency portfolio. This has all the information of needed for my specification. The only problem with it is it lacks the interface that a GUI based editor.

1.6.3 CryptoCompare - https://www.cryptocompare.com/portfolio/

1.7 General development model

Throughout the development of this application I have opted to choose 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.

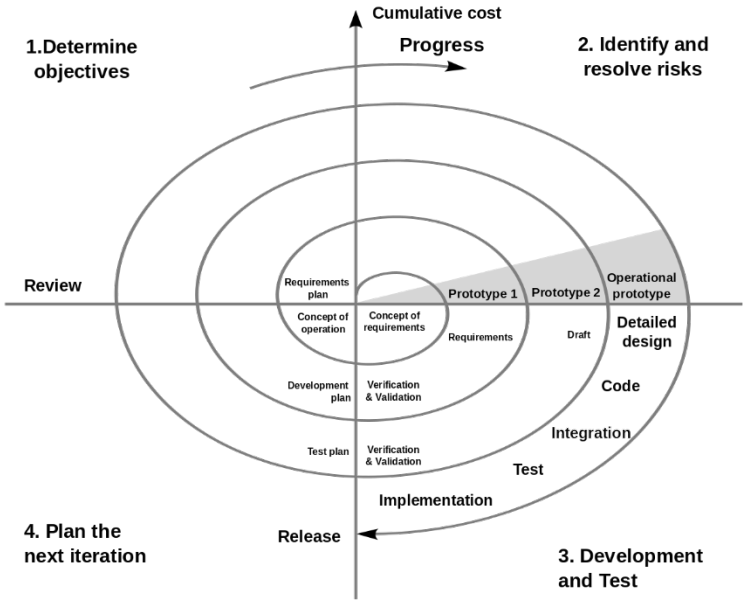


Figure 2 Spiral model development [CITATION Boe04 \l 2057]

1.8 Technologies needed

1.8.1 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 ElectronJSs.

1.8.1.1 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.

1.8.1.1.1 Advantages

+ Well supported/Much documentation
+ Very well used

1.8.1.1.2 Disadvantages

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

1.8.1.2 Java / JavaFX

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

1.8.1.2.1 Advantages

+ Well-structured language made to fit OOP

1.8.1.2.2 Disadvantages

– Learning overhead with the xml language
– Harder to make look native (cannot naturally embed native UI elements)
– 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

1.8.1.3 Electron - https://electronjs.org/

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]

1.8.1.3.1 Advantages

+ Very easy to setup
+ Cross platform
+ Can still access lower level OS features
+ Familiar technologies
+ Open Source

1.8.1.3.2 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)

1.8.1.4 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.

1.8.2 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/

1.8.3 Boilerplate comparison

When creating desktop applications with electron there can be a lot of setup in terms of properly isolating the renderer from the main process (to prevent other programs injecting code). 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. Then there's the problem of managing state in large programs which is generally done through libraries like redux which have direct bindings into Angular or React (see redux-react).

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.

