Software Development Year 3  
Professional Practice in IT Documentation

Paper-Trading Crypto Exchange

John Groves G00367771

David O’Loughran G00362038

Supervisor: Dominic Carr

GitHub Repository link: <https://github.com/johngroves1/Paper-trading-app-SFY3->

Note: Screencast is in main branch of GitHub Repository

# Abstract (John)

This documentation outlines the details of the application for Professional Practice in IT. It shows our design, planning and approach to the web application for a **paper-trading crypto exchange**. This document details the approach used to complete the project in a professional and timely manner.

Table of Contents

[Abstract (John) 2](#_Toc70109768)

[Introduction (John) 5](#_Toc70109769)

[System Requirements (John and David) 6](#_Toc70109770)

[User account (John) 6](#_Toc70109771)

[Access to multiple coins (John) 6](#_Toc70109772)

[Live graph of coins (David) 6](#_Toc70109773)

[Access to market orders and limit orders buy/sell (David) 7](#_Toc70109774)

[Percentages (John) 7](#_Toc70109775)

[Assets overview of all orders and coins (John) 7](#_Toc70109776)

[Technology Used and Why (John and David) 8](#_Toc70109777)

[Amazon Relational Database Service (John) 8](#_Toc70109778)

[PostgreSQL (John) 8](#_Toc70109779)

[Node.js (John) 8](#_Toc70109780)

[Passport.js (John) 8](#_Toc70109781)

[Bcrypt (John) 9](#_Toc70109782)

[TradingView lightweight charts (David) 9](#_Toc70109783)

[Binance API and Web Sockets (David) 9](#_Toc70109784)

[Architecture of the Solution (John) 10](#_Toc70109785)

[Register an account 10](#_Toc70109786)

[Dashboard 11](#_Toc70109787)

[Charts 12](#_Toc70109788)

[Market Orders and Limit Orders 12](#_Toc70109789)

[Assets Overview 13](#_Toc70109790)

[Database 14](#_Toc70109791)

[Design Methodology Applied (John) 15](#_Toc70109792)

[Software Development Life Cycle (John and David) 16](#_Toc70109793)

[Setting up the database – John Groves 16](#_Toc70109794)

[Connecting the database and creating a login system – John Groves 16](#_Toc70109795)

[Using session-cookies and adding user security - John Groves 17](#_Toc70109796)

[Using Socket.io and Binance API for coin price data - David O’Loughran 18](#_Toc70109797)

[Using TradingView Lightweight Charting Library to display price data – David O’Loughran 18](#_Toc70109798)

[Adding wallet table and querying to the database – John Groves 19](#_Toc70109799)

[Adding UI to the application – John Groves 19](#_Toc70109800)

[Market Order Calculations and Queries – David O’Loughran 20](#_Toc70109801)

[Limit Order Calculations and Queries – David O’Loughran 21](#_Toc70109802)

[Limit Order Execution – David O’Loughran 21](#_Toc70109803)

[Cancel Limit Orders – David O’Loughran 22](#_Toc70109804)

[Addition of ETH, XRP, BNB, and ADA added to the site – David O’Loughran 22](#_Toc70109805)

[Limitations and Known Bugs (David) 23](#_Toc70109806)

[Testing Plans (David) 24](#_Toc70109807)

[Recommendations for Future Development (John) 25](#_Toc70109808)

[Deploy Front End 25](#_Toc70109809)

[Two-factor Authentication 25](#_Toc70109810)

[News API 25](#_Toc70109811)

[Adding more cryptocurrency 25](#_Toc70109812)

[Buying/Selling real cryptocurrency 25](#_Toc70109813)

[Setting up credit card details 25](#_Toc70109814)

[Conclusions (David) 26](#_Toc70109815)

# Introduction (John)

This documentation is intended to provide different perspectives on the project. It is intended to provide both user and developer with background information for the system.

The **user is interested in what the project does**, how they interact with it and what the system looks like.  The requirements form part of this section of documentation and will provide detail on the functional components of the system.

