CoinFlip

… managing numnistic collections for the discerning investor.

UCSD CSE 134B Spring 2015 :: Team 3

Andrew Wang
Jonathan Ho
Long Tran
Kevin Tran

Contents

I. Overview

CoinFlip is a web-based application that allows its users to build portfolios of widely-traded nuministic investment units. By maintaining detailed information regarding the user's portfolio and a live stream of investment information regarding precious metals, CoinFlip provides users with the data they need to buy and sell coins and bullion bars at opportune times. In particular, we track a live stream of NYMEX and COMEX futures on gold, silver, and platinum to determine their relative strengths against each other, historical data (London spot prices) documenting the long-term trends of gold, silver and platinum prices, as well as a database with comprehensive data regarding the most widely traded coins and bullion bars. We are confident that discerning numismatics collectors will find CoinFlip to be an essential application for maintaining large and diverse portfolios of precious metals.

Fig.1 : Home page displays a live feed of the NYMEX/COMEX exchange futures for gold, silver, and platinum on the left, and the historical prices of gold/silver/platinum on the right.

II. Features

- Login [index.html]:
  - Supports email, Facebook, and Google+ authentication.
  - Provides email/password registration
  - Provides the ability to reset password via email for email accounts
  - Attempts to access any other page without authentication redirects to this page for authentication.
- Home [views/wire2.html]:
  - Live feed from NYMEX/COMEX updates bid/ask/change prices for gold/silver/metal every ten seconds when market is active
  - Provides timer for the NYMEX market's open trading hours from 8:20AM-1:30PM EST (5:20AM-10:30AM PST).
  - Provides 5 day / 1 month / 3 months / 1 year historical data range for London spot prices for each metal.
  - Calculates total portfolio gains for the trading day based on the proportions of asset allocations to each respect metal
- Metals [views/wire3_g/s/p.html]:
  - Has home page's features, but provides data exclusively for the metal
  - Also displays a table of a user's assets belonging to that metal
    - Clicking upon an asset links to item inspection page
  - Selecting '+' allows a user to add an asset via item create page
- Item inspection [views/wire4.html]:
  - Displays image of asset selected
  - Displays a wealth of information tailored to the particular asset and its type
  - Allows the user to delete the asset, update it with more information, or exit back to the metals pages.
- Item create/update [views/update.html]:
  - Two modes:
    - Create - triggered when reaching the page from selected the '+' symbol on a metals page. Allows the user to completely customize the asset.
    - Update - triggered when reaching the page from item inspection. Locks the type, name, metal, and purchase date from editing due to the fundamental nature of these fields to an asset
  - Default options tied to the page accessed from
  - Coin selections drawn from Firebase DB. Selecting a coin loads its data into the fields for the user

- Account management [views/settings.html]:
  - Logout button present at all pages for all users on the upper right corner
  - Users using email authentication will see a Settings button on the left navbar that allows them to change their password and/or login email.
  - Users using FB and G+ will not see this option because it is not relevant