On top of this it has a permissive based license

1.8.4 Note about Licenses

1.8.5 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

1.8.6 Data visualization framework

1.8.6.1 D3.js

https://d3js.org/

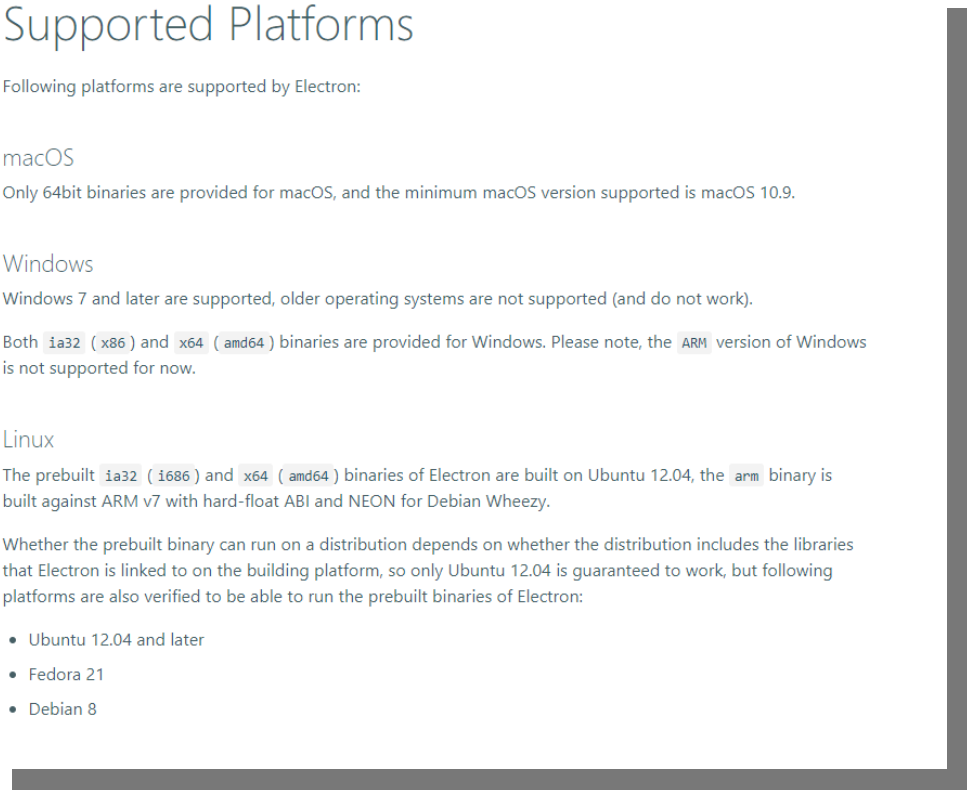
1.8.7 Testing framework

1.8.8 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 3 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].

1.8.9 Conclusion

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

Problems that will be hard to solve

1.9 Basic Layout design

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

Figure 4 – 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:

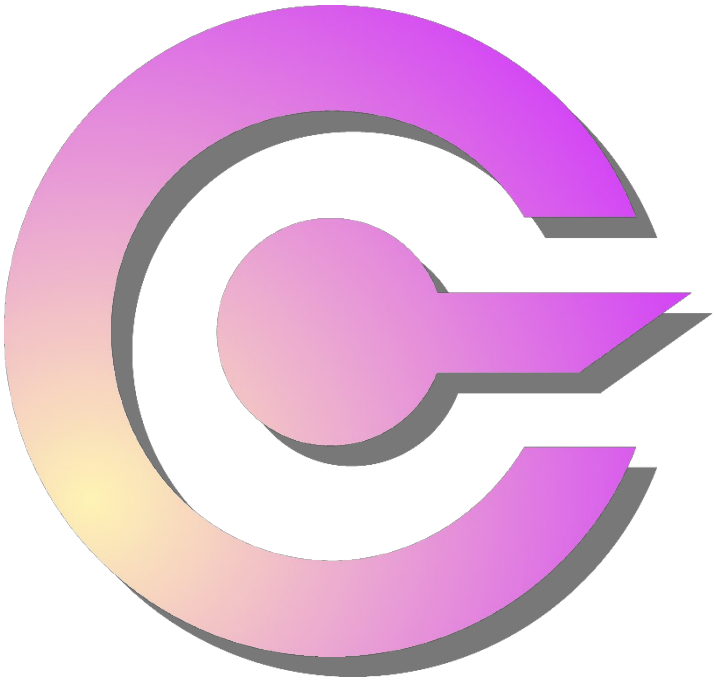


Figure 5 Retouched cryptocurrency logo / New Application logo

Colour Specification for logo:

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

1.10 Tests needed for MVP

Test name	Test Description	MVP Spec
Basic Load	The application loads up	
UI Exists	The UI is present in the rendered application	

1.11 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:

- 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 makes it sound more professional and as effective.

