



# Cryptotracker

Jacob Sickels

# Table of Contents

<b>Project Proposal</b>	<b>4</b>
<b>Deliverables</b>	<b>4</b>
<b>Project Plan Overview</b>	<b>5</b>
<b>Use Cases</b>	<b>6</b>
Use Case #1.0 - User Opens App	6
Use Case #2.0 - User Logs into App (when not authenticated)	6
Use Case #2.1 - User Logs into App (when authenticated)	7
Use Case #3.0 - User Navigates to Dashboard Page	7
Use Case #4.0 - User Navigates to Exchange Page	8
Use Case #4.1 - User Edits Amount Field on Exchange Page	9
Use Case #4.2 - User Edits Base Currency on Exchange Page	9
Use Case #4.3 - User Edits Conversion Currency on Exchange Page	10
Use Case #5.0 - User Navigates to Filtering Page on Specific Currency	10
Use Case #5.1 - User Edits Date Range on Filtered Currency	11
Use Case #5.2 - User Clears Date Range on Filtered Currency	12
Use Case #6.0 - User Navigates to Account Page	12
Use Case #6.1 - User Edits Base Currency on Their Account	13
Use Case #7.0 - User Logs Out of App	13
Use Case #8.0 - System Updates Crypto Data Once an Hour	14
Use Case #9.0 - System Updates Exchange Values Once an Hour	14
<b>Table Structure</b>	<b>16</b>
Crypto_data	16
Exchanges	17
Users	17
<b>Architecture</b>	<b>18</b>
React	18
Redux	18
Redux Diagram	19
Firebase	22
Cron	22
Data Flow Diagram	23
<b>Client-Side Routing</b>	<b>24</b>
<b>Testing Plan</b>	<b>26</b>
	2

<b>Testing Suite</b>	<b>26</b>
Jest	26
Enzyme	29
Use Case Actualization	29
<b>Dependencies</b>	<b>37</b>
Webpack	37
Yarn	37
Moment and Moment Timezone	37
React-Materialize	38
Other Libraries	38
<b>Resources</b>	<b>39</b>
<b>Source Control</b>	<b>39</b>
<b>Code Highlights</b>	<b>40</b>
App.js	40
Actions	41
Reducers	46
Components	49
Selectors	65
AppRouter	68
Tests	68
<b>User Manual</b>	<b>69</b>
Location	69
Terminology	69
Functionality	69
Pages	69
Login Page	69
Dashboard Page	71
Filter Page	72
Exchange Page	74
Account Page (and Logging Out)	75
<b>Project Journal and Timesheet</b>	<b>77</b>
<b>Reflection</b>	<b>81</b>
<b>Capstone Evaluation Rubrics Self Evaluation</b>	<b>84</b>

# Project Proposal

I want my focus for this project to be about web development and **cryptocurrencies**. A couple of months ago, a steep rise in Bitcoin piqued my interest in cryptocurrencies, and ever since then, I have wanted to track their progress. So, for my project, I want to create a web app that tracks cryptocurrency prices as well as gives the user data about when it is/was a good time to buy and sell them. In addition, I found an API that gives up-to-date conversion rates for all currencies, so being able to convert between any currency and not just the American dollar is a must. Throughout all of this, I want to teach myself a new web development language called **React**. I see this new language being ingrained as the new standard in web development (I constantly see it as requirements when applying for jobs), so I think it would be both a challenge, and a benefit to learn going forward.

## Deliverables

This project is being created as a learning experience in the React programming language as well as exploring large data manipulation.

This project will provide the following functionality:

- Provide per hour data on popular cryptocurrencies
- Graph data on cryptocurrencies on a per day basis
- Calculate trend lines for graphed data
- Provide data on when it was a best time to buy and sell during a period
- Allow the user to select a length of time to show data
- Allow the conversion between not only cryptocurrencies but all other currency types

This project is intended as a web based application, so a multi-user setting may be commonplace. A database will be used to store data on an hourly basis. Appropriate documentation will be produced, and can be accessed on the site when finished.

# Project Plan Overview

The below contains anticipated and actual completion dates for the major aspects of the project.

Task	Completion Date	Description
	<b>Anticipated</b> <b>Actual</b>	
Information gathering and Research	<b>2/19/18</b> <b>2/22/18</b>	I want to familiarize myself with other applications that may already have the functionality for which I am looking. I also want to use this time to research <b>React</b> and do tutorials as it is a new language for me.
Program Design	<b>2/26/18</b> <b>2/26/18</b>	I want to use this week for designing key aspects of the project, including the database as well as key functionality from deliverables.
Interface Design	<b>3/5/18</b> <b>3/5/18</b>	This week will be dedicated to developing most of the user-interface (which includes assets). This will be an ongoing process throughout the rest of the project.
Implementation	<b>4/2/18</b> <b>4/3/18</b>	Program design specifications will be actualized.
Testing	<b>4/13/18</b> <b>4/18/18</b>	All testing code will be finished by this point. Testing code will be written alongside implementation phase.
Documentation	<b>4/20/18</b> <b>4/24/18</b>	Documentation will be finalized and Capstone materials will be produced.
Final Completion	<b>4/30/18</b> <b>4/24/18</b>	Use these 10 days as gap time, this time will be used only when completion dates run over.

# Use Cases

Below are the Use Case definitions described for the project.

## Use Case #1.0 - User Opens App

ID	1.0.0
Title	User Opens App
Description	User navigates to <a href="http://crypto.jacobsickels.com">http://crypto.jacobsickels.com</a>
Primary Actor	User
Preconditions	User isn't logged in
Postconditions	System presents user with login screen
Main Success Scenario	<ol style="list-style-type: none"><li>1. User navigates to <a href="http://crypto.jacobsickels.com">http://crypto.jacobsickels.com</a></li><li>2. System presents user with the login screen</li></ol>
Frequency of Use	Everytime the user navigates to <a href="http://crypto.jacobsickels.com">http://crypto.jacobsickels.com</a>
Status	Completed

## Use Case #2.0 - User Logs into App (when not authenticated)

ID	2.0.0
Title	User Logs into App (when not authenticated)
Description	User uses their Google login to log into the application from the login screen.
Primary Actor	User
Preconditions	User isn't logged in
Postconditions	User is logged in
Main Success Scenario	<ol style="list-style-type: none"><li>1. User navigates to <a href="http://crypto.jacobsickels.com">http://crypto.jacobsickels.com</a></li><li>2. System finds that user is not authenticated through cookies</li><li>3. System shows login screen to user</li><li>4. User clicks the Google login button</li><li>5. System displays a popup for login</li><li>6. User enters their login data</li></ol>

	<ol style="list-style-type: none"> <li>7. System authenticates user through firebase <ol style="list-style-type: none"> <li>a. System gets auth id and stores it in Redux</li> </ol> </li> <li>8. System logs in user</li> <li>9. System routes user to Dashboard page</li> </ol>
Frequency of Use	Anytime the user would like to use the app (and is not authenticated)
Status	Completed

#### Use Case #2.1 - User Logs into App (when authenticated)

ID	2.1.0
Title	User Logs into App (when already authenticated)
Description	User navigates to <a href="http://crypto.jacobsickels.com">http://crypto.jacobsickels.com</a> to login, but is already authenticated through firebase
Primary Actor	User
Preconditions	User is already authenticated
Postconditions	User is logged in
Main Success Scenario	<ol style="list-style-type: none"> <li>1. User navigates to <a href="http://crypto.jacobsickels.com">http://crypto.jacobsickels.com</a></li> <li>2. System finds that user is authenticated through cookies</li> <li>3. System gets login data from cookies <ol style="list-style-type: none"> <li>a. System gets auth id and stores it in Redux</li> </ol> </li> <li>4. System logs in user</li> <li>5. System routes User to Dashboard page</li> </ol>
Frequency of Use	Anytime the user would like to use the app (and is already authenticated)
Status	Completed

#### Use Case #3.0 - User Navigates to Dashboard Page

ID	3.0.0
Title	User Navigates to Dashboard Page
Description	User navigates to the dashboard page
Primary Actor	User

Preconditions	User is authenticated
Postconditions	User is presented with current day of currency data for Bitcoin, Litecoin, and Ethereum.
Main Success Scenario	<ol style="list-style-type: none"> <li>1. User clicks on a link to the Dashboard Page</li> <li>2. System pushes “/dashboard” to history and url</li> <li>3. System dispatches default start and end currency <ol style="list-style-type: none"> <li>a. (The dashboard should only show the current day of currency data)</li> </ol> </li> <li>4. System selects currency points that fit the start and end currency filters</li> <li>5. System routes User to the Dashboard Page</li> </ol>
Frequency of Use	Anytime the user clicks on a link to the dashboard page, or user enters into the app when they are already authenticated.
Status	Completed

#### Use Case #4.0 - User Navigates to Exchange Page

ID	4.0.0
Title	User Navigates to Exchange Page
Description	User navigates to exchange page where they are presented with options to set an amount to check, a base currency, and the currency to exchange the amount to.
Primary Actor	User
Preconditions	User is authenticated, User is on Exchange Page
Postconditions	User is presented with Exchange Page
Main Success Scenario	<ol style="list-style-type: none"> <li>1. User clicks on a link to the Exchange Page</li> <li>2. System pushed “/exchange” to history and url</li> <li>3. System routes user to the Exchange Page</li> </ol>
Frequency of Use	Anytime the user would like to use the app exchange page (and is already authenticated)
Status	Completed



#### Use Case #4.1 - User Edits Amount Field on Exchange Page

ID	4.0.0
Title	User Edits Amount Field on Exchange Page
Description	User edits the amount field on the exchange page and can change the base amount to check for exchange rates.
Primary Actor	User
Preconditions	User is authenticated, User is on Exchange Page
Postconditions	Amount field is updated in component state.
Main Success Scenario	<ol style="list-style-type: none"><li>1. User enters into the amount field</li><li>2. User edits the amount</li><li>3. System updates Exchange Component State amount</li><li>4. System updates amount in render</li><li>5. System calculates exchange between base and conversion currencies</li><li>6. System displays exchange rate</li></ol>
Frequency of Use	Anytime the user would like to use the app exchange page (and is already authenticated)
Status	Completed

#### Use Case #4.2 - User Edits Base Currency on Exchange Page

ID	4.0.0
Title	User Edits Base Currency on Exchange Page
Description	User edits the base currency on the exchange page by clicking on a dropdown for options.
Primary Actor	User
Preconditions	User is authenticated
Postconditions	Base currency is updated in component state.
Main Success Scenario	<ol style="list-style-type: none"><li>1. User clicks base currency dropdown</li><li>2. User selects a currency option from the dropdown</li><li>3. System updates Exchange Component State for base currency</li></ol>

	<ol style="list-style-type: none"> <li>4. System updates base currency in render</li> <li>5. System calculates exchange between base and conversion currencies</li> <li>6. System displays exchange rate</li> </ol>
Frequency of Use	Anytime the user would like to use the app exchange page (and is already authenticated)
Status	Completed

#### Use Case #4.3 - User Edits Conversion Currency on Exchange Page

ID	4.3.0
Title	User Edits Conversion Currency on Exchange Page
Description	User edits the converted currency on the exchange page by clicking a dropdown for options.
Primary Actor	User
Preconditions	User is authenticated
Postconditions	User is presented with Exchange Page
Main Success Scenario	<ol style="list-style-type: none"> <li>1. User clicks exchange currency dropdown</li> <li>2. User selects a currency option from the dropdown</li> <li>3. System updates Exchange Component State for conversion currency</li> <li>4. System updates conversion currency in render</li> <li>5. System calculates exchange between base and conversion currencies</li> <li>6. System displays exchange rate</li> </ol>
Frequency of Use	Anytime the user would like to use the app exchange (and is already authenticated)
Status	Completed

#### Use Case #5.0 - User Navigates to Filtering Page on Specific Currency

ID	5.0.0
Title	User Navigates to Filtering Page on Specific Currency
Description	User clicks a link taking them to the filtering page for a specific

	currency.
Primary Actor	User
Preconditions	User is authenticated
Postconditions	User is presented with the filtering page for the currency they clicked on.
Main Success Scenario	<ol style="list-style-type: none"> <li>1. User selects a currency to filter</li> <li>2. System pushes 'filter/:currency' to history</li> <li>3. System routes user to 'filter/:currency' page</li> </ol>
Extensions	<ol style="list-style-type: none"> <li>1a. User selects currency to filter from navigation</li> <li>1b. User selects currency to filter from dashboard components</li> </ol>
Frequency of Use	Anytime the user would like to filter a specific currency (and is already authenticated)
Status	Completed

#### Use Case #5.1 - User Edits Date Range on Filtered Currency

ID	5.1.0
Title	User Edits Date Range on Filtered Currency
Description	User selects the date range picker where they can change the date range to filter currencies under.
Primary Actor	User
Preconditions	User is authenticated
Postconditions	Currency data is filtered by date range chosen from date range picker.
Main Success Scenario	<ol style="list-style-type: none"> <li>1. User selects the date range picker to edit the date range</li> <li>2. User changes the start date in the range</li> <li>3. User changes the end date in the range</li> <li>4. System takes this date range and updates filters in Redux</li> <li>5. System selects cryptos using new filters</li> <li>6. System updates graph with new cryptos</li> </ol>
Frequency of Use	Anytime the user would like to use the app exchange (and is already authenticated)
Status	Completed

#### Use Case #5.2 - User Clears Date Range on Filtered Currency

ID	5.2.0
Title	User Clears Date Range on Filtered Currency
Description	User clicks the clear button on date range picker
Primary Actor	User
Preconditions	User is authenticated
Postconditions	Currency filters are defaulted to start and end of current day, and currencies are sorted on this default filter
Main Success Scenario	<ol style="list-style-type: none"><li>1. User clicks the clear button on the date range picker</li><li>2. System updates the start date to the beginning of the current day</li><li>3. System updates the end date to the end of the current day</li><li>4. System updates start and end date filters in Redux</li><li>5. System selects currency data using new filters</li><li>6. System updates graphs with new data</li></ol>
Frequency of Use	Anytime the user would like to use the app exchange (and is already authenticated)
Status	Completed

#### Use Case #6.0 - User Navigates to Account Page

ID	6.0.0
Title	User Navigates to Account Page
Description	User navigates to Account Page where they are presented with an option to change the base currency on their account.
Primary Actor	User
Preconditions	User is authenticated
Postconditions	User is presented with Account Page
Main Success Scenario	<ol style="list-style-type: none"><li>1. User clicks on the Settings option under Account in the navigation bar</li></ol>

	<ol style="list-style-type: none"> <li>2. System pushes '/account' to history</li> <li>3. System routes user to Account Page</li> </ol>
Frequency of Use	Anytime a user would like to check or edit their account settings.
Status	Completed

#### Use Case #6.1 - User Edits Base Currency on Their Account

ID	6.1.0
Title	User Edits Base Currency on Their Account
Description	User edits the base currency on their account that they would like to convert from.
Primary Actor	User
Preconditions	User is authenticated, User is on Account PAge
Postconditions	The base currency on the Users account is changed.
Main Success Scenario	<ol style="list-style-type: none"> <li>1. User selects the base currency dropdown</li> <li>2. User selection an option in the currency dropdown</li> <li>3. System updates component state with User's selection</li> <li>4. System updates filter data on Redux for base currency</li> <li>5. System updates User's base currency on Firebase</li> </ol>
Frequency of Use	Whenever User wants to update the base currency on their account
Status	Completed

#### Use Case #7.0 - User Logs Out of App

ID	7.0.0
Title	User Logs Out of App
Description	User clicks the logout button in the navigation bar and logs out of the app.
Primary Actor	User
Preconditions	User is logged in.