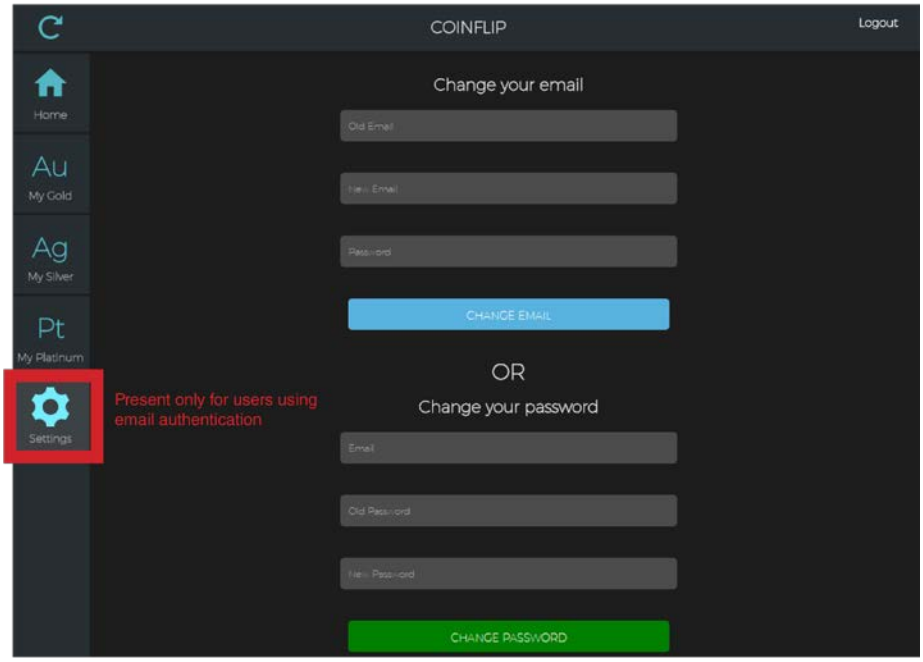


Fig 2: Advanced account management settings only available to users using email authentication

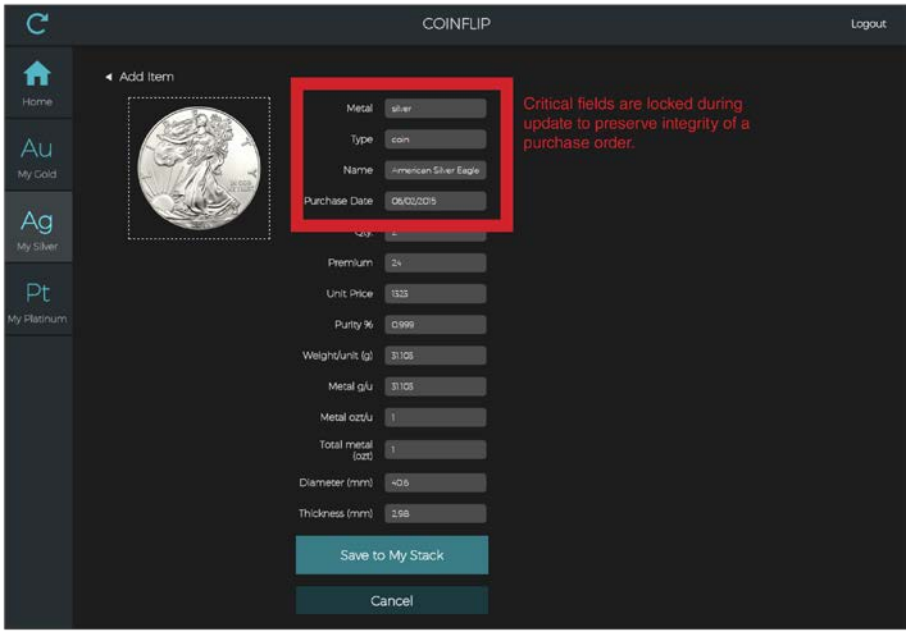
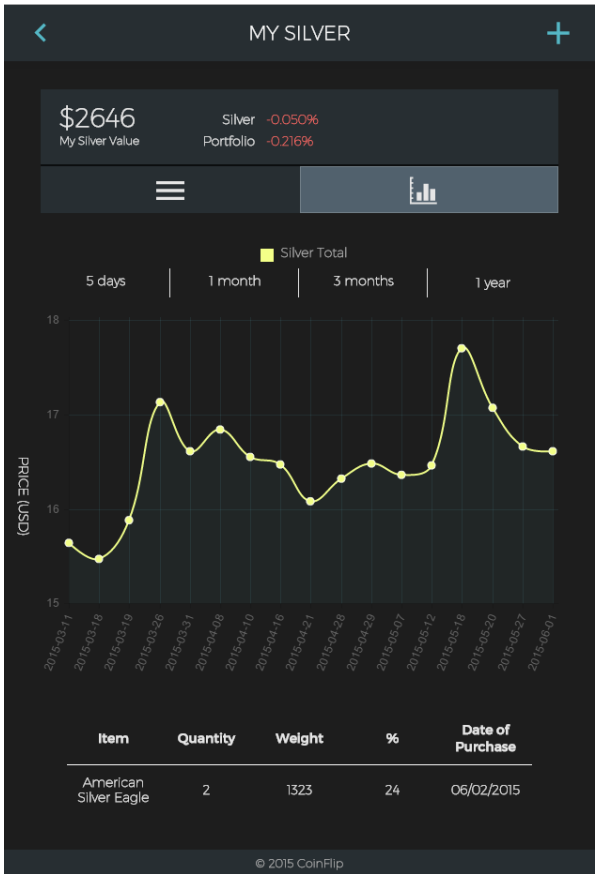
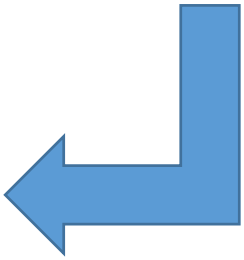