1.12 Problem splitting/Project Diagram

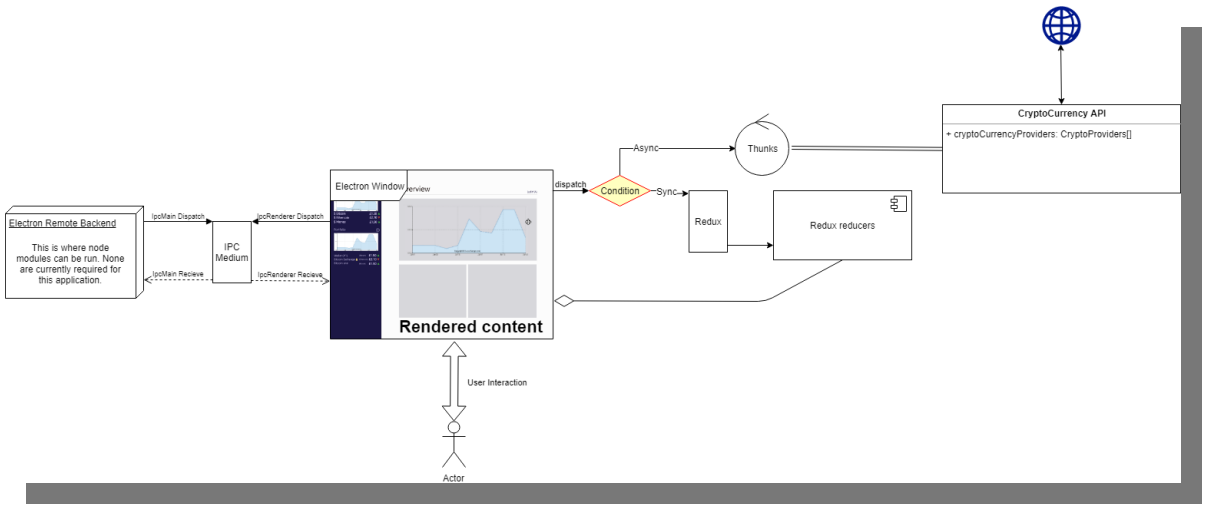


Figure 6 Complete project diagram

#Evaluation of splitting of problem

2 Development

2.1 Testing

2.2 Testing Needed

2.3 Setup

Directory layout:

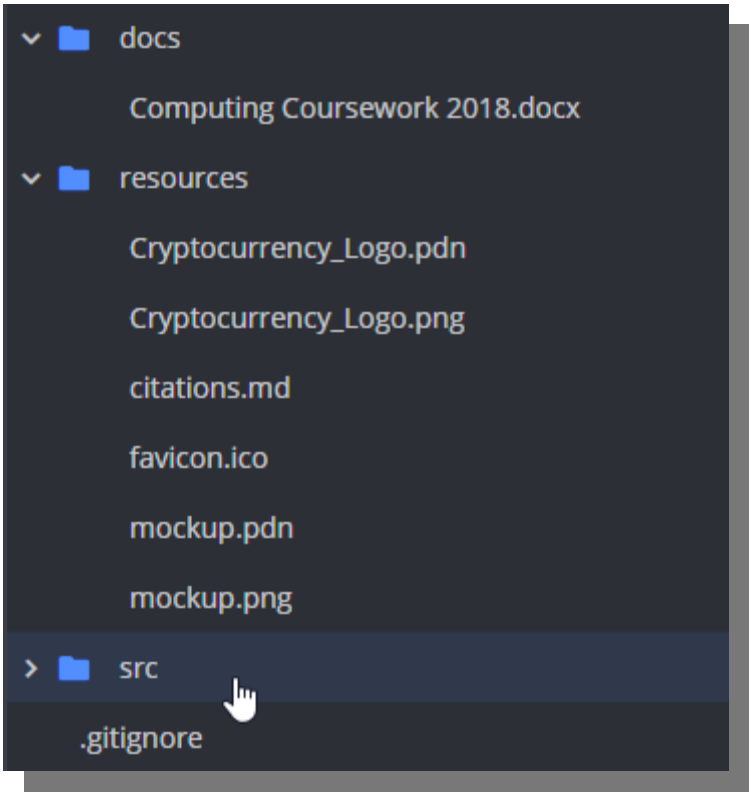


Figure 7 My basic directory layout

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:

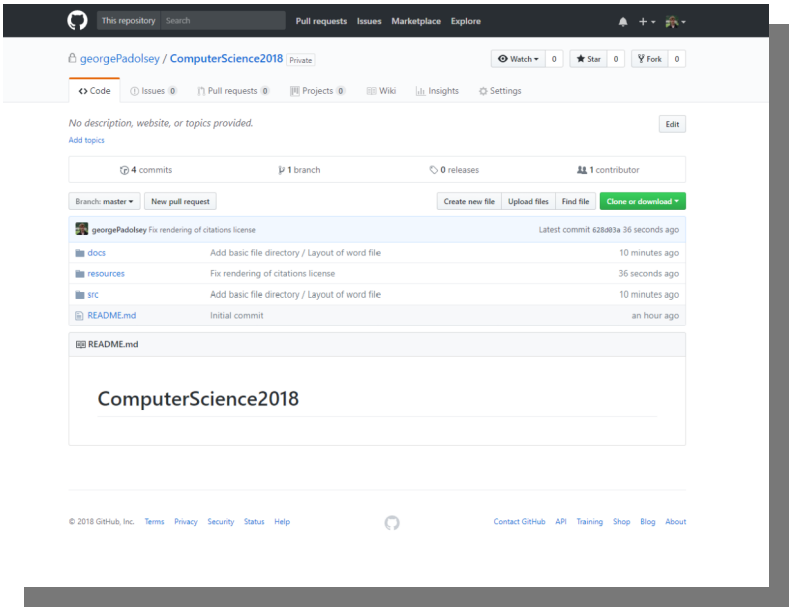


Figure 8 Github repository for the application

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:

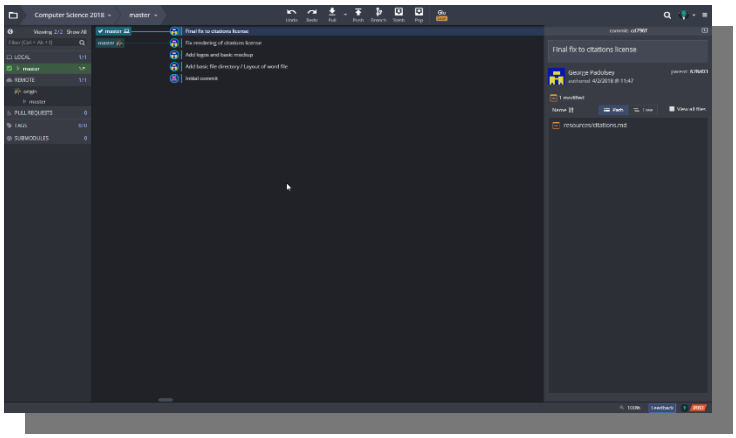


Figure 9 Setting up GitKraken as a version manager

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

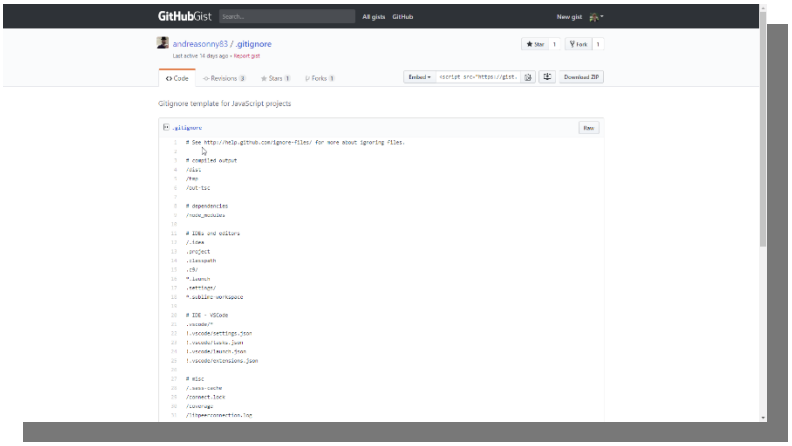


Figure 10 An example .gitignore https://gist.github.com/andreasonny83/b24e38b7772a3ea362d8e8d238d5a7bc

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 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.