Postconditions	System logs user out, User is presented with login screen
Main Success Scenario	<ol style="list-style-type: none"> <li>1. User clicks the logout button in the navigation</li> <li>2. System logs the user out through Firebase Authentication <ol style="list-style-type: none"> <li>a. System updates Redux authentication data</li> </ol> </li> <li>3. System pushes '/' to history</li> <li>4. System routes user to login screen</li> </ol>
Frequency of Use	Anytime the user would like to log out of the app.
Status	Completed

#### Use Case #8.0 - System Updates Crypto Data Once an Hour

ID	8.0.0
Title	System Updates Crypto Data Once an Hour
Description	Cron job runs once an hour running PHP script that updates Cryptocurrency data on Firebase.
Primary Actor	System
Preconditions	None
Postconditions	System has written new entries for current prices of Bitcoin, Litecoin, and Ethereum to Firebase.
Main Success Scenario	<ol style="list-style-type: none"> <li>1. Cron executes PHP script at beginning of the hour</li> <li>2. System gets cryptocurrency data from Coinbase API</li> <li>3. System updates cryptocurrency data on Firebase</li> </ol>
Frequency of Use	Cron runs once an hour.
Status	Completed

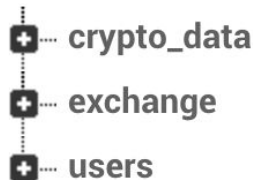
#### Use Case #9.0 - System Updates Exchange Values Once an Hour

ID	9.0.0
Title	System Updates Exchange Values Once an Hour
Description	Cron job runs once an hour, running PHP script that updates exchange rate data on Firebase
Primary Actor	System, Cron

Preconditions	None
Postconditions	System updates exchange rate data in Firebase.
Main Success Scenario	<ol style="list-style-type: none"> <li>1. Cron executes PHP script at beginning of the hour</li> <li>2. System gets exchange rate data from Coinbase API</li> <li>3. System updates exchange rate data on Firebase</li> </ol>
Frequency of Use	Cron runs once an hour
Status	Completed

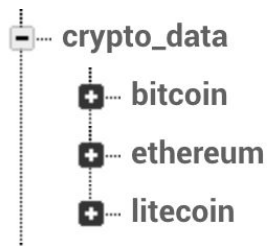
## Table Structure

Firebase is a real time NoSQL cloud database, that automatically syncs data to all connected clients. I decided to use Firebase because of it's easy integration with React and JavaScript. Firebase stores data as JSON objects, so this provided easy integration with JavaScript. My database has been sectioned into 3 distinct buckets.

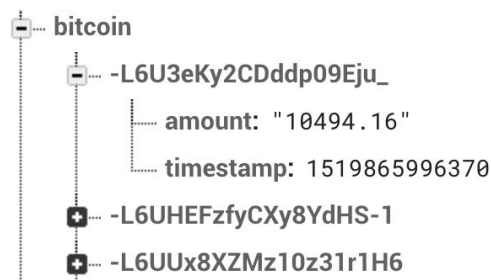


### Crypto\_data

*Crypto\_data* is the bucket designated to storing amounts and timestamps for each specific currency. Under *crypto\_data*, you will find a section for each cryptocurrency I am tracking.



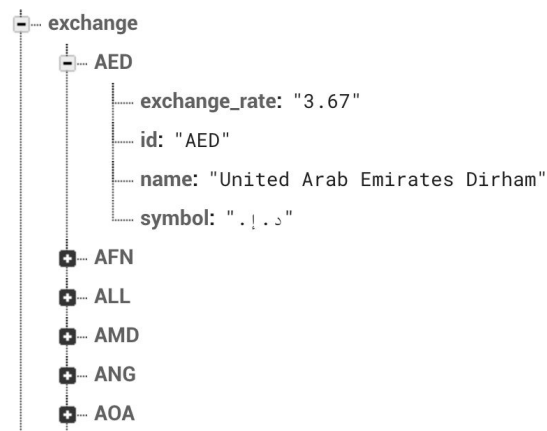
Firebase has a really nice structure, where when you push new data to a section, it creates a new UID (unique identification number) for the pushed data. So, when the cron pushes new data to each crypto once an hour, it gets pushed as a single JSON object, and that sits under a UID that Firebase creates. An example amount and timestamp object pushed from the cron may look like this:





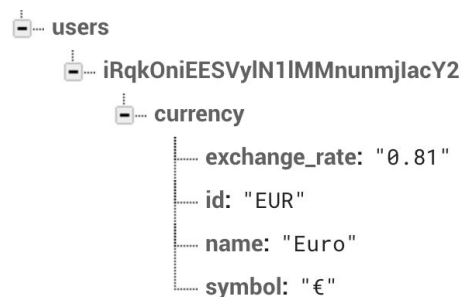
## Exchanges

*Exchanges* is the bucket designated for storing the exchange rate data objects for converting the cryptocurrency data obtained from *crypto\_data*. Users can store an object from *exchanges* under their authentication id stored in the *users* bucket of the database. Each currency under *exchanges* sits under their ID, and contains their exchange rate, id, name, and currency symbol. The *exchange* section of the database is also updated once an hour. An example exchange rate object in the database may look like this:



## Users

*Users* is the bucket designated to storing the base exchange rate a user has defined in their account settings. When a user is signed in to the app, Google creates a UID for them. If a user changes their base currency on the app, a record is written to the database under their UID (an example of this is below). Users who haven't designated a base currency in their account settings are given the base currency of USD (United States Dollar). In addition, because there is a base currency definition for the app, if a user never changes from the base currency, there is no need to write this record to the database. In this example, the user has changed their base currency to Euro.



# Architecture

This section is dedicated to explaining the Cryptotracker application architecture and organization. There are four main pieces to this application that can be first separated into parts and explained individually before they are brought together as a whole. The four main parts of the application are **React**, **Redux**, **Firebase**, and my own defined cron jobs.



## React

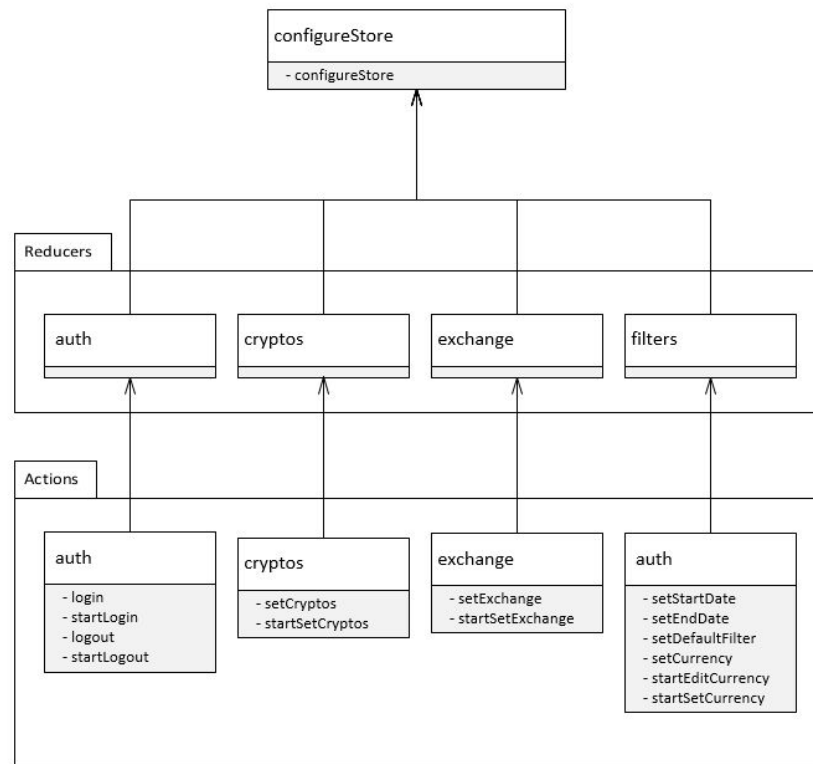
React is a javascript library for building user interfaces. An interface can be broken up into reusable modules called *components* that are built in a class-like structure (I actually used JavaScript ES6 class definitions to do this so they are very similar). React components can have their own state, life-cycle methods, user-defined methods, and *must contain* a render method. When combining React with multiple external libraries, it can become very powerful in regards to building fast and efficient user interfaces. React documentation can be found here <https://reactjs.org/>.



## Redux

Redux is a state container for JavaScript applications. In this way, Redux can be used for dealing with application state separately from React component state. Because application state is abstracted from React, data passed to React components can be more easily monitored and accessed from anywhere in the application. Combining Redux with tools like their live debugger makes testing application state and data manipulation very easy. Redux documentation can be found here <https://redux.js.org/>.

## Redux Diagram



The Redux store consists of 3 main parts: Actions, Reducers, and configureStore.

### Actions

Actions are objects created to update the Redux state. Each action file can contain many actions, in this case they are grouped under the name of the reducer they are going to edit. All action objects are defined with a *type* used by the reducers to decide how the application state should be updated. A generic action takes this form:

```
{
  type: 'EXAMPLE_ACTION',
  updatingData: data
}
```

The *type* key in this object is a string with a definition of the action that is going to be executed. The *updatingData* key holds the object data, which is the data used to update the Redux state. This is the *LOGIN* action for Cryptotracker.

```
{
  type: 'LOGIN',
  uid
}
```

This object takes the same form as above. The LOGIN type defines that action that is going to be performed, and the uid is the data that is going to be used to update the authentication section of Redux state.

## Reducers

Reducers are function that take the action objects and perform the necessary changes to their section of the Redux state. In this example, we are going to see how the LOGIN action object changes the authentication Redux state. Below is the authentication reducer.

```
export default (state = {}, action) => {
  switch(action.type) {
    //The LOGIN action.type updates the state to whatever the action.uid that is passed
    //This authenticates the user
    case 'LOGIN':
      return {
        uid: action.uid
      };
    //The LOGOUT action.type updates the state to empty, unauthenticating the user
    case 'LOGOUT':
      return {};
    //When action.type(s) aren't matched the current state is returned
    default:
      return state;
  }
};
```

This reducer takes the action object and switches on the *type* of the action that is dispatched. As you can see, the case for *LOGIN* takes the action objects uid returns an object that contains that new uid. In this reducer, a returned object is what makes up the authentication state. So, this object that is returned from the *LOGIN* case is the uid that is passed from the action object. In this way, the action object that was dispatched has updated the authentication state.

To reiterate, each reducer edits its own section of Redux state. There are four reducers in this application: auth, cryptos, exchange, and filters. The combination of these reducers into the complete Redux state is explained in the **Configure Store** section.

## Configure Store

The *configureStore* function is what brings together all of the separate reducers into a single application state. Below is the *configureStore* function.

```
export default () => {
  const store = createStore(
    combineReducers({
      auth: authReducer,
      filters: filtersReducer,
      cryptos: cryptosReducer,
      exchange: exchangesReducer
    }),
    composeEnhancers(applyMiddleware(thunk))
  );

  return store;
};
```

This function puts the reducer objects under the same names for each reducer. In this way, a reducer is responsible for a single section of the Redux state. Seen below is a representation of the complete Redux state for the application.

State		
Tree	Chart	Raw
<pre>▶ auth (pin): { uid: "iRqk0niEES_" } ▶ filters (pin): { startDate: "2018-04-22_", endDate: "2018-04-23_", currency: {_-} } ▶ cryptos (pin): { bitcoin: {_-}, litecoin: {_-}, ethereum: {_-} } ▶ exchange (pin): [{_-}, {_-}, {_-}, {_-}, {_-}]</pre>		

As you can see, each reducer is an object under a similar name. These four objects make up the current Redux state, and each reducer can only edit a single specific object.



## Firestore

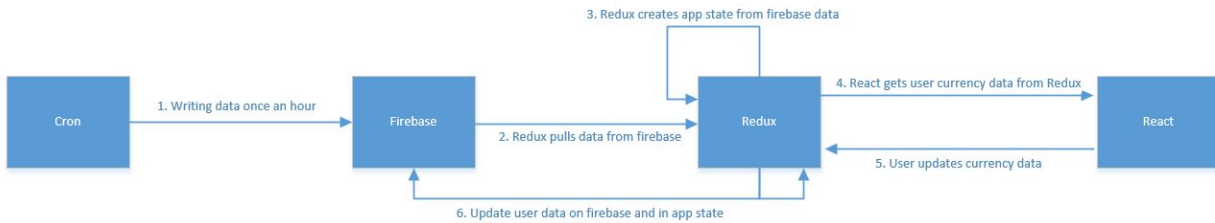
Firestore is a NoSQL cloud database solution. If you would like to learn more about Firestore and table structure for this application, please refer to the Table Structure section of this documentation. If you would like to learn more about Firestore specifically, documentation can be found here <https://firebase.google.com/docs/>.

## Cron

Cron is another very important aspect to this application. Cron and cron-jobs are scheduled tasks that run on very specific intervals. For this application, both cryptocurrency data and exchange rate data need to be obtained and written to the Firestore database once an hour. To do this, PHP scripts were written and stored on the application's host server through Godaddy. These PHP scripts access the Coinbase API (<https://developers.coinbase.com/>) to gather data on specific cryptocurrencies and exchange rates. Godaddy provides a cron scheduler through their cPanel hosting solution. So, all that needs to be done is run the PHP scripts using this scheduler once an hour. Even though this process is completely separate from the main application, it is important to understand where the data is coming from, and the processed that obtain that data. If you would like to learn more about cron you can check it out here <https://en.wikipedia.org/wiki/Cron>.

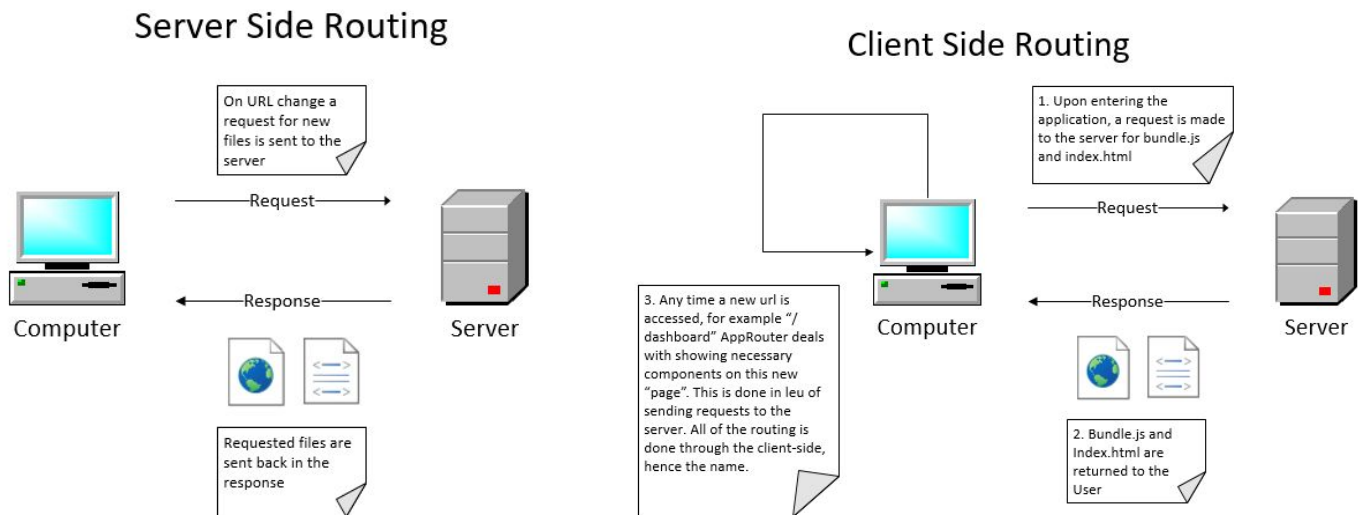
*Next is a high level overview of data flow in Cryptotracker.*

## Data Flow Diagram



Data flow through the program mostly relies on the Redux store. Firstly, newly obtained data is retrieved from the cron jobs and written to Firebase. When the application is ran, Redux picks up all the data from Firebase and puts it into the application state. React components (the user interface) grabs data from the Redux store and manipulates it as needed. The only data being written to firebase from the user interface is the base currency selection from the user. So, if a user chooses to update their base currency, that is written to the Redux state as well as to Firebase. Using this architecture, data is concurrent between Firebase and Redux at all times.

