CS 5610

WEB COIN

a Crypto-currency price tracker

PROJECT
#2

Project Report

Introduction and App Description

As the new wave of cryptocurrencies is engulfing people with enthusiasm, the unpredictability of price fluctuation for their investments is an inevitable concern. Our project web Coin helps the users to overcome this concern, providing them with alert features and visual representation of prices in real-time.

Our application WEB COIN is a cryptocurrency monitoring website dedicated towards monitoring the prices of cryptocurrencies such as Bitcoin, Ethereum, Litecoin and BitcoinCash. The application fetches real-time prices of each cryptocurrency via the Coinbase API and displays the data in the form of line charts using Chart JS library.

The price distribution charts displayed to the user span over hourly, daily, weekly and yearly data. Each of our currencies have their own charts displayed on the main page for visitors of the website to view. Users can register for an account using traditional sign-up/sign-in methods and via their Google and GitHub accounts also.

Users can input as much as fake cash as they want at the time of registration, which they can use to fake buy/sell various cryptocurrencies on our website. A pie chart shows the cash they are left with and percentages of cryptocurrencies they have bought through our fake buy/sell. Registered users are also provided with the facilities to put price alerts for any change in price for their currencies of their choice, and an email is triggered to them if the threshold is exceeded.

User Interaction

The user experience for the application is divided into three main parts: the main page, dashboard, and charts page.

Main Page: The main page displays the home page of our application displaying user charts of current trends of the prices of cryptocurrencies offered by us (Bitcoin, Ethereum, Litecoin and BitcoinCash). The charts display one-day data for the currencies in real-time in the form of a line chart. The users are provided with option to sign in and signup buttons in the navigation bar where they register or signup with us.

Sign-in: The users are provided with two options to sign in to our application. The first method is by the traditional method for signing in; using their registered email id and password. We have provided another optional method to sign in through oAuth2 with no need for the user to remember their passwords. Currently we have provided user an option to Sign In through Google and GitHub accounts using oAuth2.

Register: The users are provided with two options to register: traditional method for registration and through there Google and GitHub accounts using oAuth2.

Dashboard: The dashboard page is divided into four parts.

Coin you own chart: This section displays the user with the cryptocurrencies they own (in our fake buy/sell facility) and their money left in the wallet, in the form of a pie chart which changes dynamically if any transaction is made by the user. For the numbers on the pie chart, we multiple the current price of type of coin and the amount of this type of coins the user own, and show the users how much assets they have.

Buy/Sell Your Coin: This section provides user with the functionality to either buy or sell any cryptocurrencies of their choice on our fake portal. Money is credited/debited from user's fake wallet based on their transaction(s). The user needs to have enough money to buy certain amounts of bitcoins. And once the user bought the bitcoins, the user data will be updated immediately.

Subscribe to Notifications: The section provides user to subscribe to email alerts for the cryptocurrency of their choice. The alert(s) can be put for any threshold above or below for a cryptocurrency value of their desire. The users can also remove the notification(s) if they need to in future through the dashboard. The mail system runs every 30 mins to see if any criterion is met and triggers a mail to the registered users.

General Information: This section displays the alert notifications the user is subscribed to. This section will be updated automatically when the user makes any submissions on the alerts panel and purchase panel.

Charts: The users are displayed with line charts showing the prices of various cryptocurrencies (Bitcoin, Ethereum, Litecoin, BictoinCash). The display contains tab where the user can switch between various currencies of their choice (the tab displaying their current price). Each individual chart has the option to view the historic data for an hour, a day, a week, a month or a year. On hovering over the line chart, the user can also see the values (price) of plot-points (the corresponding time stamp). The data is being fetched through the Coinbase API at an interval of a few seconds and updated in real-time.

The Server Side

Generally, on server side, we store all the information in the elixir mapping, and using Poison encoding API to transfer the data to front end, meanwhile using Channels to broadcast the information of the coin current price and historical price.

Server Using API: In this project, we are using GDAX and Coinbase API for the historical price and current price of four different coin type (Bitcoin, Ethereum, Litcoin and Bitcoin Cash). For the historical price, we implemented the price history basing on the price for the past year, month, week, day and hour. When we implement the API, we used enumeration for the coins types and using the different arguments for different usage of the API. The API file will only export one function and it will have multiple use cases for the other developer to use, we also wrote a detailed instruction for other developers to use the API without knowing what's going on inside the API codes. We used the GenServer to use the function from the API we've set up.