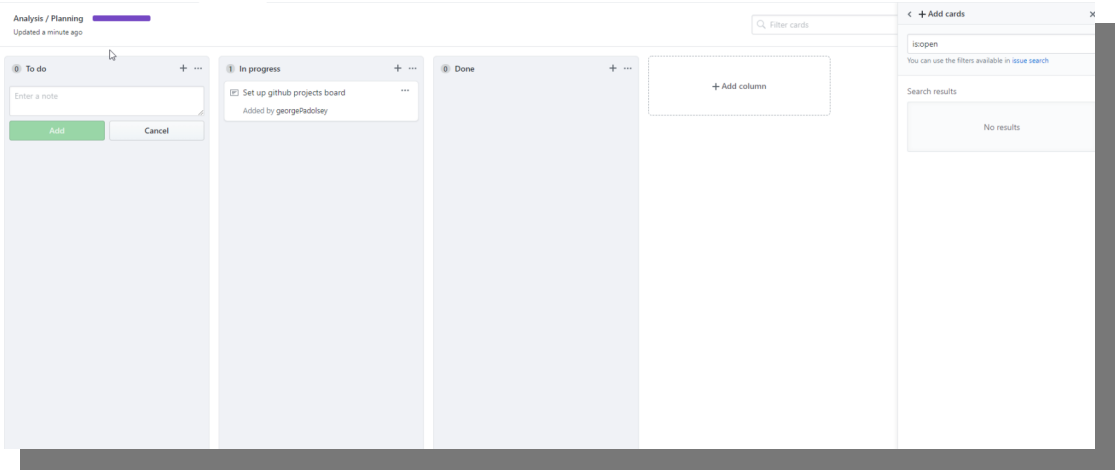


Figure 11 My Github project board for the planning part of the project

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.

