Application Design for CoinWatch
Jason Crouse and Connor Pelkey
CS4550
11/05/17

Overview

CoinWatch is a cryptocurrency price-monitoring website. The site connects to exchange APIs in order to provide realtime updates on cryptocurrency prices across various exchanges. Users can follow select cryptocurrencies in order to receive live updates on price changes on their main feed. Users can also sign up for email alerts when a currency's value passes a desired amount.

User Stories

Our app will only have two user roles: a normal user and an admin. The admin won't have many extra responsibilities or permissions, but is really there in case anything goes horribly wrong so they can get rid of data or users that are not functioning properly. As a whole, normal users will have access to all of the desired functionality of the app and there are not really any extra features that are restricted to admins or anyone else.

- As a new user, I can create an account.
- As a returning user, I can log in/log out.
- As a normal user, I can see a listing of all the currencies that are available for me to follow.
- As a normal user, I can choose certain currencies to follow/unfollow
- As a normal user, I can sign up for and receive email updates when a currency reaches a desired value
- As a normal user, I can change my email address to keep it up to date for notifications.
- As an admin user, I can remove users, maybe remove available currencies(?)

Data Design

We will keep a database of users and will use the Cryptowatch API to get data for various cryptocurrencies. Users provide their own identification data on registration (such as usernames and passwords) and more data is tracked for each user after they register and begin to interact with the site and its features.

- Users:
  Fields:
  1. email
  2. password (hash)

3. username
4. the markets they follow (through a join table)

- Markets:
  Fields:
  1. rate
  2. timestamp of their last update
  3. exchange name
  4. asset pair
  5. users who watch (through a join table)

- MarketUser (Join table):
  Fields:
  1. user id
  2. market id

App Interface

The most important page for users is the main feed, which is where users will get live(ish) updates on the price of the cryptocurrencies they select to watch. Updates will actually come every 10-20 seconds so as to not require constant updating. There will also be an index of all available currencies to allow user to have an idea of the range of currencies offered so they will have the ability to pick a few to follow on the main feed. We will likely implement some sort of account settings so users can change things about their account, especially since we are offering the ability to receive email notifications and we want users to be able to keep their email address up to date.

There will a be a landing page for new or logged out users which will provide an overview of the app and links for signing in, signing up, and a demo page that will display mock user and market data (a mock currency watch page).

Potential additional page includes a news feed, where users would be able to view the latest cryptocurrency blogs and articles from around the web. Implementation of this is dependent upon building a finished app with enough time to spare. There could additionally be a blog section of the app where users could post their own thoughts on cryptocurrency trends. Again, this is dependent on completion of the base app.

Experiments

We performed four main experiments for our project: scheduling a task to occur every so many seconds, interacting with a cryptocurrency API, sending email notifications through Phoenix, and mocking up a Javascript- and React-based front-end for the application. Our experiments were fairly successful and helped us hone our ideas, and descriptions of each individual experiment are listed below.

- Polling/scheduling

  This experiment was a test to ensure that we can perform an operation every x seconds. The experiment was successful, and we are using this to update currency values from an API every few seconds to keep users up to date on the prices of currencies they want to see.

- API

  This experiment tested to ensure that we can access the data from Cryptowatch's API and process it. This one was successful as well, and helped to solidify the base on which our cryptocurrency application will function. We are using the Cryptowatch API, which allows us to retrieve price values from multiple exchanges all from one API to give users multiple options when deciding where to get the best value for their money.

- Email

  This experiment was performed to figure out how email can be sent through Elixir/Phoenix. We were able to find success at this using Mailgun (email service) and the corresponding library that is available for use in Phoenix. Using Mailgun ended up being more straightforward than expected, and was a good intro to using SMTP for email-sending.

- Front-end

  The front-end experiment is a two page application using React. The app consists of a landing page and a demo page. The landing page is where the user begins, which we can later use to update the user on their personalized currency feed or to put any helpful information we might want users to see upon visiting the website for the first time or just logging in. The demo page currently updates automatically every 5 seconds with mock data to illustrate what our application could look like when it is actually fully functioning. We learned that we will need an easier way of storing the state of the application, (eg market data, user data) and may need to user Redux.

Project Status

As of this writing, a large portion of the backend API has been written. We have market data being pulled in from the Cryptowatch API every 20 seconds and being processed and upserted into the database. We also have users with user authentication enabled via Guardian. We still need to connect users to markets in a many to many relation, and to allow watching/unwatching markets through the API. Channels will also need to be implemented for receiving live updates whenever new market data is fetched.

The front end React application still needs to be built. We will probably be able to use much of the experiment for the landing page, but we will most likely need to spend a lot of time designing and putting together the React app to handle the data from the back end and implement a smooth user interface.