Software Development Year 3  
Professional Practice in IT Documentation

John Groves G00367771

David O’Loughran G00362038

# Abstract

Summary of document and what our application does. A web application for crypto-exchange

Table of Contents

[Abstract 2](#_Toc70020354)

[Introduction 5](#_Toc70020355)

[System Requirements 6](#_Toc70020356)

[User account 6](#_Toc70020357)

[Access to multiple coins 6](#_Toc70020358)

[Live graph of coins 6](#_Toc70020359)

[Access to market orders and limit orders both buy and sell 7](#_Toc70020360)

[See profits and percentages 7](#_Toc70020361)

[Assets overview of all orders and coins 7](#_Toc70020362)

[Technology Used and Why 8](#_Toc70020363)

[Amazon Relational Database Service 8](#_Toc70020364)

[PostgreSQL 8](#_Toc70020365)

[Node.js 8](#_Toc70020366)

[Passport.js 8](#_Toc70020367)

[bcrypt 9](#_Toc70020368)

[TradingView lightweight charts 9](#_Toc70020369)

[Binance API 9](#_Toc70020370)

[Architecture of the Solution 10](#_Toc70020371)

[Front End 10](#_Toc70020372)

[Database 10](#_Toc70020373)

[Design Methodology Applied 11](#_Toc70020374)

[Software Development Life Cycle 12](#_Toc70020375)

[Setting up the database – John Groves 12](#_Toc70020376)

[Connecting the database and creating a login system – John Groves 12](#_Toc70020377)

[Using session-cookies and adding user security -John Groves 13](#_Toc70020378)

[Limitations and Known Bugs 16](#_Toc70020379)

[Testing Plans 17](#_Toc70020380)

[Recommendations for Future Development 18](#_Toc70020381)

[Conclusions 19](#_Toc70020382)

# Introduction

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.

Note

# System Requirements

## User account

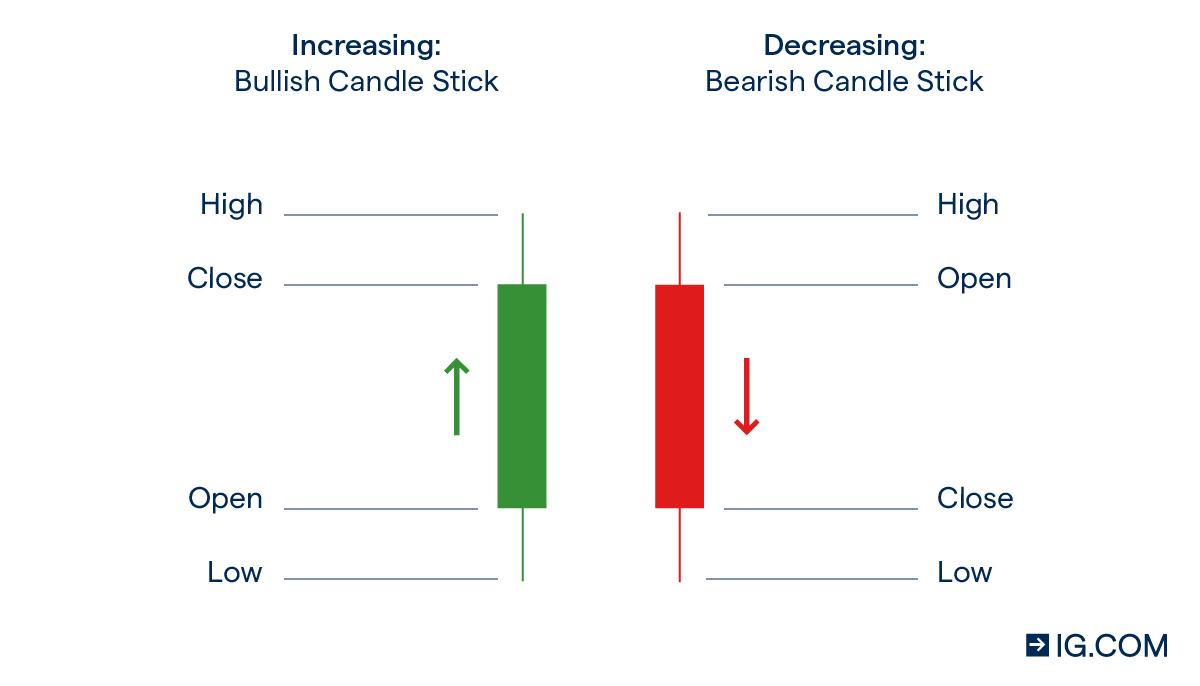
On using our application, the user will be prompt to login or register an account. The user will enter a name, email and password on registering. A user cannot register two accounts under the same email, both entered passwords must match and contain at least 6 characters. Once the user registers an account, 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

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

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. (leave to Dave for more technical analysis). The data displayed goes back 2 years.



## Access to market orders and limit orders both buy and sell

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.

## See profits and percentages

Maybe

## Assets overview of all orders and coins

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.

# Technology Used and Why

## Amazon Relational Database Service

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

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

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

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

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 u with Moore’s law. As computers get faster you can increase the has number so decrypting will be slower

## TradingView lightweight charts

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

For our application we wanted to p

# Architecture of the Solution

## Front End

Talk about each page and layout of them

## Database

Talk about data types / relationship between tables

# Design Methodology Applied

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

For the purpose of documenting which developer did what, this section will be written from a narrator’s point of view.

Note write about technologies that did not work as expected/ wrong approach taken

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 the users 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 want 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.

Database: user table for accounts also made master user for us to change table

Host to AWS

Login system with error handling etc

Added session cookies + encrypted password

Added sockets and apis pulling coin data

Set up charts

Added navbar

Database: wallet table linked to user

Focus on queries and math calculations

Added asset overview which displays all coin data from wallet database

Cleaned UI for chart page

Database: trade table set up for limit orders

More maths + query calculations

Completed Limit orders

Added other coin

# Limitations and Known Bugs

Fixed any bugs/typos/UI features

# Testing Plans

Test on limit orders

# Recommendations for Future Development

Two-factor Authentication for extra security

Buy/Sell real crypto

Add more coins

Credit card details

Deploy to front end

Stop loss

News API

Leverage Trading

# Conclusions