GenServer: We used GenServer to handle the repeating tasks, fetching data from Web Coin API. Since we need to access data from Web Coin API every second to make data display in the front-end is real-time, and we want to check if there any opportunity the price of any coin crosses user's threshold or below user's threshold in five minutes, we use GenServer to do these repeating tasks. For updating price, we broadcast data to channel, then the user who joins in this channel, can get these data. For sending email, any time a cryptocurrency crosses a price threshold or there's an arbitrage opportunity, GenServer will call sending email function to send emails to these users.

Phoenix Channels: Since we need to update the price data on frontend continuously, we need to push the data from backend to frontend, and using ReactJS state to refresh the frontend codes. The application will automatically join the channel once the users logged in. Then, the backend will start to pull data from the API we are using and broadcast the data to the channel and communicate with the frontend.

On the front end the data is being managed by an amalgamation of ReactJS props and redux store, with data being exchanged for a smooth experience.

Database: we are using four tables for storing our data: users, coins, coin_purchase and coin_alert.

> **users:** This table stores the information about the users registered for the application: email, name, money, password.
>
> **coins**: the table contains a single attribute name to store the coins currently being offered on our application.

coin_purchase: the table handles all the transactions the user does in our fake buy/sell application. It's interlinked with the users and the coins table to keep a track of all the transactions made by the users.

coin_alert: As the name suggests, this table keeps a track of all the alerts that are set by the users.

APIs: How & Why?

We are using Coinbase API (https://developers.coinbase.com/) for fetching the data for all the 4 cryptocurrencies in real time, and we also implemented GDAX API for the data for all 4 cryptocurrencies as well (https://docs.gdax.com/) but did not use it for continuous data updating.

Why? The API is the heart of our application and a good, reliable, well-documented and developer friendly API is what we needed for the success of this project. We started implementing the project using GDAX API, but we had a few hiccups in our course of action. The API provided data we needed but had a major concern: it banned the application (Web Coin) for too many requests as we needed data to be updated in real-time to display it to the user. We switched then to Coinbase API which was not only well documented but had an easy to use approach. Though using the GDAX is not very successful, but we still kept the codes for getting the data from GDAX server. We can use it for one-time comparison or one-time price view function.

How? We did research on Coinbase API, there is no package or library for elixir, so we had to use rest API form backend. There's no secret keys needed for the public data, so we just need to use HTTPoison library to pull the data from Coinbase server and transfer the JSON data to elixir mapping data. Our server will continuously poll Coinbase API and when it detects a change in price in any of the currencies we support, it will broadcast this price change via web-sockets to all clients who subscribed for updates for that currency. A GenServer instance runs in background to fetch and update the price information for each cryptocurrency in an interval of 1 second.

The second API we used for sending mail alerts to user was SendGrid API.

How and Why? The API is easy to use & implement, and provided us with free 10000 emails without any payment option. We used the Swoosh library in Elixir to implement it. A GenServer instance runs in background to keep a track if any alert criterion of any user is matched and send them alert mails after a time interval of 30 minutes.

Challenges and Solutions

We faced several challenges in the implementation of this project, which made it a fun learning curve for us.

Automated Login Challenge: The automated login for user through the Google oAuth2 API proved to be a challenge for us. We implemented it using Phoenix libraries using Oauth2 and Ueberauth. The challenge was due to not so descriptive documentation of both the libraries. We implemented them however to work but soon realized that we are working on a single page application layout and giving a redirect through elixir to a token managed website. After our research we stumbled upon react-google-login library for our requirement. It provided us with the same functionality as the other two functionalities but at the same time provided us with an easy way to integrate it within our application.

Alert Mail Challenge: For sending users the alert mails for their requirement was a tricky part. We first implemented it using simple JavaScript and then Nodemailer, but running a job to keep a track was bit cumbersome. We then used Elixir libraries: Nodemailer, Mailgun, Postmark and Swoosh but were unable to implement them properly due to their complicated documentation. The solution? Swoosh! Swoosh is an Elixir library which provides developer with a wide variety of options to use any API: Mailgun, Postmark, SendGrid. We used SendGrid as they provided us with 10000 free emails without any payment information requirement and a well written documentation to back it up.