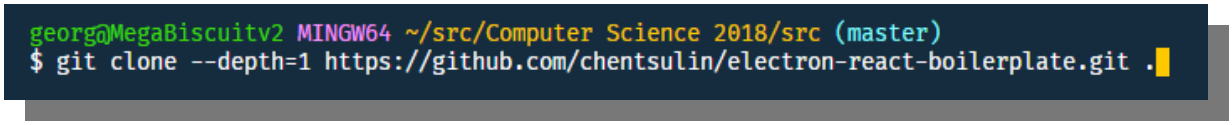


Figure 12 My original attempt at cloning the repository

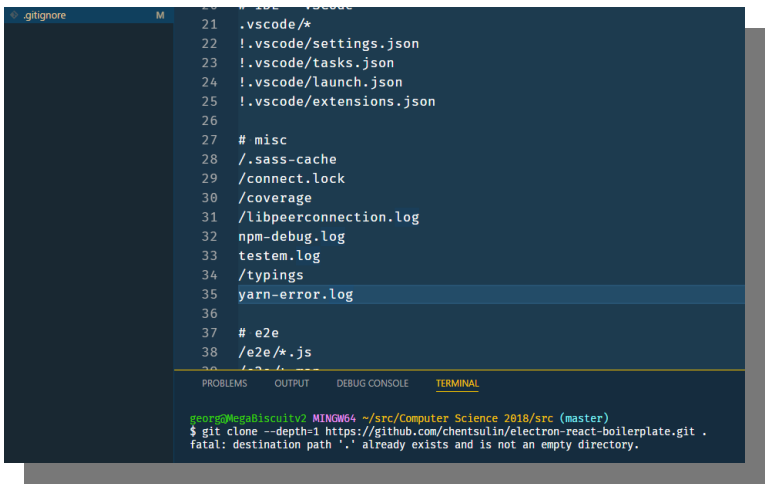


Figure 13 The .gitignore file

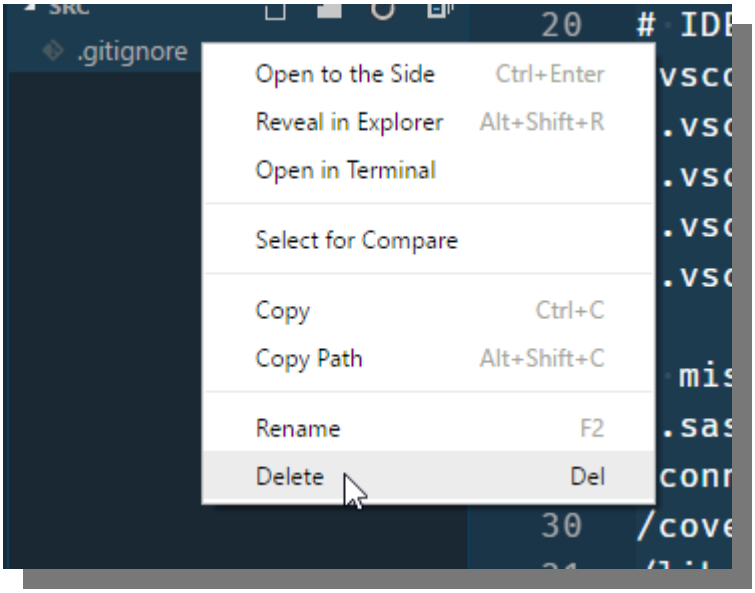


Figure 14 The .gitignore file being deleted.

Finally, I had a fully cloned boilerplate:

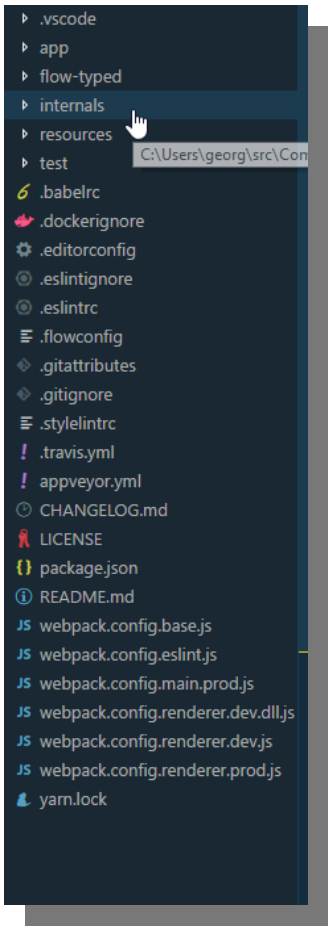


Figure 15 Fully cloned boilerplate

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:

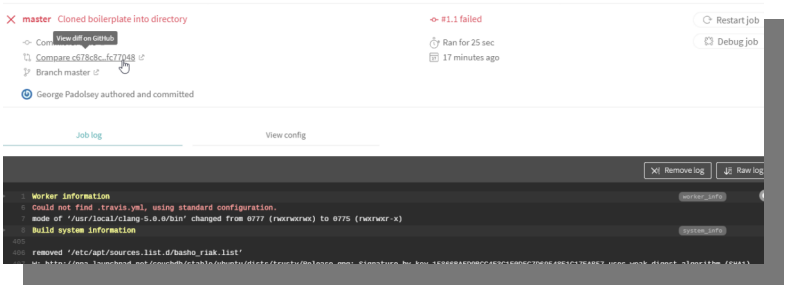


Figure 16 Travis CI error

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.

```
32  before_script:
33    - cd src/
```

Figure 17 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

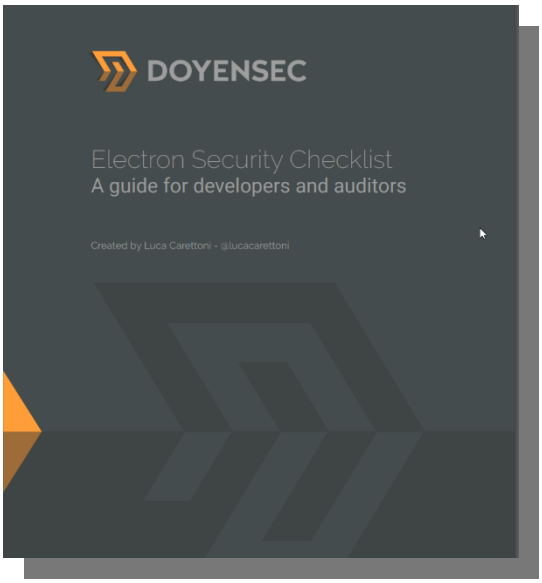
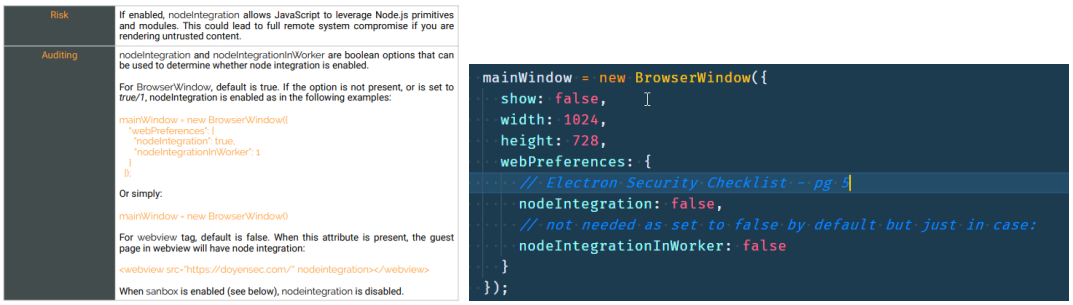