The **developer is concerned with how the application is implemented**.  This concentrates on the technical aspects of the user interactions rather than what the system looks like. As part of the development process, the documentation allows the development team to scope out the different technologies that are available to provide a solution to different aspects of system implementation. These technologies can be compared, and decisions made on the merits of each as to whether it is appropriate to use in the development of the new system.

The documentation should provide the following information:

* Architectural Overview of the system.
  + Provide a functional breakdown of the system to be developed.
* Identify and compare the possible technologies, justify the choice of technology to be used in the implementation.
* Database design.
  + Normalise the design for the implementation.
  + Specify data types, field sizes, default values and any constraints on the values to be stored.
* Screen Layout providing a blueprint for how the particular application will look to the end user.
* Each student put their name beside what section of the documentation they wrote, as it indicates in the tables of content
* For the purpose of documenting which developer did what, the software development life cycle will be written from a narrator’s point of view.

# System Requirements (John and David)

## User account (John)

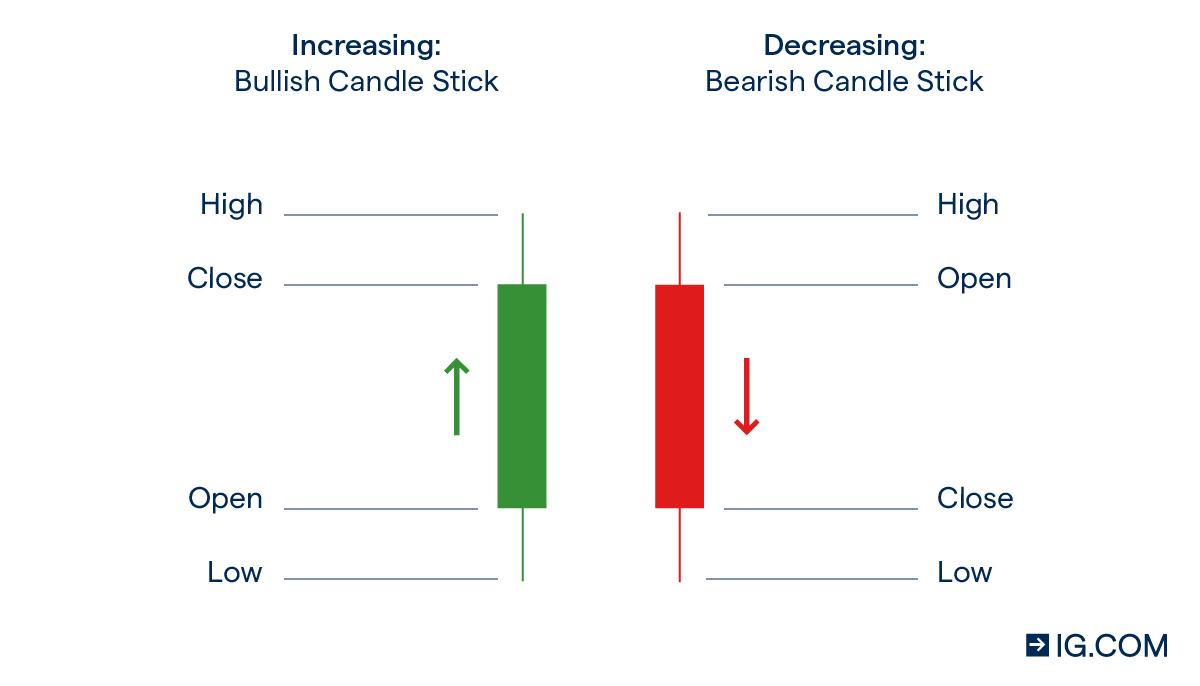
On using our application, the user will be prompt to login or register an account. The user will enter a name, email and password on registering. A user cannot register two accounts under the same email, both entered passwords must match and contain at least 6 characters. Once the user registers an account, they will be able to login under the entered email and password. The user’s password is encrypted before being stored in the database for security purposes.

## Access to multiple coins (John)

The user has access to five coins they can trade from. We decided on providing the top three coins being Bitcoin, Ethereum and XRP. And two upcoming coins in this year, Binance Coin and Cardano (ADA).