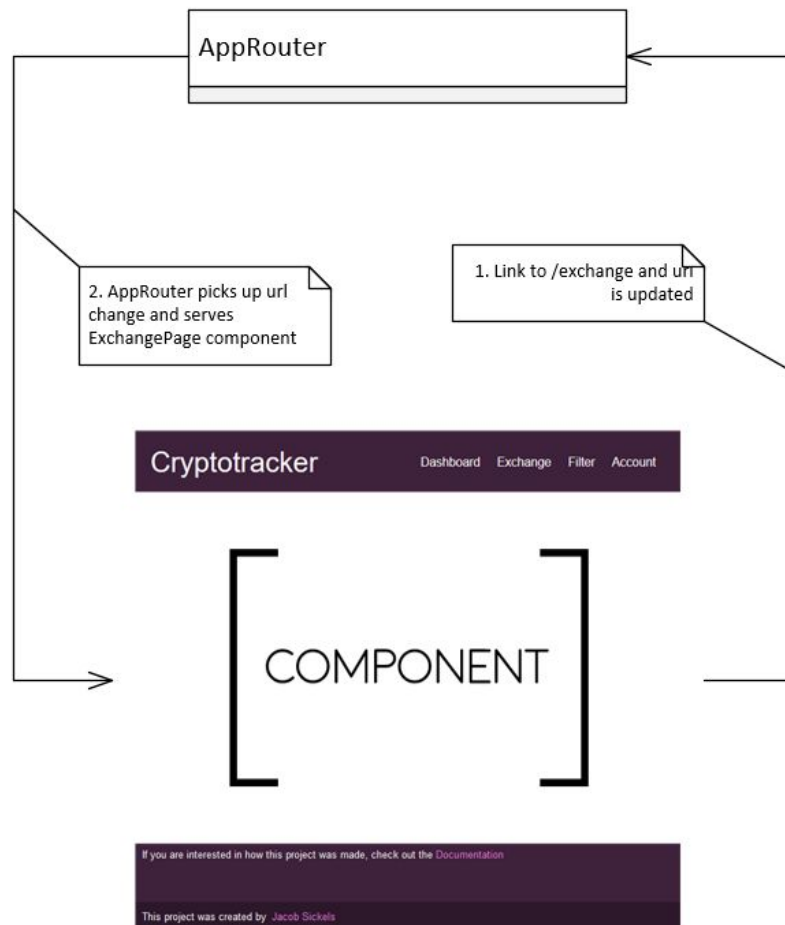
# Client-Side Routing



Client-side routing is a main function of React. The difference between server-side routing and client-side routing is how pages are served to the user. In server-side routing, When a user clicks on a link, a request is made to the server, and the server serves them the new page. In client-side routing, only an initial request is made to the server for the original application files. Webpack bundles the JavaScript files together into a single *bundle.js*, as well as bundles all of the CSS files into a single *styles.css* file. These two files along with the original *index.html* file that the application is run from are requested from the server and served to the User. After this is complete, no more requests are made to the server for this application. When a User clicks on a link, the url is changed to reflect this link. The AppRouter picks up the URL change, and serves the component assigned to that URL. If you would like to see how the AppRouter does this, please look at the code for it under the **Code Highlights** section of this documentation.

*Below is a representation of how AppRouter serves components.*





AppRouter fills the component area of the application by matching the url to a component. When a link is clicked on the application, the url is updated to reflect a page change. When this happens, the AppRouter picks that up, and serves the component that matches the url. In this example, the Exchange link is clicked in the navigation. The url is updated to <http://crypto.jacobsickels.com/exchange> then the AppRouter serves the ExchangePage component into the component area. This image is the essence of client-side routing.

# Testing Plan

In lue of a testing plan, during the research and development section of this project I learned about a testing framework called Jest. This framework allows for creating test cases, mocking function calls, simulating inputs and button clicks, and even checking rendering of functions through a library called Enzyme. So, the testing plan for this project is the creation of a testing suite for the whole of the application. In addition, the testing suite can be written *and run* alongside changes made to the base code so unwanted changes can be caught by the testing suite running in the background. Explanation of how the testing suite solves each use case can be found in the Testing Suite section below.

## Testing Suite

For this application I created a testing suite that accomplishes many of the use cases defined above. If a testing case wasn't explicitly written to accompany a use case, an explanation of testing is given. Two libraries, Jest and Enzyme were used to accomplish this task, and they are explained in more detail separately below. If you would like to learn more about Jest and Enzyme please check them out here <https://facebook.github.io/jest/> and <http://airbnb.io/enzyme/docs/api/>.

### Jest

Jest is a testing library developed by Facebook for testing JavaScript applications (especially React applications). Here is an example of a test file used to test the Login Page.

```
import React from 'react';
import { shallow } from 'enzyme';

import { LoginPage } from '../../components/LoginPage';

test('should render LoginPage correctly', () => {
  const wrapper = shallow(<LoginPage startLogin={() => {}} />);
  expect(wrapper).toMatchSnapshot();
});

test('should call startLogin on button click', () => {
  const startLogin = jest.fn();
  const wrapper = shallow(<LoginPage startLogin={startLogin} />);
  wrapper.find('button').simulate('click');
  expect(startLogin).toHaveBeenCalled();
});
```

From top to bottom, the first three lines are file and module imports (this convention is JavaScript ES6). There is an import for React (so that we can render react components), a method *shallow* from enzyme (used for doing shallow renders of components), and lastly an import of the Login Page component that is being tested.

The first test case in this file is a rendering test case.

```
test('should render LoginPage correctly', () => {  
  const wrapper = shallow(<LoginPage startLogin={() => {}} />);  
  expect(wrapper).toMatchSnapshot();  
});
```

The first line of the use case is a header that allows the test case to be named. In this case, the test is called “should render LoginPage correctly”. If this test case fails, the name is clearly seen in the console output for the failed test. Using Enzyme’s shallow rendering function (Enzyme is explained a bit more below), we can take the rendered output and make a snapshot file. The very first time this test case runs, it passes because there isn’t a snapshot file created to match it to. Snapshot files look like this:

```
exports[`should render LoginPage correctly 1`] = `  
  <div className="login">  
    <div className="login__box">  
      <div className="login__image">  
          
      </div>  
      <h1 className="login__title">  
        CryptoTracker  
      </h1>  
      <p className="login__subtitle">  
        Tracking cryptos, one hour at a time.  
      </p>  
      <button  
        className="button"  
        onClick={[[Function]]}  
      >  
        Login with Google  
      </button>  
    </div>  
  </div>  
`;
```

Snapshot files are used to compare shallow rendered html in the test case. The first time this test case is run, there isn’t a snapshot to compare to. The test case doesn’t fail, but the snapshot is created and all subsequent test cases are compared against this snapshot. If any part of the shallow rendering is different on subsequent executions, the test case fails.

The second test case in this file is making sure that an action is happening when the login button is clicked. Here is what this test case looks like:

```
test('Use Case #2.0: Should call startLogin on button click', () => {
  const startLogin = jest.fn();
  const wrapper = shallow(<LoginPage startLogin={startLogin} />);
  wrapper.find('button').simulate('click');
  expect(startLogin).toHaveBeenCalled();
});
```

Jest.fn() is a mock function. Setting startLogin to this mock function and passing it inside of the shallow render of the component allows us to check if the function has been called. This mock function overrides the real function in the component when it is passed as a prop, so the real function is never called. The test case simulates the button click inside of the actual component. Here is a look at the component with some important sections highlighted (and explained below).

```
import React from 'react';
import { connect } from 'react-redux';
import { startLogin } from '../actions/auth';

export const LoginPage = ({ startLogin }) => (
  <div className="login">
    <div className="login__box">
      <div className="login__image">
        
      </div>
      <h1 className="login__title">CryptoTracker</h1>
      <p className="login__subtitle">Tracking cryptos, one hour at a time.</p>
      <button className="button" onClick={startLogin}>Login with Google</button>
    </div>
  </div>
);

const mapDispatchToProps = (dispatch) => ({
  startLogin: () => dispatch(startLogin())
});

export default connect(undefined, mapDispatchToProps)(LoginPage);
```

In React, this is what's known as a connected component. All this means is that it is connected to Redux (and the application state) which allows it to **dispatch** actions to Redux, changing the application state. If you look at the bottom, this dispatch command is called whenever the startLogin method is called. When this component is live and not being tested, the startLogin function is passed as a **property** into the component, which can be seen from the first highlighted section. This property is passed in from *mapDispatchToProps*, which maps the startLogin function to a dispatch to Redux. When we want to override this property as in our test case, we can pass a mock function as a property so that the real function is never called. Lastly, the middle highlighted section is the button that we are simulating a click on. There is an onClick method attached to this button that calls the startLogin function passed as a property.

```

test('should call startLogin on button click', () => {
  const startLogin = jest.fn();
  const wrapper = shallow(<LoginPage startLogin={startLogin} />);
  wrapper.find('button').simulate('click');
  expect(startLogin).toHaveBeenCalled();
});

```

The last line in our test case is an expect function that checks if our mock function was called when we simulated the button click. If the mock function was never called, this test case fails.

## Enzyme

Enzyme is a testing utilities library created by Airbnb for React. From their Github page,

*Enzyme is a JavaScript Testing utility for React that makes it easier to assert, manipulate, and traverse your React Components' output.*

This application mostly uses Enzyme for shallow rendering components, and checking both rendering and mock functions.

## Use Case Actualization

Using Jest and Enzyme, testing cases can be created to test specific use cases. In the example above, **Use Case #2.0** is satisfied by the testing cases created for the Login Page. Below are the test cases created to satisfy each specific use case. If a use case doesn't require a testing case, a testing plan for that use case is also explained below.

### Testing Scenario #1

*For Use Case #1*

This use case is for testing the React Router implementation of this application. When a user enters at the root (<http://crypto.jacobsickels.com/>) they should be presented with the login page if they are not authenticated. To test this, I just made sure that I was logged out and loaded up the application. This was tested multiple times alongside the **Use Case #2.0** which deals with the login process.

### Testing Scenario #2

*For Use Case #2.0*

This use case is gone into extensively above during the explanation of the testing suite, but it is reiterated on below.

```
test('Use Case #2.0: Should call startLogin on button click', () => {
  const startLogin = jest.fn();
  const wrapper = shallow(<LoginPage startLogin={startLogin} />);
  wrapper.find('button').simulate('click');
  expect(startLogin).toHaveBeenCalled();
});
```

This test case tests for the login function being called when the login button is clicked. The startLogin function dispatches the startLogin action to Redux. This action was tested through this test case.

```
test('Should setup login action object', () => {
  const uid = '1234';
  const action = login(uid);
  expect(action).toEqual({
    type: 'LOGIN',
    uid
  });
});
```

This test case dispatches the uid of the user to Redux for login authentication. This satisfies all of the User input requirements and System requirements that are brought up for Use Case #2.0.

### Testing Scenario #3

*For Use Case #2.1*

This use case was made for testing the React Router when the user enters the app from the root and is already authenticated through cookies. Many test cases that were written for routing the user to another page where completed through the implementation of *AppRouter* and each scenario was tested to completion through the use of a development server.

### Testing Scenario #4

*For Use Case #3*

This use case was made for testing the React Router when the user navigates to the Dashboard page from a link or when they enter the application when they are already authenticated. This code was implemented though the *AppRouter* and tested to completion through a development server.

### Testing Scenario #5

*For Use Case #4.0*

This use case was made for testing the React Router when the user navigates to the Exchange page from a link on the navigation bar. This code was implemented though the *AppRouter* and tested to completion through a development server.

### Testing Scenario #6

*For Use Case #4.1*

```

test('Use Case #4.1: Should handle amount change', () => {
  const value = "12345.64";
  wrapper.find('#amount-input').at(0).simulate('change', {
    target: { value }
  });
  expect(wrapper.state('amount')).toEqual(value);
});

```

This test case was written to make sure that changing the amount on the Exchange Page updates the component state. The amount input field is found on the component through the id *amount-input*, and a simulated *change* is acted on the input. The changed value amount is passed in, and it is expected that the state is changed to the new value. This satisfies the User input requirements and the System requirements in Use Case #4.1.

## Testing Scenario #7

For Use Case #4.2

```

test('Use Case #4.2: Should handle base currency change', () => {
  const value = toExchange.id;
  wrapper.find('#base-currency').at(0).simulate('change', {
    target: { value }
  });
  expect(wrapper.state('from_element')).toEqual(toExchange);
});

```

This test case was written to make sure that changing the base currency dropdown on the Exchange Page updates the component state. The base currency dropdown is found in the component through the id *base-currency*, and a simulated *change* is acted on the dropdown. A dummy id is passed to the dropdown change method, and it is expected that the component state is updated to equal the passed in exchange object that contains the same id. This satisfies the User input requirements and System requirements for Use Case #4.2.

## Testing Scenario #8

For Use Case #4.3

```

test('Use Case #4.3: Should handle conversion currency change', () => {
  const value = toExchange.id;
  wrapper.find('#conversion-currency').at(0).simulate('change', {
    target: { value }
  });
  expect(wrapper.state('to_element')).toEqual(toExchange);
});

```

This test case was written to make sure that changing the conversion currency on the Exchange Page updates the component state. The conversion currency dropdown is found in the component through the id *conversion-currency*, and a simulated *change* is acted on the dropdown. A dummy id is passed to the dropdown change method, and it is expected that the component state is updated to equal the passed in exchange object that contains the same id. This satisfies the User input requirements and System requirements in Use Case #4.3.



## Testing Scenario #9

For Use Case #5.0

This use case was made for testing the React Router when the user navigates to the filtering page for a specific currency, either through clicking on the currencies name from the Dashboard Page, or through the filter dropdown in the navigation bar. This code was implemented through the React Router implementation and tested to completion through a development server.

*Problems:* Implementation of this code was a bit harder because React Router has a pattern matching url that can be used to reroute the user.

```
<PrivateRoute
  path="/filter/:currency"
  component={SingleCrypto}
/>
```

In this code, the “:currency” is the pattern matched part of the path. When this path is matched in the url, the component *SingleCrypto* is shown on the page. The *SingleCrypto* component needs to parse out the name of the crypto it needs to show from the url, through this code below.

```
const mapStateToProps = (state, props) => {
  return {
    name: props.match.params.currency.charAt(0).toUpperCase() + props.match.params.currency.slice(1)
  };
};
```

Once the currency name is found from the parameters it is passed to other Components that require the name to parse out the specific crypto data for that page. This section was especially hard to figure out because of the pattern matching in the url.

## Testing Scenario #10

For Use Case #5.1

```
test('Use Case #5.1: Should handle changing dates on filters', () => {
  const startDate = moment().startOf('day');
  const endDate = moment().endOf('day');
  wrapper.find('DateRangePicker').prop('onDatesChange')({startDate, endDate});
  expect(setStartDate).toHaveBeenCalledLastCalledWith(startDate);
  expect(setEndDate).toHaveBeenCalledLastCalledWith(endDate);
});
```

This testing case was written to handle changing the dates on the Filter Page for a specific component. The use case passes in a start and end date, mimicking an entry from the user. It then expects the `setStartDate` and `setEndDate` methods to have been called with the `startDate` object and `endDate` objects respectively. These methods deal with updating the Redux application state so this test case makes sure that these methods are passed the correct data.



```

test('should setup start date filter object', () => {
  const action = setStartDate(moment(0));
  expect(action).toEqual({
    type: 'SET_START_DATE',
    startDate: moment(0)
  });
});

test('should setup end date filter object', () => {
  const action = setEndDate(moment(0));
  expect(action).toEqual({
    type: 'SET_END_DATE',
    endDate: moment(0)
  });
});

```

These two test cases deal with setting up the dispatch action objects for startDate and endDate in Redux. The setStartDate and setEndDate methods call dispatch actions with these objects. With these two test cases, I can expect that the Redux store has been updated appropriately. These test cases deal with User input requirements and System requirements in Use Case #5.1.

## Testing Scenario #11

*For Use Case #5.2*

```

test('Use Case #5.2: Should handle setting default dates when cleared', () => {
  const now = moment();
  const nullCheck = null;
  wrapper.find('DateRangePicker').prop('onDatesChange')({nullCheck, nullCheck});
  expect(setStartDate).toHaveBeenLastCalledWith(now.startOf('day'));
  expect(setEndDate).toHaveBeenLastCalledWith(now.endOf('day'));
});

```

This testing case was written to handle clearing the dates on the Filter Page for a specific component. The clear button on the DateRangePicker should set the filter to the default dates, which happens to be the current day. When the clear button is clicked the DateRangePicker passes null as both values to the onDatesChange method defined in the component. It is in the onDatesChange method where the setStartDate and setEndDate methods are called to update the Redux state filters. So, this test case sends in null as both values to the onDatesChange method and makes sure that setStartDate and setEndDate are both called with the current day as parameters.

