Computing Coursework 2018

# Planning

## Client Brief

A brief outline of what the client requires out 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.

## MVP Plan

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

* Desktop based application
* Ability to make a portfolio
  + Ability to add a wallet/exchange/simple amount of coin
    - Ability to remove wallet / change simple amount of initial coin
  + Ability to watch coin gain / fall relative to the initial input
* Lookup current exchange rates
  + Support for multiple exchanges
    - Average
    - Binance
    - Bitflyer
    - Bitfinex
    - Bithumb
    - Bitsamp
    - Bittrex
    - Coinnest
    - Coinone
    - Gdax
    - Geminin
    - Hitbtc
    - Korbit
    - Kraken
    - Liqui
    - Poloniex
    - WEX
  + Allowing changing local currency conversion
* Security
  + Google Account based
    - Two factor auth
    - Password
* Analytics
  + This is to analyse what actions have been taken in the application.
* Licensing
  + 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.

## 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?mt=8>

Coin ticker for iPhone provides many features

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

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

## 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 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.

#### 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)
* 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/>

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

##### Disadvantages

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

#### 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/>

### 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. 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.

### Style choices

### Testing framework

### 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 and internet connection to fetch the data. However, it would be able to run without it and would have graceful degradation of content (W3C, 2015).

### 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 name | Test Description | MVP Spec |
|  |  |  |
|  |  |  |
|  |  |  |

## 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.

# Development

## Testing

## Testing Needed

# Evaluation

## Testing

# Bibliography

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