## Live graph of coins (David)

The user has access to live graphs of the five coins, displaying accurate historical and real time data of each coin. The user can choose between a one minute, four hours, or daily intervals of each candlestick on the graph. Candlesticks are extremely helpful as instead of showing only the price of one coin they have four different price points for each interval. The price points included are the open price, close price, high price and low price for that interval. The data displayed goes back 2 years.



## Access to market orders and limit orders buy/sell (David)

The user will have access to two different methods of buying and selling coins. Market orders will allow the user to either buy or sell coins at their live price the moment the order is executed. Limit orders on the other hand will allow the user to set which price they would like to buy or sell with the order executing if it hits the specific price or better. For example, if the current price is $55000 for 1 Bitcoin and the user executed a limit buy order of $60000, the users trade would instead be for the market value of $55000 as it is the best available price. All limit orders will then be logged to the database for the users account allowing it to be executed even if the price hits while the user is offline.

## Percentages (John)

The user has access to the price high and low of that day, and access to the 24h% gain or loss of any cryptocurrency. Displaying these figures helps the user judge and decide on when to buy/sell coins and when to put in limit orders.

## Assets overview of all orders and coins (John)

The user has access to the asset overview page which displays all data from their wallet. It displays each coin, the amount the user owns, any limit orders that are in place, the current price of that coin and the amount that is worth in USD. This makes it easier for the user to see their profits and losses on each coin and overall, how they are doing as a crypto trader.

# Technology Used and Why (John and David)

## Amazon Relational Database Service (John)

We used Amazon RDS to host our PostgreSQL database. Amazon RDS fitted the requirements of the application; it was easy to setup, cost efficient and secure. It provided fast high scalability for calling API’s and for querying to the database. It runs on the same highly reliable infrastructure used by other Amazon Web Services, with no downtime or maintenance.

## PostgreSQL (John)

After extensive research on relational databases we decided on using PostgreSQL. PostgreSQL is a powerful, open source object-relational database system with over 30 years of active development. Our main reason for choosing PostgreSQL was because its performance works best in systems which demand the execution of complex queries, we knew our application would use complex queries to calculate market orders, limit orders and percentages. PostgreSQL was easy to learn, free and the GUI element (pgAdmin) was very useful for maintaining and viewing our database.

## Node.js (John)

After doing research on many different types of front-end web development, we decided on using node.js. The reasons for this was mainly to do with its scalability, its capable to handle a huge number of concurrent connections which was especially important for our application since we would be taking in a lot of cryptocurrency data from different connections. Another big reason for choosing this technology was the simplicity of sharing one language for both server and client side, our application is coded heavily server side and using node.js on our front-end to display the finalized data was very useful. This technology is also lightweight and can reduce the application development time while achieving the same functionality, which helped us get immediate feedback from our production environment.

## Passport.js (John)

Passport is an authentication middleware for Node.js. It is easy to use and implement. With the passport-cookie library it uses session-based authentication to utilize browser cookies to manage logged-in and logged-out users.

## Bcrypt (John)

We wanted our application to have some form of extra security for its users. Bcrypt is an easy to use password-hashing function. It is a library for NodeJS which made it easy to install and implement into our application. Each accounts password is hashed 10 times before being stored in the database for extra security. Bcrypt allows you to determine how expensive the hash function will be, because of this bcypt can keep up with Moore’s law. As computers get faster you can increase the has number so decrypting will be slower

## TradingView lightweight charts (David)

For our application we wanted to provide our users with a similar experience to trading on a live crypto exchange. Following research into which charting library would be best suited for our application we realized almost every popular exchange utilized different versions of TradingViews charting libraries. They provide a wide range of easily customizable properties while also allowing data of different formats to be easily inserted, modified and displayed within their charts. We decided upon using TradingViews Lightweight Charts as its capabilities fitted our needs while also being free to use for all developers. It allowed us to easily insert historical and live data into the charts from the Binance API. These charts are also very responsive allowing the user to adjust both the time and price scales while providing the ability to zoom in and out on different sections of the chart using their mouse wheel.