## Testing Scenario #12

*For Use Case #6.0*

This use case was made for testing the React Router when the user navigates to the Account Page from a link on the navigation bar. This code was implemented though the *AppRouter* and tested to completion through a development server.

## Testing Scenario #13

*For Use Case #6.1*

```

test('Use Case #6.1: Should handle changeBaseCurrency', () => {

  const currency = {
    "exchange_rate": "106.31",
    "id": "ALL",
    "name": "Albanian Lek",
    "symbol": "Lek"
  };

  const wrapper = shallow(
    <AccountPage store={store} currency={currency} startEditCurrency={mockStartEditCurrency} exchanges={exchanges}/>
  );

  wrapper.find('Input').simulate('change', {
    target: { value: currency.id }
  });
  expect(wrapper.state('currency')).toEqual(currency);
});

```

This testing case was created to test the user changing the base currency on their account. The dropdown is found on the Account Page, and a currency id is passed to the dropdown with a “change” simulation.

```

test('should setup currency filter object', () => {
  const currency = {
    "exchange_rate": "3.67",
    "id": "AED",
    "name": "United Arab Emirates Dirham",
    "symbol": "ﷵ"
  };

  const action = setCurrency(currency);
  expect(action).toEqual({
    type: 'SET_CURRENCY',
    currency
  });
});

test('should get currency object from user section of database', (done) => {
  const currency = {
    "exchange_rate": "3.67",
    "id": "AED",
    "name": "United Arab Emirates Dirham",
    "symbol": "ﷵ"
  };

  const store = createMockStore(defaultAuthState);
  store.dispatch(startSetCurrency()).then(() => {
    const actions = store.getActions();
    expect(actions[0]).toEqual({
      type: 'SET_CURRENCY',
      currency
    });
    done();
  });
});

```

```

test('should set currency object on user in firebase', (done) => {
  const currency = {
    "exchange_rate": "106.31",
    "id": "ALL",
    "name": "Albanian Lek",
    "symbol": "Lek"
  };

  const store = createMockStore(defaultAuthState);
  store.dispatch(startEditCurrency(currency)).then(() => {
    const actions = store.getActions();
    expect(actions[0]).toEqual({
      type: 'SET_CURRENCY',
      currency
    });
    done();
  });
});

```

These test cases deal with creation the dispatch currency object as well as testing the getting and setting of the currency object in Firebase. This testing case deals with all User input requirements and System requirements defined in Use Case #6.1.

### Testing Scenario #14

*For Use Case #7.0*

```

test('Use Case #7.0: Should call startLogout on button click', () => {
  const startLogout = jest.fn();
  const wrapper = shallow(<Header startLogout={startLogout} />);
  wrapper.find('#logout').simulate('click');
  expect(startLogout).toHaveBeenCalled();
});

```

This testing case was created to test the user logging out of the application. This code shallow renders the navigation bar, finds the logout button, and simulates a click. The expectation for this testing scenario is that the startLogout function is called when the button is clicked.

```

test('Should setup logout action object', () => {
  const action = logout();
  expect(action).toEqual({
    type: 'LOGOUT'
  });
});

```

The *startLogout* function dispatches the logout action to Redux, and this action deals with testing the creation of the logout action object that is dispatched. These testing cases deal with all User input requirements defined in Use Case #7.0.

### Testing Scenario #15

*For Use Case #8.0*

This use case was harder to test. At the beginning of the project I was using Google Cloud SDK for running my cron jobs and it wasn't reliable. I switched to using my website host's cron scheduler combined with PHP scripts to achieve this functionality. Everytime the cron job

was run, an email was sent saying whether or not the cron job was completed. This helped with testing because the emails would let me know almost immediately what the issue was in the script so that I could rectify it.

## Testing Scenario #16

For Use Case #9.0

This use case was the same as Use Case #8.0. My website host's cron scheduler was used combined with a separate PHP script to update exchange rate data. Again, whenever the script was run, an email was sent explaining whether or not the cron job was completed.

## Additions

In addition to testing specific use cases, each component also has a testing case for rendering. If you would like to see an example of a test case for rendering, please refer to the Jest section in the Testing Suite explanation where you can find a test rendering function for the Login Page.

## Conclusion

In conclusion, there were 21 test suites written (*test suite being a single file consisting of many test cases*), 45 test cases written, and 13 snapshots created to test this project. There were parts of the project that could benefit from more situations from test cases, but I these test cases turned out to be a very well rounded way to test the functionality of this project. A sample execution of the test cases running is seen below on the left. If a test case fails, what would be seen is something like the right image.

```
D:\Cryptotracker>yarn run test
yarn run v1.3.2
$ cross-env NODE_ENV=test jest --config=jest.config.json
PASS src\tests\components\NotFoundPage.test.js
PASS src\tests\components\CryptoFilter.test.js
PASS src\tests\components\CryptoList.test.js
PASS src\tests\components\CryptoItem.test.js
PASS src\tests\reducers\exchange.test.js
PASS src\tests\components\ExchangePage.test.js
PASS src\tests\components\DashboardPage.test.js
PASS src\tests\actions\filters.test.js
PASS src\tests\components\CryptoInfo.test.js
PASS src\tests\actions\cryptos.test.js
PASS src\tests\components\Header.test.js
PASS src\tests\components\LoginPage.test.js
PASS src\tests\components\SingleCrypto.test.js
PASS src\tests\actions\auth.test.js
PASS src\tests\components\AccountPage.test.js
PASS src\tests\components>LoadingPage.test.js
PASS src\tests\actions\exchange.test.js
PASS src\tests\reducers\crypto.test.js
PASS src\tests\components\Footer.test.js
PASS src\tests\reducers\filters.test.js
PASS src\tests\reducers\auth.test.js

Test Suites: 21 passed, 21 total
Tests: 45 passed, 45 total
Snapshots: 13 passed, 13 total
Time: 4.929s
Ran all test suites.
Done in 6.07s.

D:\Cryptotracker>
```

```
PASS src\tests\reducers\crypto.test.js
PASS src\tests\reducers\auth.test.js

Summary of all failing tests
FAIL src\tests\reducers\exchange.test.js
  ● Should set exchange values

    expect(received).toEqual(expected)

    Expected value to equal:
    [{"exchange_rate": "3.67", "id": "AED", "name": "United Arab Emirates Dirham", "symbol": ".B"}, {"exchange_rate": "69.50", "id": "AFN", "name": "Afghan Afghani", "symbol": "ؑ"}, {"exchange_rate": "106.31", "id": "ALL", "name": "Albanian Lek", "symbol": "Lek"}, {"exchange_rate": "412", "id": "AMD", "name": "Armenian Dram", "symbol": "֏"}, {"exchange_rate": "1.79", "id": "ANG", "name": "Netherlands Antillean Guilder", "symbol": "ƒ"}, {"exchange_rate": "214.12", "id": "AOA", "name": "Angolan Kwanza", "symbol": "Kz"}, {"exchange_rate": "20.14", "id": "ARS", "name": "Argentine Peso", "symbol": "$"}]

    Received:
    undefined

    Difference:

    Comparing two different types of values. Expected array but received undefined.

    at Object.<anonymous> (src\tests\reducers\exchange.test.js:17:29)
    at new Promise (<anonymous>)
    at Promise.resolve.then.e1 (node_modules/p-map/index.js:46:16)
    at <anonymous>
    at process._tickCallback (internal/process/next_tick.js:188:7)

Test Suites: 1 failed, 20 passed, 21 total
Tests: 1 failed, 44 passed, 45 total
Snapshots: 13 passed, 13 total
Time: 5.049s
Ran all test suites.
error Command failed with exit code 1.
info Visit https://yarnpkg.com/en/docs/cli/run for documentation about this command.
```

# Dependencies

This project was built using a application bundler called Webpack and a package manager called Yarn. This project also includes a lot of smaller dependencies, and dependencies on dependencies. Web application development allows for the use of external libraries to supplement development, and Cryptotracker is no different. In this section, I would like to highlight some of the tools and libraries used in Cryptotracker. Not all libraries are highlighted in this section, but if you would like to get a closer look at all of the package dependencies in Cryptotracker please reference the end of this section.

## Webpack

From the Webpack website:

*Webpack is a static module bundler for modern JavaScript applications. When webpack processes your application, it recursively builds a dependency graph that includes every module your application needs, then packages all of those modules into one or more bundles.*

(<https://webpack.js.org>)

## Yarn

From the Yarn website:

Yarn is a package manager for your code. It allows you to use and share code with other developers from around the world. Yarn does this quickly, securely, and reliably so you don't ever have to worry.

(<https://yarnpkg.com>)

Yarn is really good as a dependency manager because modules that are downloaded to supplement the project are also brought in with version numbers. So, if someone wanted to download Cryptotracker locally all they need to do to do is call *yarn install* in the same directory as the root of the project and yarn will automatically install all of the same dependencies *with the same version numbers* that the current project has. This keeps project development consistent while using many dependencies.

## Moment and Moment Timezone

Moment and Moment Timezone are external libraries made for parsing, validating, manipulating and displaying dates in JavaScript. Cryptotracker uses all parts of these libraries, but most importantly, Moment automatically gets the users timezone. This is important because the crypto data points need to be sorted by timezone. A user in Eastern Time will get one less datapoint than a user in Central time on the Dashboard Page because it's showing the current day of data. Moment is very easy to use and can be found here: <https://momentjs.com>.

## React-Materialize

React Materialize is a React Component wrapper for a front-end styling framework called MaterializeCSS (which can be found here: <http://materializecss.com/>). React Materialize takes the styling classes from MaterializeCSS and wraps them in React components which allows for easier use with defined React components

## Other Libraries

The other libraries listed below are also used in the project, but many are used for smaller parts of the project that don't require an explanation for each library. The file included below is the package.json file for the project. This file is how Yarn manages packages and dependencies for the project. If you would like to learn more about Yarn's package.json files you can read this link: <https://yarnpkg.com/lang/en/docs/package-json/>.

```
"dependencies": {
  "babel-cli": "6.24.1",
  "babel-core": "6.25.0",
  "babel-loader": "7.1.1",
  "babel-plugin-transform-class-properties": "6.24.1",
  "babel-plugin-transform-object-rest-spread": "6.23.0",
  "babel-polyfill": "6.26.0",
  "babel-preset-env": "1.5.2",
  "babel-preset-react": "6.24.1",
  "css-loader": "0.28.4",
  "express": "4.15.4",
  "extract-text-webpack-plugin": "3.0.0",
  "firebase": "4.2.0",
  "history": "4.7.2",
  "materialize-css": "^0.100.2",
  "moment": "^2.21.0",
  "moment-timezone": "^0.5.14",
  "node-sass": "4.5.3",
  "normalize.css": "7.0.0",
  "numeral": "2.0.6",
  "raf": "3.3.2",
  "react": "16.0.0",
  "react-addons-shallow-compare": "15.6.0",
  "react-dates": "12.7.0",
  "react-dom": "16.0.0",
  "react-materialize": "^2.1.0",
  "react-modal": "^3.3.1",
  "react-redux": "5.0.5",
  "react-router-dom": "4.2.2",
  "recharts": "^1.0.0-beta.10",
  "redux": "3.7.2",
  "redux-mock-store": "1.2.3",
  "redux-thunk": "2.2.0",
  "sass-loader": "6.0.6",
  "style-loader": "0.18.2",
  "uuid": "3.1.0",
  "validator": "8.0.0",
  "webpack": "^3.10.0"
},
"devDependencies": {
  "cross-env": "5.0.5",
  "dotenv": "4.0.0",
  "enzyme": "3.0.0",
  "enzyme-adapter-react-16": "1.0.0",
  "enzyme-to-json": "3.0.1",
  "jest": "20.0.4",
  "react-test-renderer": "16.0.0",
  "webpack-dev-server": "2.5.1"
}
```



# Resources

The following are useful application resources and libraries used in the development of this application.

Webpack: <https://webpack.js.org/>

Yarn: <https://yarnpkg.com/en/>

NodeJS: <https://nodejs.org/en/>

React: <https://reactjs.org/>

Redux: <https://redux.js.org/>

Firebase: <https://firebase.google.com/>

Jest: <https://facebook.github.io/jest/>

Enzyme: <http://airbnb.io/enzyme/docs/api/>

Cron: <https://en.wikipedia.org/wiki/Cron>

Sass: <https://sass-lang.com/>

Babel: <https://babeljs.io/>

Tutorial Course: <https://www.udemy.com/react-2nd-edition/>

StackOverflow: <https://stackoverflow.com/>

# Source Control

From the beginning of the project, all files have been source controlled. If you would like to see the source code for this project, you can find it on Github at:

<https://github.com/JacobSickels/Cryptotracker>

# Code Highlights

*Some coding comments have been removed for readability. Files that were considered redundant or not important enough for a highlight have been excluded.*

## App.js

App.js is the main entry point for the application. The Redux store and AppRouter are created from this entry point. This class also deals with dispatching all of the necessary actions to the Redux store on application startup.

```
//Creating Redux Store
const store = configureStore();
const jsx = (
  <Provider store={store}>
    <AppRouter />
  </Provider>
);
//Renders AppRouter which displays the correct page
const renderApp = () => {
  ReactDOM.render(jsx, document.getElementById('app'));
};
//Rendering Loading Page while app is loading
ReactDOM.render(<LoadingPage />, document.getElementById('app'));
firebase.auth().onAuthStateChanged((user) => {
  ReactDOM.render(<LoadingPage />, document.getElementById('app'));
  if(user) {
    //Dispatching user id to Redux store
    store.dispatch(login(user.uid));
    //Gets crypto data for graphing
    store.dispatch(startSetCryptos()).then(() => {
    //Gets default currency from account
    store.dispatch(startSetCurrency()).then(() => {
      //Gets exchange rates and puts them in redux state
      store.dispatch(startSetExchange());
      //If user is on the login page and authenticated, push to dashboard
      if(history.location.pathname === '/') {
        history.push('/dashboard');
      }
      renderApp();
    });
  });
}
else {
  //When auth state is changed and user doesn't exist, the user has logged out
  store.dispatch(logout());
  history.push('/');
  renderApp();
}
});
```



## Actions

Below are the Redux action files. After each image is a small explanation of the actions and/or the purpose of the file. Actions are explained in more depth during the **Redux Diagram** section of this documentation. Please refer to the **Source Control** section of the documentation if you would like to see all of the files for this project.

### Auth

```
//Creating a login action object for dispatching to Redux store
export const login = (uid) => ({
  type: 'LOGIN',
  uid
});

//Creates the Google Popup for signin to authenticate the user
export const startLogin = () => {
  return () => {
    return firebase.auth().signInWithPopup(googleAuthProvider);
  };
};

//Creating the logout action object for dispatching to Redux store
export const logout = () => ({
  type: 'LOGOUT',
});

//Calls Firebase signout, un-authenticating the user
export const startLogout = () => {
  return () => {
    return firebase.auth().signOut();
  }
};
```

This is the authentication action objects file, which contains four actions. The first, *login*, creates the login action object for editing the Redux state. The second, *startLogin*, is called when the user clicks the *Login with Google* button, this function is what creates the signin popup. The third, *logout*, creates the logout action object for editing the Redux store. Lastly, *startLogout*, is the function that is called when the user clicks the *Logout* button, this function authenticates the user through Firebase.