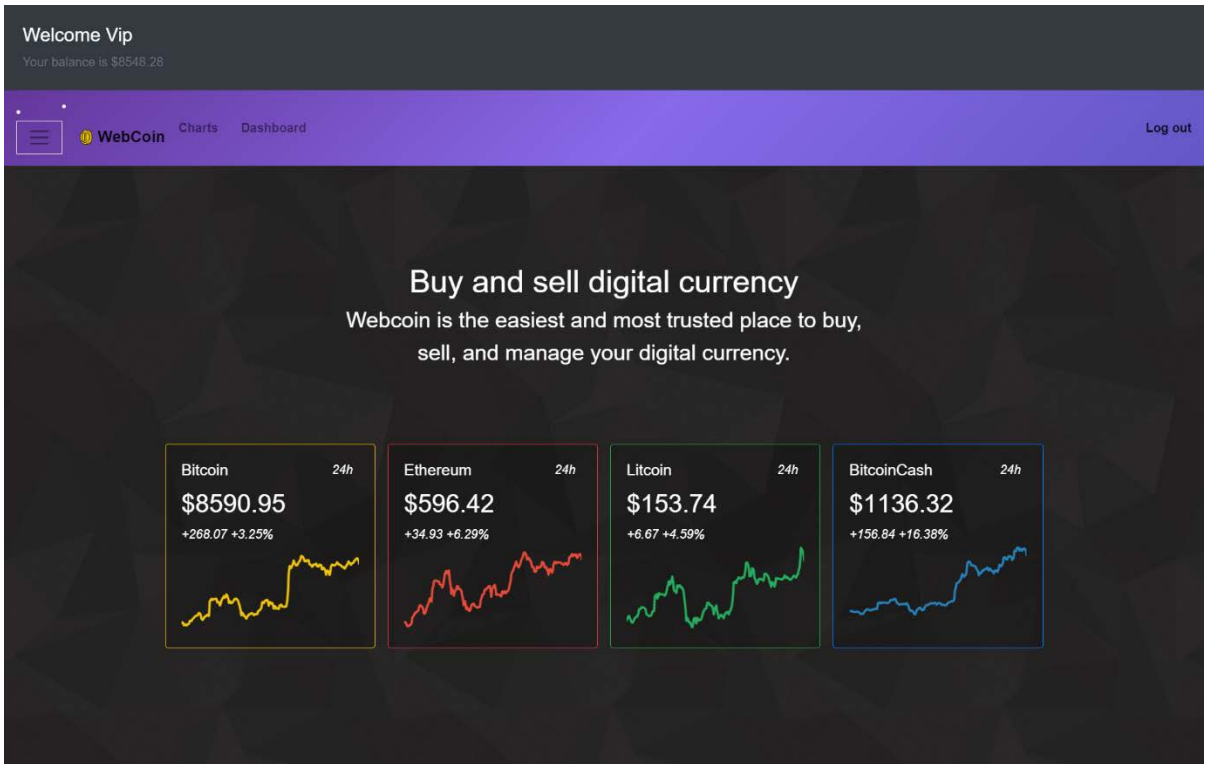
For scheduling the automatic email alerts, we had two options: Quantum Elixir Library and using GenServer. We started the implementation using the Quantum Elixir Library, but due to an old and unmaintained documentation were unable succeed. We also used Nodemailer for sending the application but it was causing conflicts to run in parallel on the same port as out application. The solution to this problem was using another instance of GenServer for keeping a track of all the alert criterions and send a mail every 30 mins. We used a separate instance as one instance was already fetching data from Coinbase API and populating it, running at a second's delay (as we want to provide user with the most updated real-time data).

Crypto API challenge: As mentioned earlier, this API is the heart of our application providing user with real-time values of various cryptocurrencies to help them make that purchase decision. We started out implementation using the GDAX API, but faced a few glitches: it didn't allow us to fetch data too frequently and banned us for too many requests defeating the sole purpose of our application. On researching a bit, we stumbled upon the famous Coinbase API which is not only developer friendly, well documented but also provided us with frequent data fetch to display it to the user.