## Binance API and Web Sockets (David)

After researching multiple different api’s for reading in coin price data we concluded that Binance’s api would be the perfect fit for the application. Binance’s exchange is rated the best overall by many due to its low fees matched with the most volume for any crypto exchange. The vast majority of traders have a Binance account allowing for a seamless transition from our application to a live cryptocurrency exchange. Binance API allows us to retrieve price data for a multitude of different currencies accessing both live and historical data to update our TradingView Lightweight Charting library. Binance has its own libraries for nodeJS with high quality documentation which made it an obvious choice to easily integrate within our application using web sockets to listen to a wide variety of different data streams on a single port. We used the Socket.io library due to its simple integration for nodeJS and its ability to work with any platform, browser or device. The combination of nodeJS and Socket.io is perfect for our application as it is built on the premise of seamless scalability which is crucial to our applications need or retrieving substantial amounts of price data for each coin.

# Architecture of the Solution (John)

## Register an account

Graphical user interface, application

Description automatically generated

On entering our web application, the user is prompt to register an account. Once registered the user can login and access our application.

## Dashboard

Graphical user interface, application

Description automatically generated

Once the user has logged in, they are redirected to the dashboard. This page gives a brief summary of what our application does and below that it shows a live table of each coin’s latest price and percentage change in the last 24hours. On the navigation bar the user can select the drop-down list of coins which will redirect the user to the selected chart of that coin. To the right of the navigation bar the users is displayed, for this example it is “demonstration”, by clicking on demonstration the user is redirected to the asset overview page. The user also has the option to logout of their account on the navigation bar.

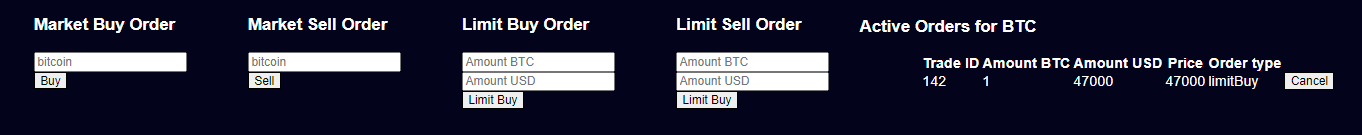
## Charts

Chart, histogram

Description automatically generated

On selecting a coin from the drop-down list, you will navigate to the chart page, where a large range of data is displayed. The current price of the coin is displayed in dollars at the top of the chart, below that the 24hr percentage change is displayed and beside that the 24hr highest price and lowest price are displayed. The user can change the time intervals on the graph to 1minute, 4hours or 1day by selecting the buttons displayed, the graph goes back 2 years so the user can view previous patterns of the coin. Below the chart each account has 100,000 USD to trade with and below that shows how much coin that user owns.

## Market Orders and Limit Orders



On the same page below the chart is where the user can select their orders. They have the option to buy and sell at market price or to put in a limit buy/sell order. The user will specify how much of that coin they want to buy or sell, if the user has not enough funds in their wallet the order won’t execute. For limit orders the user must specify how many coins they want and at what price they want to buy, to the right of the input boxes you can see an active order for 1 Bitcoin at the price of 47,000. If the price of bitcoin drops to 47,000 the order will be executed, and 1 Bitcoin is added to the user’s wallet, the user has the option to cancel any limit order they set.

## Assets Overview

Graphical user interface, application

Description automatically generated

By selecting your name on the navigation bar, the user is redirected to the asset overview page where it displays the users entire trading wallet. Each coin is displayed showing how much of that coin you own, what orders are in place, the amount of USD you own in that coin, the live price of that coin and a trade link to navigate back to the chart. As you can see above the user has a limit order of 1 bitcoin in place right now and the user owns 100 XRP at the price of 1.08USD so their wallet amount in XRP is 108USD. All data of prices are live and update every second.