## Cryptos

The *cryptos* action file contains two actions. The first action in this file, *setCryptos*, creates the *setCryptos* action object for editing the Redux state. The second action, *startSetCryptos*, gets the cryptocurrency data from Firebase and sets up the cryptos object to be dispatched inside of the *setCryptos* to edit the Redux state.

```
//Creating a setCryptos action object for dispatching to Redux store
export const setCryptos = (cryptos) => ({
  type: 'SET_CRYPTOS',
  cryptos
});

export const startSetCryptos = () => {
  return (dispatch) => {
    //Getting data from Firebase
    return database.ref('crypto_data').once('value')
      .then((snapshot) => {
        // Creating an object to load Firebase data into
        const cryptos = {
          bitcoin: [],
          litecoin: [],
          ethereum: []
        };
        // Looping through Firebase data and loading the object
        snapshot.forEach((childSnapshot) => {

          if(childSnapshot.key === 'bitcoin') {
            //For each uuid key under 'bitcoin' get the value and push it to
            cryptos.bitcoin array
            Object.keys(childSnapshot.val()).forEach(function(key) {
              cryptos.bitcoin.push(childSnapshot.val()[key]);
            });
          }
          else if(childSnapshot.key === 'litecoin') {
            //For each uuid key under 'litecoin' get the value and push it to
            cryptos.litecoin array
            Object.keys(childSnapshot.val()).forEach(function(key) {
              cryptos.litecoin.push(childSnapshot.val()[key]);
            });
          }
          else {
            //For each uuid key under 'ethereum' get the value and push it to
            cryptos.ethereum array
            Object.keys(childSnapshot.val()).forEach(function(key) {
              cryptos.ethereum.push(childSnapshot.val()[key]);
            });
          }
        });

        // Dispatching object to the Redux store.
        dispatch(setCryptos(cryptos));
      });
    };
}
```

## Exchange

The *exchange* action file contains two actions. The first action in this file, *setExchange*, creates the *setExchange* action object for editing the Redux state. The second action in this file, *startSetExchange*, retrieves the exchange data from Firebase and sets up an *exchanges* array to be dispatched with *setExchange* to edit the Redux state.

```
//Creating a setExchange action object for dispatching to Redux store
export const setExchange = (exchanges) => ({
  type: 'SET_EXCHANGE',
  exchanges
});

/*
  startSetExchange()
  This is a dispatching wrapper for the setExchange action object
  This method goes to firebase, grabs the necessary exchange data, and loads an
  object to be passed and dispatched to the Redux store.
*/
export const startSetExchange = () => {
  return (dispatch) => {
    //Getting data from Firebase
    return database.ref('exchange')
      .once('value')
      .then((snapshot) => {

        // Creating an object to load Firebase data into
        const exchanges = [];

        // Looping through Firebase data and loading the object
        snapshot.forEach((childSnapshot) => {
          exchanges.push(childSnapshot.val());
        });

        // Dispatching object to the Redux store.
        dispatch(setExchange(exchanges));

      });
  };
}
```

## Filters

*startEditCurrency* is a dispatching wrapper for the *setCurrency* action object. This method goes to firebase, edits the user's base currency, and loads an object to be passed and dispatched to the Redux store. *startSetCurrency* is a dispatching wrapper for the *setCurrency* action object. This method goes to firebase, grabs the necessary exchange data, and loads an object to be passed and dispatched to the Redux store. Other action objects have been commented in the code for their functionality.

```
//Creating a setStartDate action object for dispatching to Redux store
export const setStartDate = (startDate) => ({
  type: 'SET_START_DATE',
  startDate
});

//Creating a setEndDate action object for dispatching to Redux store
export const setEndDate = (endDate) => ({
  type: 'SET_END_DATE',
  endDate
});

//Creating a setDefaultFilter action object for dispatching to Redux store
export const setDefaultFilter = () => ({
  type: 'SET_DEFAULT_DATES',
  defaultState: {
    startDate: moment().startOf('day'),
    endDate: moment().endOf('day')
  }
});

//Creating a setCurrency action object for dispatching to Redux store
export const setCurrency = (currency) => ({
  type: 'SET_CURRENCY',
  currency
});

export const startEditCurrency = (newCurrency) => {
  return (dispatch, getState) => {

    //Get user id from Redux state
    const uid = getState().auth.uid;

    //Updating the currency under the specific users section in Firebase
    return database.ref(`users/${uid}/currency`)
      .update(newCurrency)
      .then(() => {
        //Dispatching setCurrency to Redux store.
        dispatch(setCurrency(newCurrency));
      });
  }
};

export const startSetCurrency = () => {
```

```

return (dispatch, getState) => {

  //Get user id from Redux state
  const uid = getState().auth.uid;

  return database.ref(`users/${uid}/currency`)
    .once('value')
    .then((snapshot) => {

      let currency = snapshot.val();

      //If user doesn't have a base currency, set default to USD
      if(currency == null) {
        currency = {
          id: "USD",
          exchange_rate: 1,
          name: 'United States Dollar',
          symbol: '$'
        }
      }

      //Dispatching setCurrency to Redux store.
      dispatch(setCurrency(currency));
    });
}
}

```

## Reducers

Below are the Redux reducer files. After each image is a small explanation of the reducer and/or the purpose of the file. Reducers are explained in more depth during the **Redux Diagram** section of this documentation.

### Auth

This is the Authentication data reducer. This class deals with dispatching actions to the auth section of the Redux state. The singular function in this class is the reducer. Action objects come into the function and the reducer edits the state accordingly.

```
export default (state = {}, action) => {
  switch(action.type) {
    //The LOGIN action.type updates the state to whatever the action.uid that is passed
    //This authenticates the user
    case 'LOGIN':
      return {
        uid: action.uid
      };
    //The LOGOUT action.type updates the state to empty, unauthenticating the user
    case 'LOGOUT':
      return {};
    //When action.type(s) aren't matched the current state is returned
    default:
      return state;
  }
};
```

## Cryptos

This is the Cryptos data reducer. This reducer deals with dispatching actions to the cryptos section of the Redux state.

```
// This is the default store state for all of the crypto data
const cryptosReducerDefaultState = {
  bitcoin: [],
  litecoin: [],
  ethereum: []
};

/*
  This function is the reducer
  action.type(s) come into the function, and the reducer updates the state accordingly
*/
export default (state = cryptosReducerDefaultState, action) => {

  switch(action.type) {
    //This updates state to action.cryptos
    case 'SET_CRYPTOS':
      return action.cryptos;
    //When there is no action, the current state is returned
    default: return state;
  }
};
```

## Exchange

This is the Exchanges data reducer. This reducer deals with dispatching actions to the exchanges section of the Redux state.

```
export default (state = [], action) => {
  switch(action.type) {
    //The SET_EXCHANGE action object updates the state to the exchanges attached to the action
    case 'SET_EXCHANGE':
      return action.exchanges;
    //When there is no action, the current state is returned
    default: return state;
  }
};
```

## Filters

This is the Filters data reducer. This reducer deals with dispatching actions to the filters section of the Redux state.

```
//This is the default state for the filters reducer
const filtersReducerDefaultState = {
  startDate: moment().startOf('day'),
  endDate: moment().endOf('day'),
  currency: {
    id: "USD",
    exchange_rate: 1,
    name: 'United States Dollar',
    symbol: '$'
  }
}

/*
  This function is the reducer
  action.type(s) come into the function, and the reducer updates the state accordingly
*/
export default (state = filtersReducerDefaultState, action) => {

  switch(action.type) {
    //SET_START_DATE updates the startDate on the Redux filter state
    case 'SET_START_DATE':
      return {
        ...state,
        startDate: action.startDate
      };
    //SET_END_DATE updates the endDate on the Redux filter state
    case 'SET_END_DATE':
      return {
        ...state,
        endDate: action.endDate
      }
    //SET_DEFAULT_DATES updates both the startDate and endDate on the Redux filter state
    case 'SET_DEFAULT_DATES':
      return {
        ...state,
        startDate: action.defaultState.startDate,
        endDate: action.defaultState.endDate
      }
    //SET_CURRENCY updates the base currency on the Redux filter state
    case 'SET_CURRENCY': {
      return {
        ...state,
        currency: action.currency
      }
    }
    default: return state;
  }
};
```



## Components

Below are the components created for Cryptotracker. If you would like to learn more about React components, please refer to the **Resources** section of this documentation.

### AccountPage

AccountPage is a React Component that contains the functionality for the Account Settings Page. This component contains a dropdown for updating the User's base currency.

```
export class AccountPage extends React.Component {

  constructor(props) {
    super(props);

    //Component state for updating the dropdown when it is clicked
    this.state = {
      currency: props.currency
    }
  }

  //This function is called when the dropdown is changed
  changeBaseCurrency = (e) => {
    //Finding currency object with same id from exchanges array
    const currency = this.props.exchanges.find((element) => {
      if(element.id === e.target.value){
        return element;
      }
    });
    //Updating component state to chosen currency
    this.setState({ currency });
    //Updating Redux state with chosen currency
    this.props.startEditCurrency(currency);
  }

  //Render function renders the components to the screen
  render() {
    return (
      <div className="page-container">
        <div className="account__title">
          <div className="container">
            <div className="account__title">
              <h1>My Account Page</h1>
            </div>
          </div>
        </div>
        <div className="container">
          <div className="account__option">
            <h2>Set Base Currency</h2>
            <Input s={5} type='select' value={this.state.currency.id}
onChange={this.changeBaseCurrency}>
          </div>
        </div>
      </div>
    );
  }
}
```

```

options
    //Inline that function loads all the currency data into the dropdown as
    this.props.exchanges.map((currency) =>
    <option key={currency.id} value={currency.id}>{currency.name}</option>
    )
    }
    </Input>
  </div>
</div>
</div>
);
}
}

/*
  This function maps Redux state to props that are passed into the component

  This function passes sorted exchange data from Redux state, and the current
  base currency in the filters data from Redux state to the component.
*/
const mapStateToProps = (state, props) => {

  const sortedExchange = sortExchanges(state.exchange);
  return {
    exchanges: sortedExchange,
    currency: state.filters.currency
  };
};

/*
  This function takes Redux dispatch actions and maps them to a function
  This new function is passed as a property in the component and is called in changeBaseCurrency
*/
const mapDispatchToProps = (dispatch) => {
  return {
    startEditCurrency: (currency) => dispatch(startEditCurrency(currency))
  };
};

//This creates a Higher Order Component letting the component access the Redux state and dispatch
actions
export default connect(mapStateToProps, mapDispatchToProps)(AccountPage);

```

## CryptoFilter

CryptoFilter is a React Component that contains a DateRangePicker. This component is used for updating the startDate and endDate filters in the Redux state.

```
export class CryptoFilter extends React.Component {

  //Component state for tracking if the DatePicker is focused
  state = {
    calendarFocused: null
  };

  //Calls Redux dispatch functions for chosen dates in DatePicker
  onDatesChange = ({ startDate, endDate }) => {

    //If startDate and endDate are null, the clear button was clicked
    //The current day is passed when the clear button is clicked
    if(!startDate && !endDate) {
      this.props.setStartDate(moment().startOf('day'));
      this.props.setEndDate(moment().endOf('day'));
    }
    else {
      this.props.setStartDate(startDate);
      this.props.setEndDate(endDate);
    }
  }

  //This is called when Date Picker focus is changed
  onFocusChange = (calendarFocused) => {
    this.setState(() => ({ calendarFocused }));
  }

  render() {
    return (
      <div className="date-container">
        <div className="container">
          <DateRangePicker
            startDate={this.props.filters.startDate}
            endDate={this.props.filters.endDate}
            onDatesChange={this.onDatesChange}
            focusedInput={this.state.calendarFocused}
            onFocusChange={this.onFocusChange}
            numberOfMonths={1}
            minimumNights={0}
            showClearDates={true}
            isOutsideRange={(day) => {
              const today = moment();
              return day.isAfter(today) || day.isBefore("2018-03-01");
            }}
          />
        </div>
      </div>
    )
  }
}

/*
```

```

    This function maps Redux state to props that are passed into the component

    This function takes the Redux state.filters and maps it to a filters object
    This new object is passed as a property to the component
*/
const mapStateToProps = (state) => ({
  filters: state.filters
});

/*
    This function takes Redux dispatch actions and maps them to startSetDate and setEndDate
    These new functions are passed as properties and are called in onDatesChange
*/
const mapDispatchToProps = (dispatch) => ({
  setStartDate: (startDate) => dispatch(setStartDate(startDate)),
  setEndDate: (endDate) => dispatch(setEndDate(endDate))
});
//This creates a Higher Order Component letting the component access the Redux state and dispatch
actions
export default connect(mapStateToProps, mapDispatchToProps)(CryptoFilter);

```

CryptoInfo is a React component that contains information on when it was the best time to buy and sell during a filtering period for a specific cryptocurrency.

```
export class CryptoInfo extends React.Component {

  constructor(props) {
    super(props);
  }

  render() {
    return (
      <div className="chart-info">
        <div className="container">
          <Row>
            <Col s={12} m={4} l={4}>
              <h1> Max Increase </h1>
              {this.props.maxDifference}
            </Col>
            <Col s={12} m={4} l={4}>
              <h1>Time To Buy</h1>
              <em>{moment(this.props.minElement.timestamp).format('ha MMM Do YYYY')}</em>
              <p>{this.props.minElement.amount}</p>
            </Col>
            <Col s={12} m={4} l={4}>
              <h1>Time To Sell</h1>
              <em>{moment(this.props.maxElement.timestamp).format('ha MMM Do YYYY')}</em>
              <p>{this.props.maxElement.amount}</p>
            </Col>
          </Row>
        </div>
      </div>
    );
  }
}

/*
  This function maps Redux state to props that are passed into the component

  This function relies on the getMaxMinData function that resides in selectors/cryptos
  The data is seperated into a object and the key values from the object are passed as props
*/
const mapStateToProps = (state, props) => {
  const data = getMaxMinData(state.cryptos, state.filters, props.name.toLowerCase());
  return { ...data }
};

//This creates a Higher Order Component letting the component access the Redux state
export default connect(mapStateToProps)(CryptoInfo);
```

## CryptolItem

CryptoItem is a React Component that contains all the functionality needed to graph a specific cryptocurrency data as well as showing the current price, current percentage increase, and trend data for the cryptocurrency.

```
export class CryptoItem extends React.Component {

  constructor(props) {
    super(props);

    //Component state to keep track of showing and hiding trend line.
    this.state = {
      showTrendLine: true
    }
  }

  render() {
    return (
      <div className="chart">
        <div className="chart-title">
          <div className="container">
            <div className="chart-title--container">
              <div>
                <Link className="chart-title__link"
to={` /filter/${this.props.name.toLowerCase()}`` >
                  <h1> {this.props.name} </h1>
                  </Link>
                </div>
                <div className="chart-title__amount">
                  <h1>
                    {this.props.currency_symbol} {this.props.current.amount}
                  </h1>
                </div>
                <div className="chart-title--percentage">
                  <h1 style={{
                    color:
                      (this.props.percentage >= 0) ? 'green' : 'red'
                  }}>
                    {this.props.percentage.toFixed(2)}%
                  </h1>
                </div>
              </div>
            </div>
          <div className="container">
            <div className="button-container">
              <button className="button--trend"
onClick={() => {
                //Function for updating state to show or hide trend line
                const show = this.state.showTrendLine;
                this.setState({showTrendLine: !show})
              }}>
                {
                  (this.state.showTrendLine) ? 'Hide Trend' : 'Show Trend'
                }
              </button>
            </div>
          </div>
        </div>
      </div>
    );
  }
}
```

```

        </button>
      </div>
      <div className="chart-container">
        <ResponsiveContainer height='100%' width='100%'>
          <LineChart data={this.props.data} margin={{top: 5, right: 10, left: 30, bottom:
5}}>
            <XAxis dataKey="timestamp"/>
            <YAxis tickSize={10} dataKey="amount" domain={["dataMin", "dataMax"]}
allowDataOverflow={true} />
            <CartesianGrid strokeDasharray="3 3" stroke="#FFDDED"/>
            <Tooltip/>
            <Legend />
            {
              //Inline function for showing and hiding trend line on graph
              (this.state.showTrendLine) && (
                <Line dot={false} type="monotone" dataKey="trend"
stroke="#E477D4" strokeWidth={4}/>
              )
            }
            <Line dot={false} type="monotone" dataKey="amount" stroke="#7F4276"/>
          </LineChart>
        </ResponsiveContainer>
      </div>
    </div>
  );
};

}

/*
  This function maps Redux state to props that are passed into the component
  This function relies on the selectCryptos function that resides in selectors/cryptos
  Crypto data is selected in the filtering range for the specific crypto
  The selected data is converted to the currency defined by the exchange rate
  Then the data is passed as props to the component
*/
const mapStateToProps = (state, props) => {

  let crypto = selectCryptos(state.cryptos, state.filters)[props.name.toLowerCase()];

  const exchangeRate = parseFloat(state.filters.currency.exchange_rate);

  //Converting crypto amounts to the exchange rate chosen from Redux state
  crypto = convertToExchange(exchangeRate, crypto);

  return {
    ...getTrendData(crypto),
    currency_symbol: state.filters.currency.symbol //The currency symbol for the base currency
  }

};

//This creates a Higher Order Component letting the component access the Redux state
export default connect(mapStateToProps)(CryptoItem);