UI Screenshots

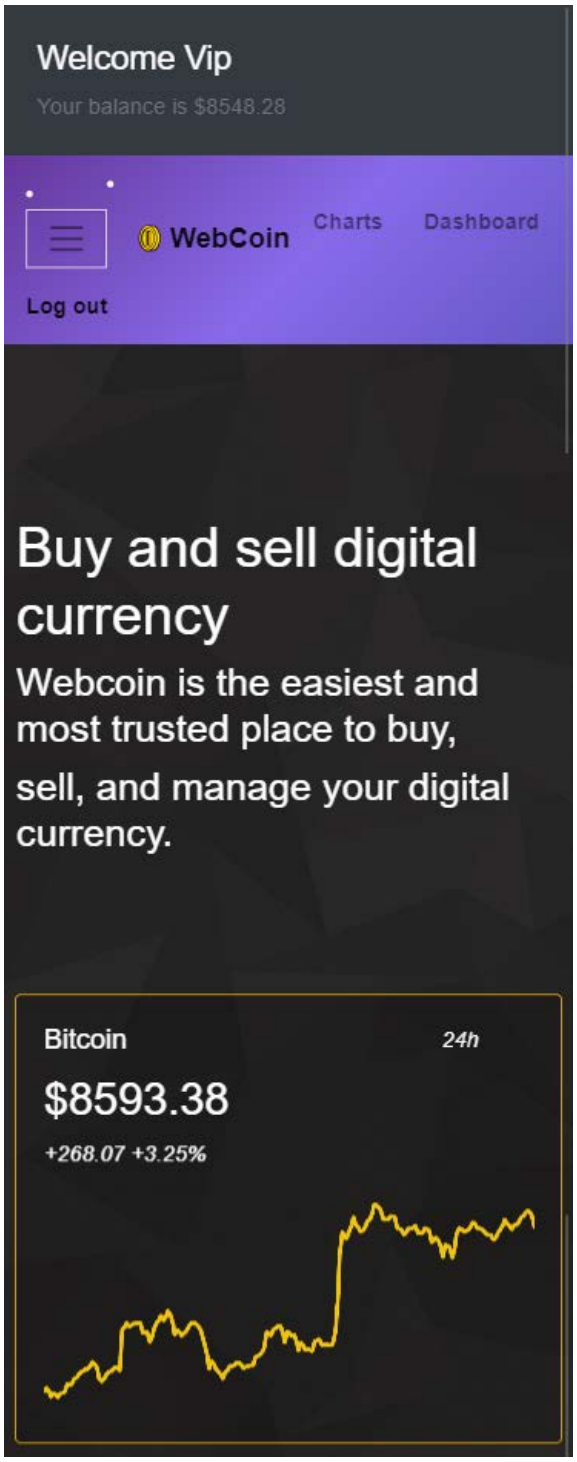
We have used Bootstrap to make sure the application looks beautiful independent of the screen and devices being used by the users.

Home Page (Desktop):