## Database

A picture containing table

Description automatically generated

Above is a schema of our database, every user has a wallet and every wallet has a trade. The user table stores the credentials of their account, being a unique ID, their name, email and encrypted password. The wallet table stores each coin on of the users account as well as the USD balance the user can trade with, each coin is of a decimal value as some coins can go to 100 of digits after the decimal point. The wallet references the primary key of the user ID and autoincrements its own wallet ID. The trades table stores all forms of limit orders, it specifies the amount of the coin, the price you set the limit order at, an enumeration of which coin selected, the order type being a limit buy or limit sell order and a time integer of when the order was placed. The time that the order is placed is stored into the trades table to determine in the future if that price had hit, by comparing the live price to the limit price of that date. For example, if a user placed a buy order of bitcoin for 47000 on the 17th of April and five days later that price hit, the order will be executed. The trades table autoincrements its own ID and references the primary key of the wallet ID.

# Design Methodology Applied (John)

The Design methodology we used for our project was the Iterative SDLC Model, our reasons for choosing this design methodology was because we knew our application would end up being quite complex with mathematical calculations and queries to the database. One of the main advantages to using this methodology was that we wanted to have a working model of our application at a very early stage of development, which made it easier to identify functional or design flaws as we progressed. Our requirements for the final product of our application were predefined as all crypto exchanges would have market orders, limit orders and calculated profits and losses. We knew from the start that we would be building up our charts page from just displaying a basic graph of data to eventually executing complex tasks.

We initially put a large amount of work into developing the functional requirements of our application, being the charts page and the login functionality. We spend a week separately working on these features making sure all the basics worked, each of our initial commits were quite large and in retrospect we should have divided those large commits into several smaller commits as if would have been easier to track our work from the start.

Once we developed the basic functional requirements to the application, we developed and pushed in shorter iterations. Doing this made it easier for testing and debugging each equation and query we added, anytime we had problems or risks from the last iteration, it was easy to define and prevent in the next commit. Using this methodology also made it easier to control the high-risk tasks that were later developed in our application such as the limit orders.

We also made use of the project section on GitHub to track our ideas, what functionality was in progress and when it was completed. Having a meeting with our supervisor every two weeks was great for keeping us on track with our commits and making sure we were on track for the project submission deadline.

# Software Development Life Cycle (John and David)

At the start of this project the developers quickly decided on making a paper trading crypto-exchange application as they both had a large interest in cryptocurrencies. They initially spent the first two weeks doing extensive research of what technologies to use in the application. While researching they kept in mind what database would suit our application best, what front end would work efficiently for displaying large sums of data and how to pull live data of individual coins into our application. Once each technology was decided and a plan was laid out, the development process began.

## Setting up the database – John Groves

The developer John Groves worked on setting up the PostgreSQL database. After doing research and watching an in detailed [YouTube tutorial](https://youtu.be/qw--VYLpxG4) on PostgreSQL by [freeCodeCamp.org](https://www.freecodecamp.org/), John set up the project database and created a superuser with all access and privileges to the database, he also created a user’s table with the appropriate datatypes. After this, the next step was to figure out how to host the database to a server, the options came down to using Amazon AWS or Google Cloud platform. He decided on using Amazon AWS as it was a new technology since he has previously used Google Cloud platform for another module and Amazon AWS had great reviews for being easy to manage, free pricing and had clear and concise documentation. On the Amazon AWS website there is a user guide for hosting a PostgreSQL database which was easy to follow, once the database was hosted John had the other developer David to locally access the database to see if he could view the users table from his machine and he could. The next stage in development was to setup our web application.

## Connecting the database and creating a login system – John Groves

Both developers felt that connecting the database to the web application was the first step that should be taken. Since John was already familiar with PostgreSQL, he worked on setting up the environment and database configuration. Connecting the database to the web application was not that difficult as the developer had previous experience from past modules, the next step and to confirm if the database was communicating between the server and the front end, John began to code a login functionality.

He started with a basic login/register page where the user is prompt to enter their name, email, password and password confirmation. Once entered, the form will send the data to the server file which writes a query to the database inserting that data. Error handling was then added to each input box to make sure the user enters all fields. The user could only enter an email if it has the right characters such as ‘@’ symbol and could not register two accounts under the same email, this was done by querying to the database and checking to see if an email already exists in the database. Both passwords had to match too and had to be at least 6 characters long.