```

## CryptoList

CryptoList is a wrapper React Component used for the Dashboard Page. This component renders a CryptoItem component for each name that is passed to it. The Dashboard page renders Bitcoin, Litecoin, and Ethereum data in a sequence and this is the component that achieves that functionality. This component doesn't have the JavaScript ES6 class definition for a React Component because it doesn't require all of the other functionality components of that nature need.

```
const CryptoList = () => (  
  <div>  
    <CryptoItem name='Bitcoin' />  
    <CryptoItem name='Litecoin' />  
    <CryptoItem name='Ethereum' />  
  </div>  
)  
  
export default CryptoList;
```



## DashboardPage

DashboardPage is a React Component that is responsible for resetting the Redux filters to the current day when it is rendered, as well as rendering an instance of CryptoList.

```
class DashboardPage extends React.Component {

  constructor(props) {
    super(props);
  }

  //Called before the component renders
  componentWillMount() {
    //Dashboard should only show the current day of data.
    //This dispatch resets the filters to the current day
    this.props.setDefaultFilter();
  }

  render() {
    return (
      <div>
        <div className="page-container">
          <CryptoList />
        </div>
      </div>
    );
  }
}

/*
  This function takes Redux dispatch actions and maps them to a function
  This new function is passed as a property in the component and is called in componentWillMount
  setDefaultFilter Redux action resets the startDate and endDate filters to the current day
*/
const mapDispatchToProps = (dispatch) => ({
  setDefaultFilter: () => dispatch(setDefaultFilter())
});

//This creates a Higher Order Component letting the component access the Redux dispatch actions
export default connect(null, mapDispatchToProps)(DashboardPage);
```

## ExchangePage

ExchangePage is a React Component responsible for allowing the user to check exchange rates. ExchangePage contains two dropdowns and an input field that allow the user to check exchange rates for many different types of currencies, including the cryptocurrencies from the application

```
export class ExchangePage extends React.Component {

  constructor(props) {
    super(props);

    this.state = {
      amount: 1, //Amount entered in field
      from_element: this.props.from_element, //Base Currency
      to_element: this.props.to_element, //Conversion Currency
      conversion: (amount) => {
        //This function converts the amount to the conversion currency
        let conv = (this.state.from_element) ? (amount / this.state.from_element.exchange_rate)
* this.state.to_element.exchange_rate : 0;
        //Floating point numbers give weird error when inputs are the same value, value should
be equal to amount
        if(this.state.from_element.id === this.state.to_element.id){
          conv = parseFloat(amount);
        }
        return conv.toFixed(2);
      }
    }
  }

  //Called when the Conversion currency dropdown is changed
  onToInputChange = (e) => {
    const target = this.props.exchanges.find((element) => element.id === e.target.value);
    this.setState({to_element: target});
  }

  //Called when the Base currency dropdown is changed
  onFromInputChange = (e) => {
    const target = this.props.exchanges.find((element) => element.id === e.target.value);
    this.setState({from_element: target});
  }

  //Called when the Amount field is changed
  onAmountChange = (e) => {
    let amount = e.target.value;
    if(!amount || amount.match(/^\d{1,10}(\.\d{0,2})?$/)){
      this.setState(()=> ({ amount }));
    }
  }

  render() {
    return (
      <div className="page-container">
```

```

        <div className="exchange-display">
        <div className="container">
            <h3> {(this.state.amount) ? this.state.amount : 0}
{this.state.from_element.name}</h3>
            <h3> {this.state.conversion(this.state.amount)}
{this.state.to_element.name}</h3>
        </div>
        </div>
        <div className="container exchange">
        <Row className="selector">
            <Col s={5}>
                <input
                    type="text"
                    placeholder="Amount"
                    value={this.state.amount}
                    onChange={this.onAmountChange}
                />
            </Col>
            <Col s={2} className="arrow">
                <Icon medium>arrow_forward</Icon>
            </Col>
            <div s={5}>
                <h3>{this.state.conversion(this.state.amount)}</h3>
            </div>
        </Row>
        <Row className="selector">
            <Input s={5} id="base-currency" type='select'
value={this.state.from_element.id} onChange={this.onFromInputChange}>
                {
                    this.props.exchanges.map((currency) =>
                        <option key={currency.id} value={currency.id}>{currency.name}</option>
                    )
                }
            </Input>
            <Col s={2} className="arrow">
                <Icon medium>arrow_forward</Icon>
            </Col>
            <Input s={5} id="conversion-currency" type='select'
value={this.state.to_element.id} onChange={this.onToInputChange}>
                {
                    this.props.exchanges.map((currency) =>
                        <option key={currency.id} value={currency.id}>{currency.name}</option>
                    )
                }
            </Input>
        </Row>
        </div>
        </div>
        );
    }
}
/*

```

It also takes two default exchange objects and passes them to props as well as the sorted exchange Redux data to be used in the dropdowns.

```
*/  
const mapStateToProps = (state, props) => {  
  
  //Sorting the dropdowns alphabetically  
  const sortedExchange = sortExchanges(state.exchange);  
  
  //Base currency to convert from  
  const BTC_EXCHANGE = {  
    "exchange_rate": "0.00011300",  
    "id": "BTC",  
    "name": "Bitcoin",  
    "symbol": "BTC"  
  };  
  
  //Conversion currency to convert to  
  const USD_EXCHANGE = {  
    "exchange_rate": "1.00",  
    "id": "USD",  
    "name": "United States Dollar",  
    "symbol": "$"  
  };  
  
  return {  
    exchanges: sortedExchange,  
    from_element: BTC_EXCHANGE,  
    to_element: USD_EXCHANGE  
  };  
};  
  
//This creates a Higher Order Component letting the component access the Redux state  
export default connect(mapStateToProps)(ExchangePage);
```

## Footer

Footer is a React Component that is responsible for rendering the footer contents on every page.

```
const Footer = () => (  
  <div className="footer">  
    <div className="container">  
      <p>If you are interested in how this project was made, check out the  
        <a href="#">Documentation</a></p>  
    </div>  
    <div className="footer-end">  
      <div className="container">  
        <p>This project was created by <a href="http://www.jacobsickels.com">Jacob  
Sickels</a></p>  
      </div>  
    </div>  
  </div>  
);  
  
export default Footer;
```

## NotFoundPage

NotFoundPage is a React Component responsible for showing an error message when a page is navigate to that doesn't exist in AppRouter. In this case, a dummy 404 message is shown and prompts the user to go back to the Dashboard.

```
const NotFoundPage = () => (  
  <div>  
    <Header />  
    <div className="page-container">  
      <div className="container">  
        <h1>We're Sorry! This page doesn't exist </h1>  
        <p>Error Code: 404 - <Link to="/">Go Home</Link></p>  
      </div>  
    </div>  
    <Footer />  
  </div>  
);  
  
export default NotFoundPage;
```

## Header

Header is a React Component that is responsible for the navigation of the application. There are many buttons in the Header component that *Link* other pages. When a *Link* is clicked in the Header, the url is appended to and the AppRouter serves the correct page associated with the url. See AppRouter for more information.

```
export const Header = ({ startLogout }) => (  
  <header className="header">  
    <div className="container">  
      <div className="header__content">  
        <Link className="header__title" to="/dashboard">  
          <h1>Cryptotracker</h1>  
        </Link>  
        <div>  
          <Link className="button button--link" to="/dashboard">Dashboard</Link>  
          <Link className="button button--link" to="/exchange">Exchange</Link>  
          <Dropdown  
            trigger={  
              <button data-beloworigin="true" className="button button--link">Filter</button>  
            }>  
            <Link to="/filter/bitcoin">  
              <button className="button--dropdown">Bitcoin</button>  
            </Link>  
            <Link to="/filter/litecoin">  
              <button className="button--dropdown">Litecoin</button>  
            </Link>  
            <Link to="/filter/ethereum">  
              <button className="button--dropdown">Ethereum</button>  
            </Link>  
          </Dropdown>  
          <Dropdown  
            trigger={  
              <button data-beloworigin="true" className="button button--link">Account</button>  
            }>  
            <Link to="/account">  
              <button className="button--dropdown">Settings</button>  
            </Link>  
          <button id="logout" className="button--dropdown" onClick={startLogout}>Logout</button>  
        </Dropdown>  
      </div>  
    </div>  
  </header>  
)  
;  
  
/*  
  This function takes Redux dispatch actions and maps them to a function  
  The startLogout function is called when the Logout button is clicked in the Header  
*/  
const mapDispatchToProps = (dispatch) => ({  
  startLogout: () => dispatch(startLogout())  
});  
  
//This creates a Higher Order Component letting the component access the Redux dispatch actions  
export default connect(undefined, mapDispatchToProps)(Header);
```

## LoadingPage

LoadingPage is a React Component responsible for showing the loading icon when a page is loading. This component is used in App.js for when the application is loading.

```
const LoadingPage = () => (  
  <div className="loader">  
      
  </div>  
)  
  
export default LoadingPage;
```

## LoginPage

LoginPage is a React Component responsible for showing the Login Page content. This page is served from the Approuter when the user isn't authenticated.

```
export const LoginPage = ({ startLogin }) => (  
  <div className="login">  
    <div className="login__box">  
      <div className="login__image">  
          
      </div>  
      <h1 className="login__title">CryptoTracker</h1>  
      <p className="login__subtitle">Tracking cryptos, one hour at a time.</p>  
      <button className="button" onClick={startLogin}>Login with Google</button>  
    </div>  
  </div>  
)  
  
/*  
  This function takes Redux dispatch actions and maps them to a function  
  The startLogin function is called when the (Login with Google) button is clicked  
*/  
const mapDispatchToProps = (dispatch) => ({  
  startLogin: () => dispatch(startLogin())  
});  
  
//This creates a Higher Order Component letting the component access the Redux state  
export default connect(undefined, mapDispatchToProps)(LoginPage);
```

## SingleCrypto

Single Crypto is a React Component responsible for showing the contents of the Filtering page for a specific cryptocurrency. SingleCrypto shows a CryptoFilter component, CryptoItem component, and a CryptoInfo component for a specific cryptocurrency. This page is navigated to whenever a “/filter/:currency” is matched in AppRouter. The currency to show is retrieved from the props in mapStateToProps.

```
export class SingleCrypto extends React.Component {

  constructor(props) {
    super(props);
  }

  render() {
    return (
      <div>
        <CryptoFilter />
        <CryptoItem name={this.props.name} />
        <CryptoInfo name={this.props.name} />
      </div>
    );
  }
}

/*
  This function maps Redux state to props that are passed into the component

  This function gets the name of the currency matched from AppRouter and gets
  the matched currency parameter and passes it as props to SingleCrypto. This
  name prop is passed to CryptoItem and CryptoInfo for use in themselves.
*/
const mapStateToProps = (state, props) => {
  return {
    name: props.match.params.currency.charAt(0).toUpperCase() +
    props.match.params.currency.slice(1)
  };
};

//This creates a Higher Order Component letting the component access the Redux state
export default connect(mapStateToProps)(SingleCrypto);
```



## Selectors

### Cryptos

The *selectCryptos* method takes in an array of crypto data, and a filters object that contains a start and end date. The array of crypto data is sorted greatest to least using the start and end date filters. The *getMaxMinData* takes in crypto data array, a filters object of start and end date, and a cstring crypto name. The function is responsible for getting the data for determining when the best time to buy and sell a cryptocurrency during a period would be.

```
export const selectCryptos = (cryptos, {startDate, endDate}) => {
  //Creating a filtered object to store data
  const filteredCryptos = {
    bitcoin: [],
    litecoin: [],
    ethereum: []
  };
  //Looping through cryptos and storing them according to startDate and endDate
  for(var prop in cryptos) {
    filteredCryptos[prop] = cryptos[prop].filter((crypto) => {
      const createdAtMoment = moment(crypto.timestamp);
      const startDateMatch = startDate ? startDate.isSameOrBefore(createdAtMoment, 'day') : true;
      const endDateMatch = endDate ? endDate.isSameOrAfter(createdAtMoment, 'day') : true;
      return startDateMatch && endDateMatch;
    });
    //filteredCrypto is sorted by (unix) timestamp
    filteredCryptos[prop].sort((a, b) => {
      return a.timestamp - b.timestamp;
    });
  }
  return filteredCryptos;
};

export const getMaxMinData = (cryptos, filters, name) => {
  //Selecting the specific crypto with name
  let crypto = selectCryptos(cryptos, filters)[name];
  //Converting the crypto to the base currency of user from filters
  crypto = convertToExchange(filters.currency.exchange_rate, crypto);
  let maxDifference = 0;
  let minElement = {};
  let maxElement = {};
  //Loops through cryptos to get the range of max difference between two points
  for(var i = 0; i < crypto.length; i++) {
    for(var j = i + 1; j < crypto.length; j++) {
      let check = crypto[j].amount - crypto[i].amount;
      if(check > maxDifference) {
        maxDifference = crypto[j].amount - crypto[i].amount;
        minElement = crypto[i];
        maxElement = crypto[j];
      }
    }
  }
  return { maxDifference: maxDifference.toFixed(2), minElement, maxElement }
}
```

### Exchanges

This default method is the *sortExchanges* function used in other components. This function is responsible for sorting the exchange data alphabetically. The *convertToExchange* function is responsible for converting an array of cryptocurrency data to the exchangeRate that is passed in.

```
export default (exchanges) => {

  //Copying exchanges array to temp array for manipulating
  const filteredExchanges = exchanges.slice(0);

  //Sorting filteredExchanges by name
  filteredExchanges.sort((a, b) => {
    if(a.name > b.name)
      return 1;
    else if( b.name > a.name)
      return -1;
    else return 0;
  });

  //Returning filteredExchanges array
  return filteredExchanges;
};

export const convertToExchange = (exchangeRate, cryptos) => {

  //Map function going through each object in cryptos array and converting it to the exchange
rate
  const converted = cryptos.map((entry) => ({
    amount: (parseFloat(entry.amount) * exchangeRate).toFixed(2),
    timestamp: entry.timestamp
  }));

  //Returning converted cryptocurrency array
  return converted;
}
```

This default function *getTrend* is responsible for getting the trendline data used in the graphs. This trendline is the line of best fit for the data points. The mathematics for this algorithm was found at

[https://www.varsitytutors.com/hotmath/hotmath\\_help/topics/line-of-best-fit](https://www.varsitytutors.com/hotmath/hotmath_help/topics/line-of-best-fit).

```
export default (crypto) => {
  //Array for datapoints on x axis
  let x = [];
  //Pushing 0->crypto.length to x datapoints array
  for(let i = 0; i < crypto.length; i++) {
    x.push(i);
  }
  //Summing all x values
  const xSum = x.map(num => num).reduce((prev, next) => prev + next);

  //Summing all amounts (on y axis)
  const ySum = crypto.map(entry => parseFloat(entry.amount)).reduce((prev, next) => prev + next);
  //Getting average x value
  const averageX = xSum / crypto.length;
  //Getting average y value
  const averageY = ySum / crypto.length;
  //Getting top sum of slope function
  const topSlopeArray = [];
  for(let i = 0; i < crypto.length; i++) {
    topSlopeArray.push((x[i] - averageX) * (crypto[i].amount - averageY));
  }
  const topSlope = topSlopeArray.reduce((prev, next) => prev + next);
  //Getting bottom sum of slope function
  const bottomSlope = x.map(num => (num - averageX) * (num - averageX)).reduce ((prev, next) =>
prev + next);
  //Calculating slope
  const slope = topSlope / bottomSlope;
  //Calculating y-intercept
  const yIntercept = averageY - (slope * averageX);
  //Calculating percentage changed between first element and last element
  const percentage = (crypto[crypto.length - 1].amount - crypto[0].amount) / crypto[0].amount *
100;

  //Creating objects and putting them into data array
  const data = crypto.map((entry, index) => ({
    timestamp: moment(entry.timestamp).format('ha'),
    amount: parseFloat(entry.amount),
    trend: slope * index + yIntercept
  }));

  return {
    data,
    percentage,
    current: crypto[crypto.length - 1]
  };
};
```

## AppRouter

```
export const history = createHistory();

export const AppRouter = (props) => (
  <Router history={history}>
    <div>
      <Switch>
        <PublicRoute
          path="/"
          component={LoginPage}
          exact={true}
        />
        <PrivateRoute
          path="/dashboard"
          component={DashboardPage}
        />
        <PrivateRoute
          path="/exchange"
          component={ExchangePage}
        />
        <PrivateRoute
          path="/account"
          component={AccountPage}
        />
        <PrivateRoute
          path="/filter/:currency"
          component={SingleCrypto}
        />
        <Route
          component={NotFoundPage}
        />
      </Switch>
    </div>
  </Router>
);

export default AppRouter;
```

AppRouter is the client-side router for Cryptotracker. If you want to learn more about client-side routing please refer to the **Client-side Routing** section of this documentation.