Figure 18 Security Checklist - https://www.blackhat.com/docs/us-17/thursday/us-17-Carettoni-Electronegativity-A-Study-Of-Electron-Security-wp.pdf

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	nodeIntegration and nodeIntegrationInWorker are boolean options that can be used to determine whether node integration is enabled.
	
	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:
	
	`mainWindow = new BrowserWindow({ "webPreferences": { "nodeIntegration": true, "nodeIntegrationInWorker": 1 } });`
	
	Or simply:
	
	`mainWindow = new BrowserWindow()`
	
	For webview tag, default is false. When this attribute is present, the guest page in webview will have node integration:
	
	`<webview src="https://doyensec.com/" nodeintegration></webview>`
	
	When sanbox is enabled (see below), nodeintegration is disabled.

```
mainWindow = new BrowserWindow({
  show: false,
  width: 1024,
  height: 728,
  webPreferences: {
    // Electron Security Checklist - pg 4
    nodeIntegration: false,
    // not needed as set to false by default but just in case:
    nodeIntegrationInWorker: false
  }
});
```

Figure 19 Documentation vs implementation of the checklist

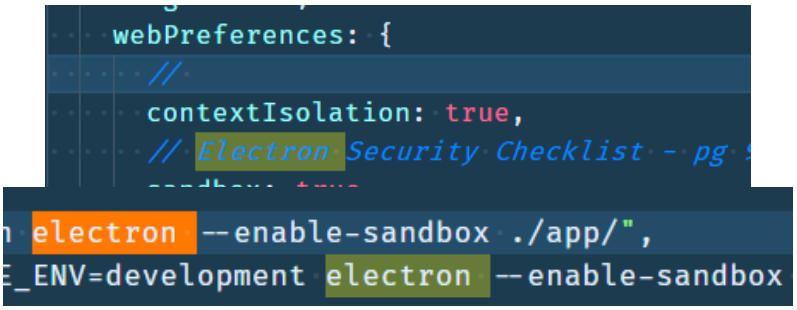


Figure 20 Another example of securing the application – in this case making the build scripts run in sandbox mode [cite]

2.4 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 pleothra 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
	https://www.npmjs.com/package/cosmiconfig	+		MIT[CITATION Git17 \l 2057]
	https://www.npmjs.com/package/properties	+		MIT[CITATION Git17 \l 2057]
	https://www.npmjs.com/package/rc	+		MIT[CITATION Git17 \l 2057] and others
	https://www.npmjs.com/package/configstore	+		BSD 2-clause [CITATION Git171 \l 2057]
	https://www.npmjs.com/package/preferences	+ Allows encryption		MIT[CITATION Git17 \l 2057]
	https://www.npmjs.com/package/config	+		MIT[CITATION Git17 \l 2057]
	https://www.npmjs.com/package/electron-store	+ Can use from renderer / main – no need for ipc transport		MIT[CITATION Git17 \l 2057]
	https://github.com/nathanbuchar/electron-settings	+		ISC [CITATION Git172 \l 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.

3 Evaluation

3.1 Testing

4 Conclusion

4.1 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" [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.

4.2 Project Structure / Code Written

4.3 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.

5 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 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].