## Using session-cookies and adding user security - John Groves

John then went onto researching Passport.js, an authentication middleware for Node.js that had many libraries available to download. One of such libraries was passport-cookie, a cookie authentication strategy for passport. After reading the documentation on GitHub and installing the package into the application, John was able to set up local session cookies to save your data to the browser. When the user logs in and if the entered email and password matches to the database, the users ID and all other details are stored in a local session-cookie. Now if the user tried to navigate to the login or register page once logged in, by changing the website address in the search bar, the user will be navigated back to the homepage.

Initially the reason for researching Passport.js was to use the passport-auth0 library to add two-factor authentication on logging in to the application. Nearly all crypto exchanges would have some form of two factor authentication to add extra security to the user’s account. The application doesn’t trade with real crypto but both developers wanted to create as accurate of an exchange as they could. After spending several days trying to incorporate the auth0 library into the application it became clear how hard it would be to implement this feature as now each user would have to register with a valid email to receive an authentication code.

The developer went back to researching other forms of user security and decided on using bcrypt, a password hashing function. Bcrypt is a library for Node.js so it was easy to install and the documentation was clear and concise. Now if a user registers an account the entered password is passed into a hashing function and the encrypted password is saved to the database. Bcypt allows you to choose a balance of speed and security as you decide on the hash value, for the purposes of the application John decided on using a hash value of 10. So now if a user entered a six-character password, its hashed password that’s stored in the database is now 60 characters long and is virtually uncrackable.

## Using Socket.io and Binance API for coin price data - David O’Loughran

David began researching different methods and apis to be used for retrieving both historical and real-time data to be displayed and manipulated throughout the application. He discovered that Binance a reliable and well renowned cryptocurrency exchange provided free access to price data for each coin listed on their exchange. Following this discovery, he began investigating their documentation where he seen it had seamless integration for nodeJS applications making it the perfect fit.

The developer began utilizing the node-fetch library to retrieve historical data from the urls listed within the binace api documentation. These URLs are optimized for modification allowing him to access historical data for multiple coins and different time intervals by changing the parameters within the url. As the method of displaying the data to a chart was not yet researched, he logged all necessary historical price data to the console for Bitcoin to ensure it was working correctly.

After confirming the historical data was being logged correctly, he moved on to retrieving real-time data from the binance api. From his reading of the Binance API documentation he discovered web sockets were the most efficient way of retrieving real time data. This is where he discovered Socket.io and it’s easy to use nodeJS libraries which work on all platforms. The documentation was easy to follow so he began writing a nodeJS script file to retrieve real-time data from the api. There were initially a few difficulties setting up the socket connections, but this was quickly resolved as there was just an issue with repeated variables for the different socket connection. Using the documentation, he was able to set up a live socket connection using on port 3000 to retrieve live data for Bitcoin on the daily time interval receiving KLINE data which consists of the opening time, closing time, opening price, closing price, highest and lowest prices for the current day.

## Using TradingView Lightweight Charting Library to display price data – David O’Loughran

After correctly logging the necessary price data to populate candlestick charts he researched the different ways of creating and displaying charts with candlestick data. His own experience in the world of cryptocurrency's led him to first checkout Tradingview as a feasible option due to their popularity among the top exchanges. This proved to be the case as Tradingview offered three different charting libraries with high quality documentation on how to customize and insert candlestick data to be displayed. He went for the TradingView Lightweight Charts Library as it satisfied our applications needs while also being free to use for all developers.

With the Binance candlestick data already being logged correctly he got to work on familiarizing himself with the different chart properties. One of these was the update function where was able to instantly create and populate a chart with the ArrayList retrieved using the fetch request on the apis URL for Bitcoin. There were some initial issues displaying the opening and closing times due to unfamiliarity with the Epoch timescale. But after a little research he found it was a simple calculation to convert to readable data and insert it into the charts setData function. After this change the historical data for Bitcoin was displayed correctly to a responsive candlestick chart.

He was pleased to find that there was function to update the candlestick chart which could be fed the live data being emitted on port 3000 from socket.io. Socket.io’s documentation proved to be extremely useful as without any issues the live data was retrieved from port 3000 using socket.on. This kept the stream of real-time data active on the client side while the page was running and the .off function was not called for the socket in use. This allowed for the chart to be updated in real-time without effecting the historical data as it would update each candle based on the epoch time from the live socket overriding the price of only the most recent candle stick until a new opening time was reached at the end of the day.