## Tests

Significant testing code is explained during the **Testing Suite** part of this documentation. If you would like to see all of the testing functions you can find them at:

<https://github.com/JacobSickels/Cryptotracker/tree/master/src/tests>

# User Manual

## Location

Cryptotracker can be found at <http://crypto.jacobsickels.com/>

## Terminology

**Filter or Filtering:** Data on Cryptotracker is displayed in a date range. Data is always shown between a start date and an end date. Whenever *filter* is mentioned, this will refer to the start date and end date filters that the cryptocurrency data is *filtered* with and displayed.

**Base Currency:** On the Account Page, a user can edit the currency that is displayed throughout the application. The default currency for the application is USD, but the user can change the *base currency* for the application and it will automatically be converted.

**Component:** A React Component, a part of the user interface.

## Functionality

Cryptotracker provides all the following functionality from project deliverables:

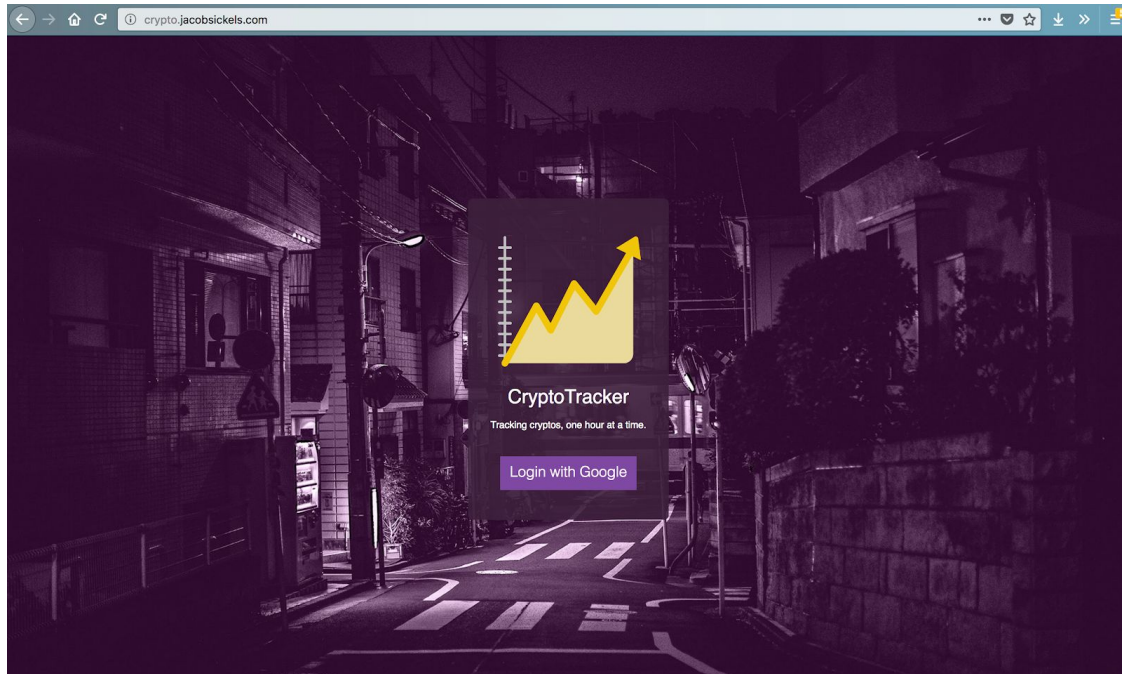
- Provides per hour data on popular cryptocurrencies
- Graphs data on cryptocurrencies on a per day basis through the Dashboard Page
- Calculates trend lines for graphed data
- Provides data on when it was a best time to buy and sell during a period on Filters Page
- Allows the user to select a length of time they want to filter crypto data
- Allows the conversion between not only cryptocurrencies but all currency types
  - This functionality is achieved through the User setting a base currency for the app in their account settings

## Pages

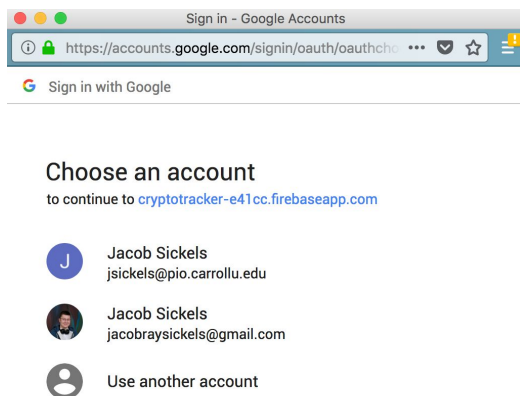
The following section describes the pages and navigation systems a user can interact with on Cryptotracker.

### Login Page

When a User first enters the application and isn't logged in, they are presented with the Login Page (seen below).

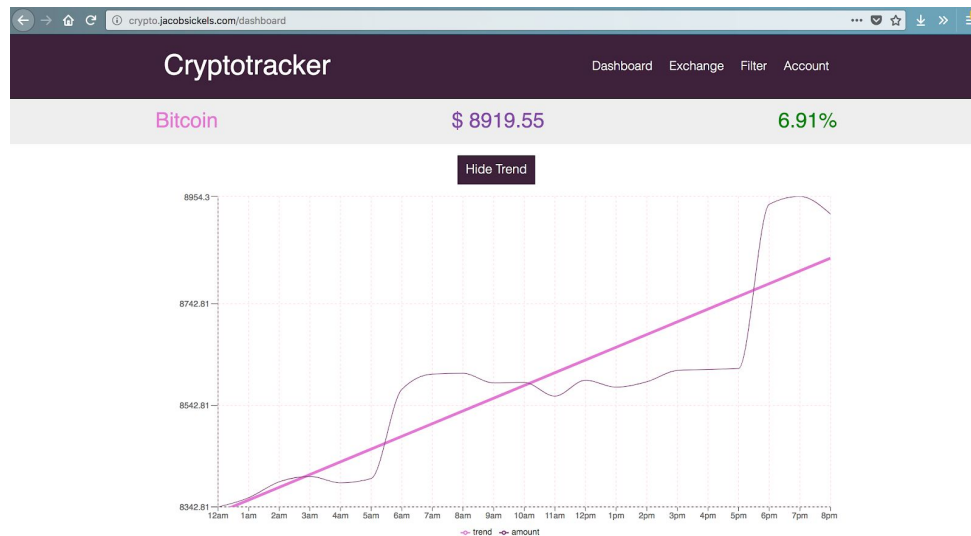


To log in to the application, the User needs to click the *Login with Google* button. The User is then prompted with a Google Authentication Login (seen below).



Select, or enter, a Google Login. Once this is done the app will redirect the User to the Dashboard Page.

## Dashboard Page

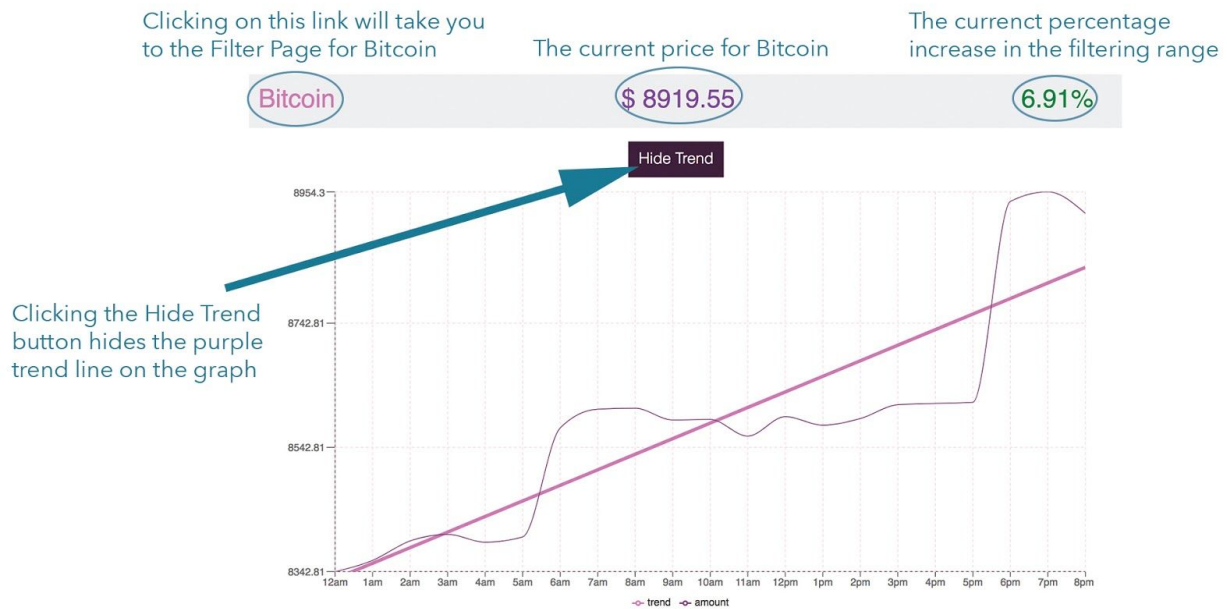


The Dashboard Page is the main hub of the application.



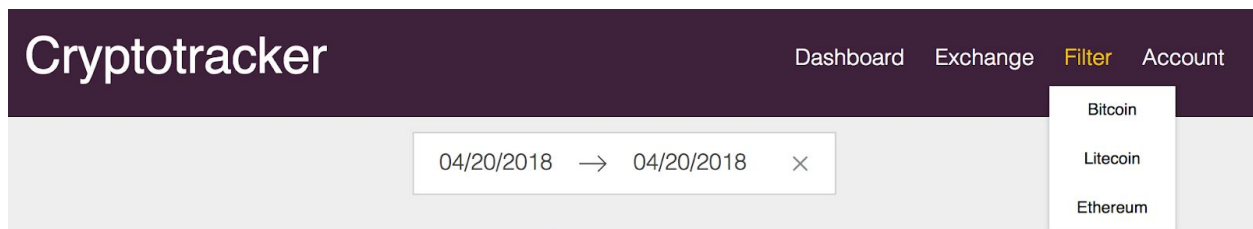
There are four sections you can access in the application: Dashboard (where the user is currently), Exchange, Filter, and Account. The other three sections are described in more detail below. The Dashboard Page contains the current day of data for three different cryptocurrencies: Bitcoin, Litecoin, and Ethereum.

All of the currencies on the Dashboard Page are interacted with the same way, so we will use Bitcoin as the example for navigation objects. In addition, cryptos on the Dashboard Page are always filtered under the current day, so only the current day of data is displayed.



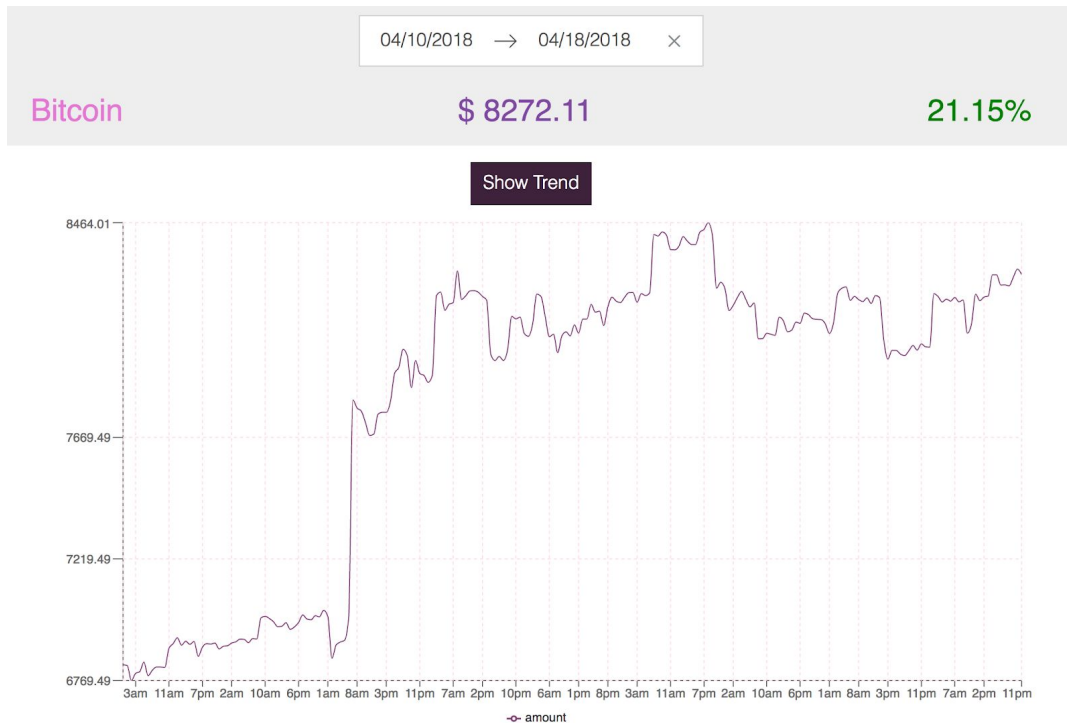
## Filter Page

There are two different ways to navigate to a filter page on a cryptocurrency. You can click on the name of the cryptocurrency as suggested above, or you can click on filter in the navigation bar. When this is done, a dropdown gives you the choice to click on a specific currency you would like to filter. See an example of this below.



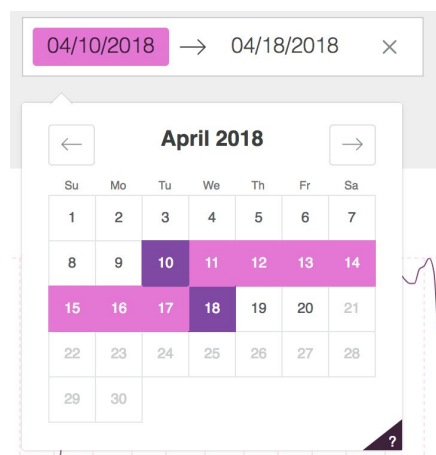
Most of the page is taken up by the same cryptocurrency graph component. However, there is an additional component at the top of the page, a Date Range Picker. This Date Range Picker allows the user to change the start and end date that the cryptocurrency is filtered under. Once this is done the graph updates, showing the user the range of data points that fall under the filtering range. See an example of this below.





In this example, the date range has been changed to 4/10/2018 to 4/18/2018, and the Hide Trend button has been clicked, removing the purple trend line.

When the Date Range Picker is clicked, a window like the one below is displayed.



This window lets you choose the days for which you would like to filter data. I've restricted the start date to 3/1/2018, as this is the chosen application epoch.