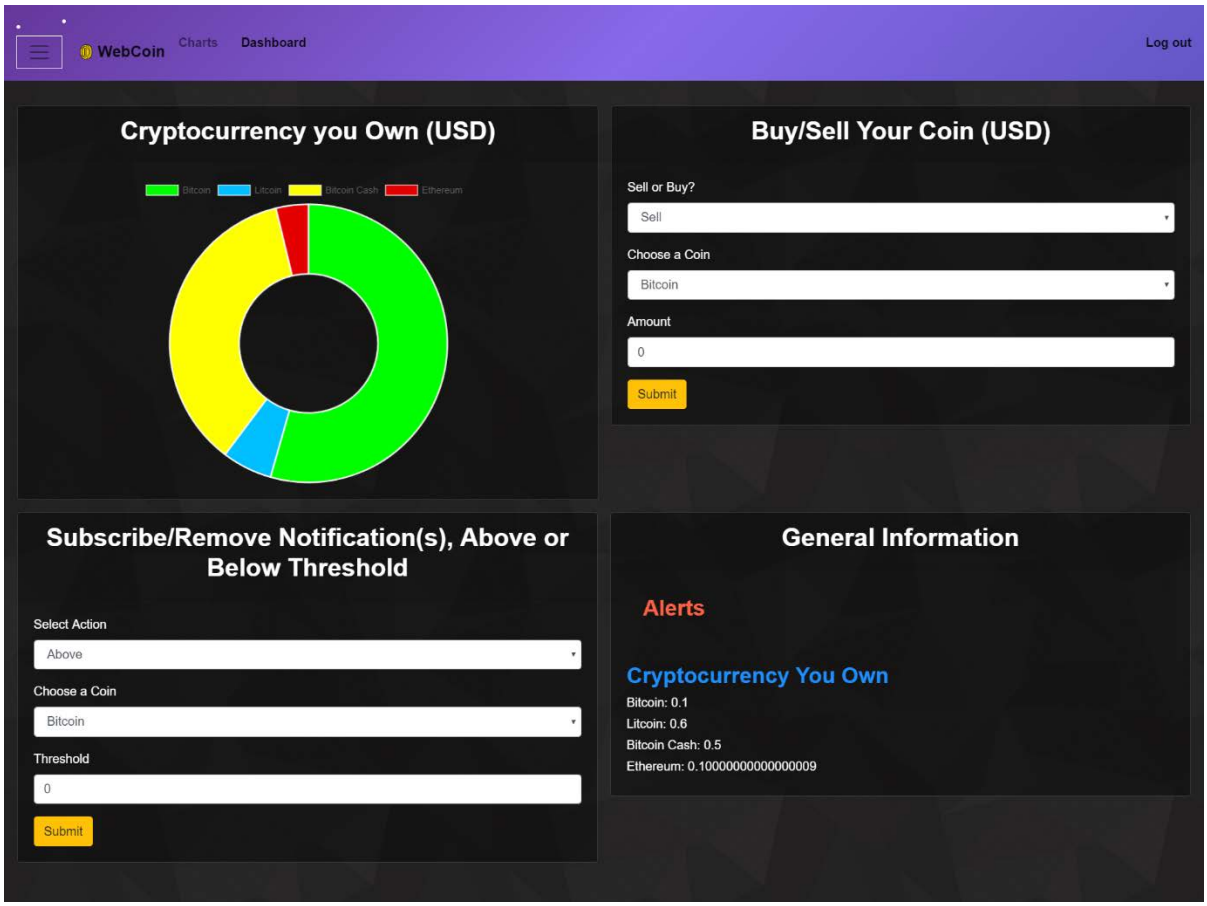
UI Screenshots

Home Page (Mobile):