## Adding wallet table and querying to the database – John Groves

Once David had added a live chart displaying Bitcoin data, the next stage in development was to set up a wallet table in the database to store the user’s assets. The developers had planned that every user would have one wallet, so as John set up the wallet table, he referenced the primary key of the user table to create a 1-to-1 relationship between both tables. For the purpose of testing, and to keeps things simple at this stage of development, the wallet table only stored bitcoin values as it was the only chart currently displayed. Once the table was set up in the database, John went on to testing queries to see if he could display data from the database onto the front-end of the application. Having node.js as both front end and back end it made it easy and efficient for displaying data to the application as EJS has simple template tags for displaying database data.

## Adding UI to the application – John Groves

After setting up the required tables in the database, John went on to focus on the UI aspects of the application. Firstly, he focused on cleaning up the login/register page, by using bootstrap style sheet he was able to create a template for the login inputs with a clean simplistic aesthetic. He then used photoshop to add the application title to a trading stock image and added it to the page.

He then went on to setting up a navigation bar linking each page together and tried to keep a common theme among each page. After which he added an assets page, the idea of this page was to display all the user’s wallet data. The user should be able to see how each of their investments are doing, and how much of each they own. John set up a table with an image of each coin used in the application, he then displayed the amount of Bitcoin the user owned from the database and the value they owned. He also added an order column to display any orders of each coin, but the functionality of limit orders had not been implanted yet. Then with the help of David, he set up sockets to read in and display the current value of Bitcoin.

The last focus on UI features in the application was the dashboard page, when the user logs in they are directed straight to the dashboard page. So, John wrote a short explanation of the application and instructions on how to navigate to each chart page and the assets overview page. He then displayed the live price of each coin implemented in the application and the 24hr percentage change in price so the user could see how well or poor each coin is performing.

## Market Order Calculations and Queries – David O’Loughran

After John set up the wallet table for the database, both developers got to work on setting up the market order functionality for Bitcoin. David created input boxes on the chart page for Bitcoin to allow the user to insert the exact number of Bitcoin they would like to purchase at market value, While John helped server-side with queries to insert the newly acquired into the database. Using the live data from the web sockets to retrieve the current market price, David then created a post request using a from in EJS to the server so the amount of bitcoin entered by the user along with the coins name, current price, order type and the users id would now be available server-side so calculations could be made to insert the correct data into the users wallet . For market orders there is only one calculation needed, which is to multiply the price the order was executed at by the amount of bitcoin specified. Once the calculations were finished John helped David with setting up queries to the database to update the wallet correctly depending on whether a market buy, or market sell order was submitted by the user. Market buy would subtract from the users USD balance in the wallet and add the amount of bitcoin bought to the database and the opposite of a sell order was executed.

After they got the correct values inserted into the database for both market orders, David added validation checks to make sure the user either had enough USD for buying and enough Bitcoin for selling were added displaying an error message to the user if their balance was insufficient to complete the order.

## Limit Order Calculations and Queries – David O’Loughran

With the market orders set up and working in conjunction with the database David began working on the set up of limit orders as he has more experience with trading on Cryptocurrency Exchange’s. First, he set up forms for both Limit Buy orders and Limit Sell orders to submit a post request to the server containing the necessary values to be inserted into the database. Input boxes were added to the form so the user can enter the price they would like the order to execute and the amount of Bitcoin they would like to buy or sell at that price. From his knowledge of limit orders from trading on live exchanges he put checks in place to ensure the users order would match market price if for limit buy orders if the users specified price was more than Bitcoins current market price or for limit sell orders if the users price input is lower than bitcoins market price. Included but hidden in the form is the users wallet id, the coin symbol, order type and the time submitted. Just like with market orders, if the user does not have sufficient amount of USD or Bitcoin in their wallet an error message will be displayed.