The top display bar of data has also changed as the date range has changed. The current price of Bitcoin takes the very last datapoint in the dataset that is graphed, so this number has

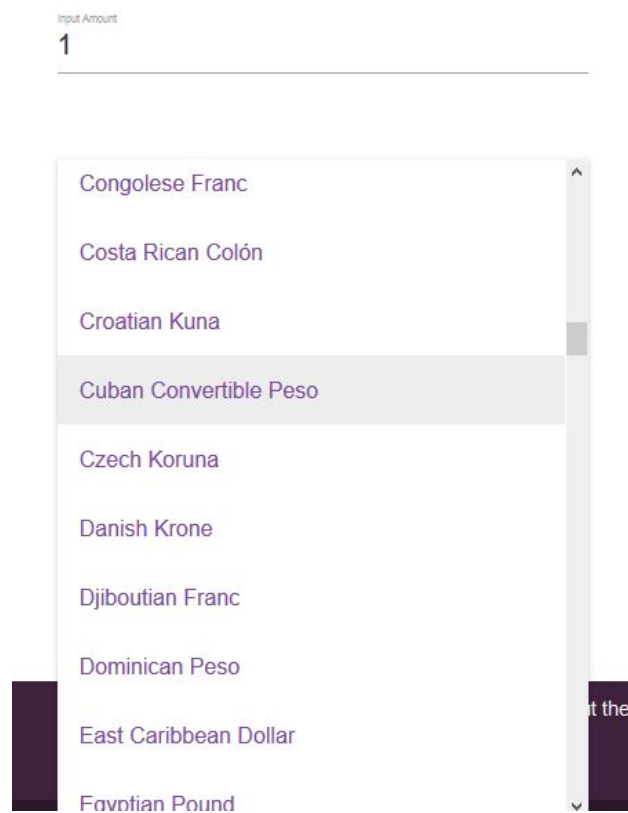
changed to \$8272.11. The percentage increase has also changed. This percentage increase takes the increase (or decrease) from the very first and very last numbers in the graphed dataset, so this number has changed to 21.15%.

## Exchange Page

The Exchange Page can be accessed from the navigation bar.



The Exchange Page offers a couple of options for checking the exchange rates between currencies. The base currency option on the page is Bitcoin to United States dollar. If the user would like to change the conversion or base currencies, they can click the dropdown and choose a different currency for that section.

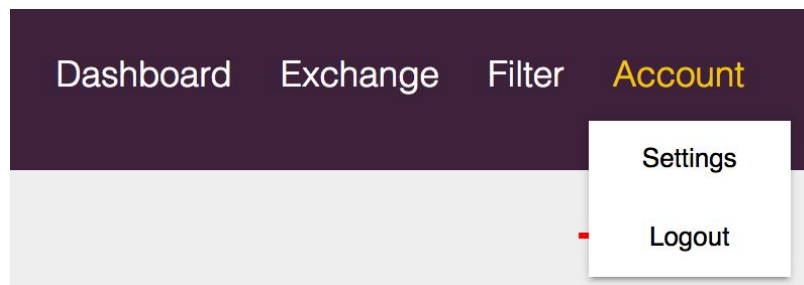
A screenshot of the Exchange Page. At the top, there is an input field labeled 'Input Amount' with the number '1' entered. Below this is a dropdown menu with a list of currencies. The currencies listed are: Congolese Franc, Costa Rican Colón, Croatian Kuna, Cuban Convertible Peso (which is highlighted), Czech Koruna, Danish Krone, Djiboutian Franc, Dominican Peso, East Caribbean Dollar, and Egyptian Pound. The dropdown menu is open, showing the list of currencies.

If the User would like to update the base amount on the Exchange page, they can click into the amount field and type in a new amount.

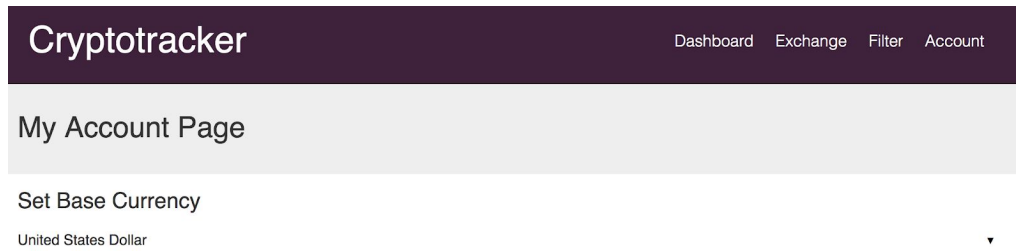
The image shows a dark gray rectangular box representing a Bitcoin wallet or balance. Inside the box, the text "1234 Bitcoin" is displayed in white, with "10920353.98 United States Dollar" below it. Below this box, there is a conversion interface. It features two input fields. The first field is labeled "Input Amount" in small green text and contains the value "1234". To its right is a right-pointing arrow. The second field contains the value "10920353.98". Below the first field is a dropdown menu currently showing "Bitcoin". To its right is another right-pointing arrow. The second dropdown menu is currently showing "United States Dollar".

#### Account Page (and Logging Out)

The Account Settings Page can be accessed from the navigation bar. When you click on the account button a dropdown gives you two options: Settings and Logout.



If you would like to log out of the application you can do it by clicking the Logout button. This will redirect you back to the login page. However, if you click on the Settings button, you will be directed to the Account Page.



The Account Page let's you set the base currency you would like to see throughout the application. This means that graphs and amounts on the application will be converted to this base currency. The default base currency in the application is USD, but if you would like to change this, just select the dropdown and choose the currency you would like. This option is stored on your account, and the settings are applied when you log into the application from anywhere.

# Project Journal and Timesheet

Research / Tutorial	Design	Implementation	Documentation / Testing
---------------------	--------	----------------	-------------------------

Date	Hours	Description
2/5/2018	1.5	React Research / Tutorial
2/6/2018	2	React Research / Tutorial
2/7/2018	2.5	React Research / Tutorial
2/8/2018	2	React Research / Tutorial
2/9/2018	2	React Research / Tutorial
2/10/2018	3	React Research / Tutorial
2/11/2018	3.5	React Research / Tutorial
2/12/2018	2	React Research / Tutorial
2/13/2018	2	React Research / Tutorial
2/14/2018	2	React Research / Tutorial
2/15/2018	2	React Research / Tutorial
2/16/2018	2	React Research / Tutorial
2/17/2018	2.5	React Research / Tutorial
2/18/2018	3	React Research / Tutorial
2/19/2018	2	React Research / Tutorial
2/20/2018	2.5	React Research / Tutorial
2/21/2018	3	React Research / Tutorial
2/22/2018	4.5	React Research / Tutorial

8		
2/23/2018	8	1 Designing Firebase Schema
2/24/2018	8	1 Designing Firebase Schema
2/25/2018	8	1.5 Designing Router Schema/ Some Page Design
2/26/2018	8	2 Setup Firebase schema, getting ready for cron implementation
2/27/2018	8	1.5 Cron research
2/28/2018	8	3 Cron research and actualization, using Google Cloud SDK and functions-cron from github
3/1/2018		1.5 Working on Page Designs
3/2/2018		1 Working on Page Designs
3/3/2018		1 Working on Page Designs
3/4/2018		2 Added way to get data from firebase to application through Redux store. Created test cases to suit additions
3/5/2018		1.5 Working on Page Designs
3/6/2018		2.5 Added components and filters for redux store data. Spotted error in cron jobs and am exploring fixes.
3/7/2018		2 Header and Footer component Design
3/8/2018		1 Styling
3/9/2018		1 Styling and component design
3/10/2018	8	3 Foundation components for singling out crypto, worked on router and navigation
3/11/2018	8	2 Added components for singling out a crypto, added some styling
3/12/2018	8	1.5 CryptoFilter component fixes
3/13/2018	8	1 Creation of CryptoInfo component foundation
3/14/2018	8	2 Some route bug fixing, and 404 error fixing
3/15/2018	8	1 Worked on 404 page styling
3/16/2018	8	3 Added ability to filter crypto data by a range of dates

3/17/2018	3	Added calculations for trend lines on data
3/18/2018	2.5	Worked on some styling for trend lines with materialize
3/19/2018	2.5	Added showing and hiding trendlines on graphs
3/20/2018	1.5	Some Page Creation
3/21/2018	1	Page and component Creation
3/22/2018	1.5	Component Creation
3/23/2018	0	Break
3/24/2018	0	Break
3/25/2018	3	Worked on creating Exchange page, including logic
3/26/2018	0	Break
3/27/2018	3.5	Worked on styling the login page, and some bug fixes
3/28/2018	2	Diagnosing bug with loading into dashboard for first time
3/29/2018	2.5	Diagnosing and fixing bug when loading into dashboard (check trello for fix)
3/30/2018	1.5	Designing and coding function for best time to buy and sell inside cryptoinfo
3/31/2018	2	Implementing and styling best time to buy and sell in period
4/1/2018	3	Implementing functionality for users to set the base currency on their account
4/2/2018	2	Setup foundation for testing suite as well as wrote some test cases, also fixed some styling uses and a small filter issue with CryptoInfo
4/3/2018	2.5	Writing test cases for components
4/4/2018	1.5	Documentation
4/5/2018	1	Documentation
4/6/2018	2	Test Cases
4/7/2018	1	Documentation
4/8/2018	2	Test Cases

4/9/2018	2	Test Cases
4/10/2018	1	Test Cases
4/11/2018	1.5	Documentation
4/12/2018	1	Test Cases
4/13/2018	2.5	Test Cases (Found error where Materialize doesn't want to shallow render)
4/14/2018	2	Documentation
4/15/2018	1.5	Documentation
4/16/2018	4	Test Cases and Documentation
4/17/2018	1.5	Test Cases and Documentation
4/18/2018	4.5	Test Cases and Documentation (Finished test cases)
4/19/2018	2.5	Documentation
4/20/2018	2	Documentation
4/21/2018	3.5	Documentation and looking through code comments
4/22/2018	2	Documentation
4/23/2018	5	Documentation
4/24/2018	2	Finishing Documentation

Research / Tutorial	44
Design	17
Implementation	54
Documentation / Testing	46
Total	161





# Reflection

This project was absolutely one of the hardest things I have ever had to do in my schooling career. From beginning to end, every part of the project needed to be planned out, both time and software design needed to be watched very carefully and scrutinized. This was one of the hardest things for me to get over, but in the long run I absolutely think it was beneficial. Throughout my time here at Carroll, not many computer science problems eluded me. Many times, I understood the problem, so I would just start working on the solution without any planning. This problem was different, and for good reason. I wanted to teach myself a different programming language throughout the creation of this project, and this gave me some pause. I needed to be able to plan every aspect of this project before delving into its creation because there were many things that I had no idea how to accomplish. Planning these parts out before-hand, and giving myself ample time for tutorials and research really benefited the timing, planning, and actualization of this project. In the end, I was able to accomplish everything I set out to do with this project, and I was able to add a couple of things along the way that I didn't know where possible in the timeframe.

The information gathering and research section of this project took a significant amount of time, and it ended up being almost 25% of the full time spent on the project. I found a very good tutorial online that went through just about everything I would want to know about React and using it for web development. Finding this resource was imperative to the creation of this project because of the vast amounts of information I was able to obtain from it. The tutorial itself consisted of about 40 hours of videos that took me through building some smaller applications using React and introduced me to many other application tools such as Redux and Firebase. The tutorials ended up touching on many of the things I wanted to accomplish with my project, so I was able to use the things I learned from the tutorials almost directly into the creation of my project.

Program design was also heavily inspired by the tutorials that I did in the information gathering section of this project. For the most part, React-Redux applications are split into three main sections: Redux for application state, React for User Interface, and Firebase for database. I gleaned this from the tutorials, and set up my application to do just about the same thing. However, a main difference ended up being how I was going to incorporate the data collection into the main program design. I ended up deciding that a separate service could be used to write the data to Firebase, and this would inherently decouple the main application from the service that collects the data.

Interface design was the part of the application that came the most easily to me. My background in web design helped me create a fairly easy to use design, that I feel translated very nicely to code and to the screen. I did the interface design after learning the information gathering section to more deeply understand how React works, and this helped inform the creation of the user interface and components.

The implementation phase of this project was one of the larger sections, and one of the greater challenges of this project. I gave myself four weeks to complete this section, but it's one of those things that I feel is never going to be complete. For the most part yes, this section was

completely relatively on time, but there have been some small aspects of the project that I have been working on here and there to make them better since the completion of the section. As for difficulty, this section was absolutely the most difficult. Having learned what I needed to from the information gathering section, React components are relatively separate entities, but they can be coupled together in parent components. Knowing this, the implementation phase turned into structuring components by page first, and then creating the components that go onto a page. I also had to make sure that I wasn't repeating code between components, as components can be shared. This knowledge helped with the creation of the Filter Page because the graphing component from the Dashboard Page can be used on this page as well, and just passed different data for graphing. Overall, this section was a bit tricky, just due to the sheer mass of information and freedom, but I was able to reign it in and finish the main deliverables on time.

During the implementation phase, I would write down any additional functionality that I wanted to add to the application. Below are some of the additional functions that I wrote down and whether or not I was able to complete them.

### **Account Settings**

In my deliverables, I wanted to create a section of the application that let users check the exchange rates of different currencies. This section ended up being the Exchange Page, however, as I was programming this I thought that having a way to set a base currency for the application would be a cool idea. One of the tutorial applications I did had a way to store data under accounts. I used this implementation to store a base currency under the user's account, and load that base currency when they enter the application. This effectively creates a user setting for base currency. I created an Account Page for future additions to user settings. *I completed this addition.*

### **Filtering by Hour**

In the beginning of this application, I was only interested in filtering cryptocurrencies by day and seeing their rise and fall. But during development, I thought it might be interesting to implement a way to also sort by a specific hour in the day instead of just the beginning and end of the day. *I was not able to implement this feature due to time constraints.*

### **Currency Symbols**

This addition hinged on the **Account Settings** future addition and whether or not that was completed. The original base currency of the application was USD, so I thought there wasn't really a need to show any other currency symbols. However, after the completion of the **Account Settings**, I added currency symbols to the exchange section of the database so that I could grab them whenever a currency was exchanged, or grab it from the base currency on the user's account to display on the Dashboard Page. *I completed this addition.*

### **Mobile Application (Responsive Design)**

Although this is primarily intended as a desktop application, I am displaying it on a website which can be accessed by a smartphone. So, I wanted to style the website as much as I could to be *responsive*, so that any size device can access it. Much of the site *does* work on mobile, but some inputs still don't work and are currently logged as future bug fixes. *I was able to do some work for mobile, but there are still bugs.*

The testing for this application was one of the easier sections, but it also took a longer time than expected. I wanted to write many of the test cases for my testing suite as I wrote the components they were testing, effectively working with Test Driven Development, but that wasn't as easy as I thought. I did write *some* of the test cases for the components I was working on but many of the smaller methods, particularly for testing Redux actions and reducers had to wait until the end of the application, because of time constraints. For the most part, I was able to run the Testing Suite at the same time as the development server, so I knew if a change I made was breaking other parts of the application. However, the rest of the test cases were written and full testing of the application was completed during this section of development.

In conclusion, in my opinion, the format of this project is what makes it excel. Having me sit down and really plan out this project as part of the grading criteria really helped flush out what needed to be done. Really planning out how the application is going to work before programming it is what has made this project successful, without it, this project would never have been finished on time in the form it is currently.

# Capstone Evaluation Rubrics Self Evaluation

## **Project Appropriateness - 5**

For this project I challenged myself to learn a new programming language to build this application. Web applications are a domain that weren't covered during the computer science curriculum here at Carroll. Overall I feel that the project size and scope are well suited for a Capstone project.

## **Project Planning and Follow-through - 5**

All project deliverables were actualized, and almost 25% of the project time was spent on Research and learning about the new language, React. Project planning was well defined, and a detailed project log was created for day-to-day updates.

## **Soft Analysis and Design - 4**

I spent a lot of time in Research and Design for this project, I wish I had more practice with design principles and diagramming so this section was scored lower.

## **Code Quality - 5**

I feel that the code created was well put together and commented. The code should be very readable and the organization should be very easy to follow. Components were created in separate files, application state has been abstracted to Redux, and proper variable names were used for JavaScript.

## **Software Usability - 5**

The web application aspect of this project makes it very usable and user friendly. The application works much like a website so the user interface is pretty self explanatory. Some effort was put forth to also make the application mobile friendly.

## **Project Verification - 5**

I created a testing suite to accent this project, so the testing of each aspect of the project was very thorough.

## **Effort - 4**

I feel that learning a new language before the creation of this project made the effort required for this project much higher. In addition, one of the main aspects of this project was wanting to learn React as well as creating the application using the new language. So, the effort required to create this project was considered carefully even before the project started.

## **Binder - 4**

Documentation was created to an extent that I feel is appropriate, but I feel that diagrams and user manual could have been done a little better. I intend to create a web portal for the

documentation later on.

**Presentation - 4\***

I've given myself a 4 on this section because I am unsure about how the presentation will turn out and this documentation is submitted before the actual presentation. I anticipate it should be a good presentation and it will cover the materials I need for explaining Web Application design and this application as a whole.