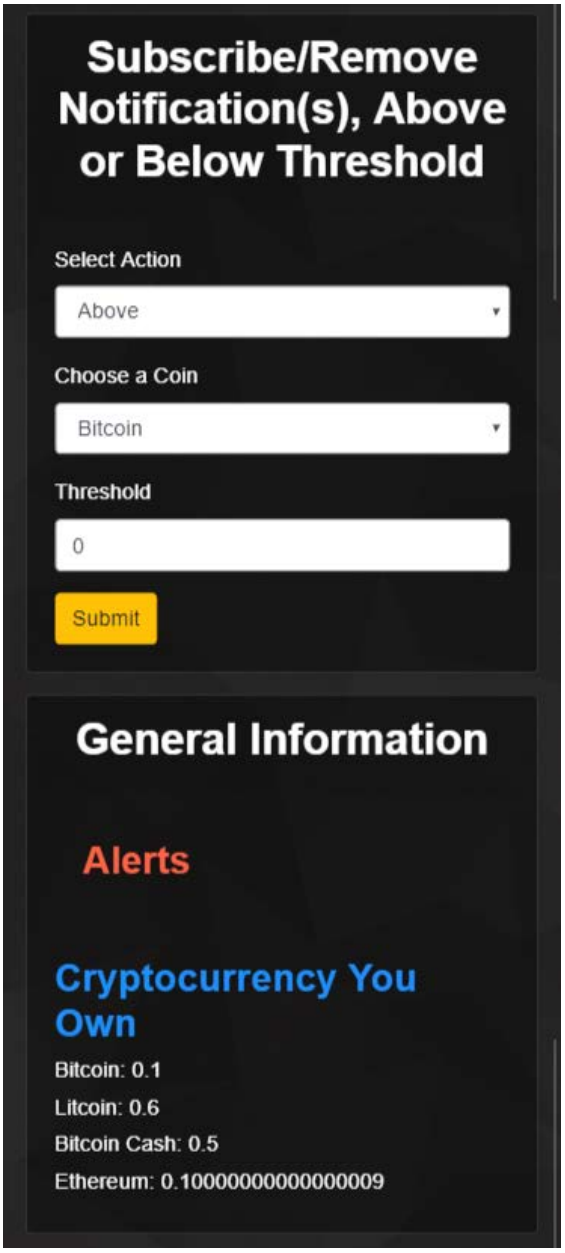
UI Screenshots

Dashboard (Desktop):



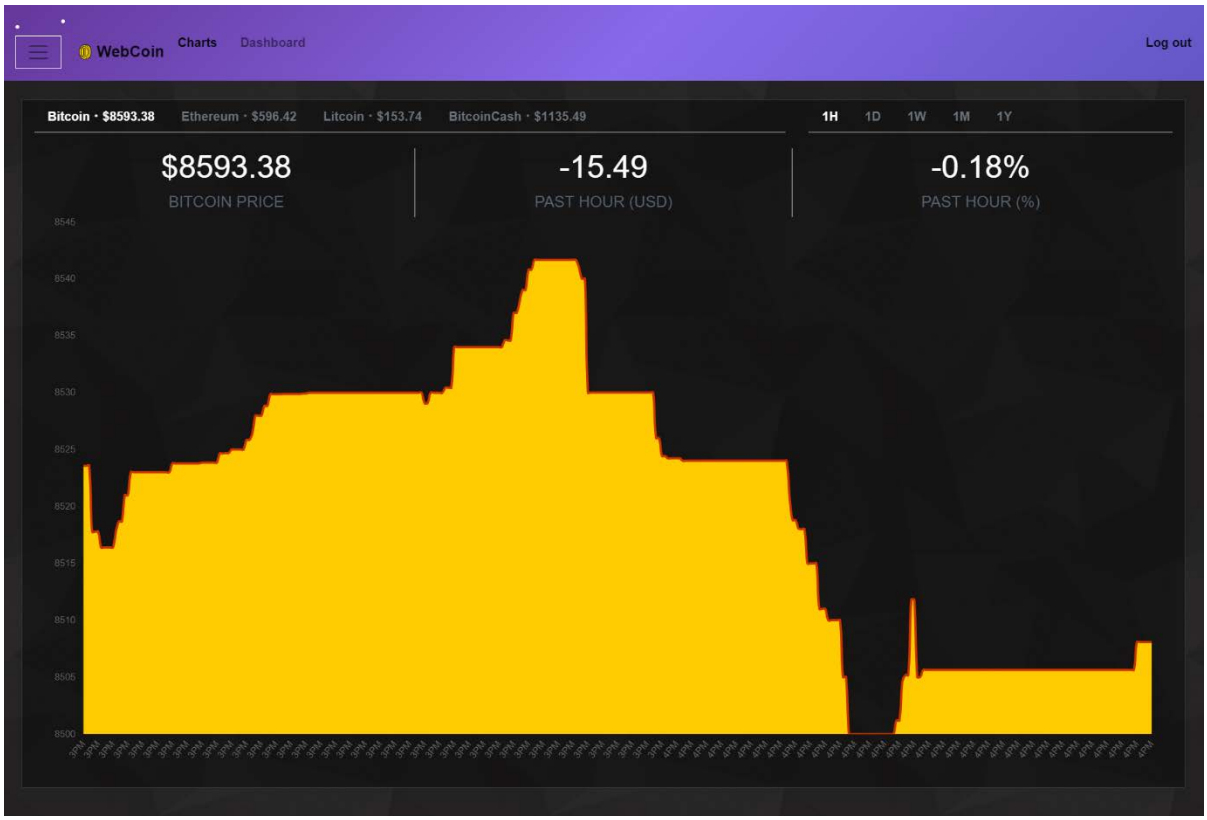
UI Screenshots

Dashboard (Mobile):



UI Screenshots

Charts (Desktop):



UI Screenshots

Charts (Mobile):