David then set-up a trades table with a one to one relationship between the wallets table and the new trades table using the users wallet id as the reference for each user. David then set up queries to insert the details from the limit orders post requests into the trades table based on the users wallet id. After the trades inserted into the database correctly He added a query to update the users wallet for each order. Limit Buy, the execution price specified is multiplied by the amount of Bitcoin entered by the user and subtracted from USD in the users wallet. Limit Sell, the amount of Bitcoin specified by the user is subtracted from Bitcoin in the users wallet.

## Limit Order Execution – David O’Loughran

Once both Limit Orders were correctly inserted into the trades table David started working on the calculations to check if a previously submitted trade had been executed for the user. He began by using the fetch request to retrieve Bitcoins Kline (candlestick) data from the binance api for the past 2 years. When the user logs in the server queries the trades table in the database and compares the users trades execution prices, order type, order time and the amount of bitcoin ordered for each trade. ForEach statements are used to loop through both Binances API price data for Bitcoin as well as the results from the trades table to check if the price has been hit. He achieved this by comparing both ArrayLists to check if the price was hit and the time the order was submitted is more or equal to time it was hit. If these requirements are hit for a Limit Buy order, the amount of bitcoin specified by the user will added to their wallet and for a Limit Sell order, the amount of USD is added to the user’s wallet.

## Cancel Limit Orders – David O’Loughran

After the limit order functionality was completed and trades where the price was hit gets executed, David added a table to the charts page to display active trades for Bitcoin with a cancel button submitting a post request to the server. This post request would have the trade ID along with the amounts the trade is for along with the wallet id and order type. He then created queries to update the wallet and remove the order from the trade table in the database.

## Addition of ETH, XRP, BNB, and ADA added to the site – David O’Loughran

David repeated the steps outlined above for each different coin available on the website. The coins were already set up in the database so changes to variables and queries were all that was needed to add each coin to the website.

# Limitations and Known Bugs (David)

* Binance API is limited to approximately 2 years of price data so data displayed in charts does not show a full history of the coin.
* Bug that allows user to cancel a trade after it has been executed if page is not refreshed.
* UI features don’t scale to page if the page is shrunk.

# Testing Plans (David)

While setting up the Limit Orders Execution, David attempted multiple tests to ensure if the users specified price was hit after the time the order was submitted the trade order would be executed. This was done by setting up a Limit Buy Order for BTC on April 16th for 3 Bitcoin if the price hit $55000 while the current price was $61235. On April 18th Bitcoins price dropped to $50000 and the users order was executed updating the user’s wallet correctly and removing the trade from the database.

# Recommendations for Future Development (John)

## Deploy Front End

Hosting our node.js frontend to an Amazon AWS server would be one of the first future developments, it was our initial plan to host our application, but we ran out of time coming up to the submission deadline.

Two-factor Authentication

Adding two-factor authentication to the users email and phone number for an extra layer of security would be a huge benefit to the application.

## News API

Adding a cryptocurrency news API to our homepage that would display a large range of information regarding new coins or any updates on current coins.

## Adding more cryptocurrency

Our application having five coins limits the users learning experience for trading cryptocurrency, in the future we would like to add more coins as they were easy to implement.

## Buying/Selling real cryptocurrency

With Binances API it is possible to buy real cryptocurrency. It is very complex and hard to understand but in the possible future we could try and incorporate it into our application.

## Setting up credit card details

Setting up a working credit card system so the user can purchase cryptocurrency.

# Conclusions (David)

In conclusion to this documentation we approached from a user perspective and a developer perspective, in our system requirements we outlined what a user would expect from a crypto-currency exchange. In our Software development life cycle, we showed a technical aspect of the user interactions on our application and how we researched and implemented new technologies. We provided a functional break down of our system, talked about our methodology, our plan and how our database was implemented.

In conclusion to this module we learned many new skills and practices. One of the most important skills we gained from this module would be the ability to self-learn a new technology. To be able to understand the documentation and implement the technology into our application, almost all aspects of our project were using new technologies and it shows clearly the skills we have developed over the past two years as we become software developers. We also learned how to work well together in a project, by dividing the work fairly and communicating between each other regarding our git commits and any problems that arose. We followed a methodology, did extensive research from the start and had a concise plan, in retrospect it was very beneficial as it kept us on track with our project and submission.