Fig.3: Certain critical fields are locked during update operations on purchase orders





Mobile-friendly

III. Technology

CoinFlip is based on an HTML interface made flexible/data-driven with Javascript scripting, with a back-end running on Firebase's services. External libraries used include JQuery for intuitive DOM manipulation, JQuery UI for its DatePicker, and FireBase's library for interfacing with the FireBase backend. Not including logging/analytics, CoinFlip utilizes no other external libraries, with much of its content being driven by pure HTML/JS/CSS.

A significant amount of the HTML template was written by CSE 134B's Dream Team, although throughout development the template has been heavily modified to suit the needs of our own team. For example, update.html's input forms has been shifted into creation via Javascript (see js/main.js) in order to handle the dynamic nature of the contents of update (e.g. type/name changes require restructuring HTML).

For our backend, we developed the following hierarchy to store our data:

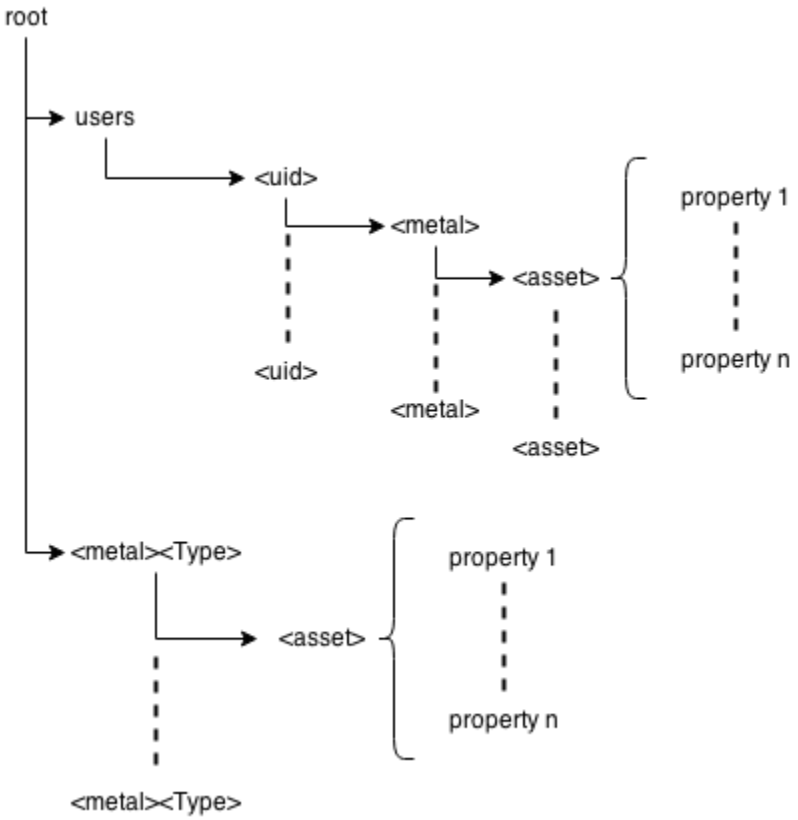


Fig. 4: CoinFlip database schema as stored in CoinFlip

In this hierarchy, we essentially maintain a pool of possible assets under each <metal><Type> (e.g. goldCoins) that get assigned as a child to a user ID upon that user's update or create action. These update and create option place these assets under a child indicating the metal type for easy of access when querying the database for specific metals (common). Although it is arguable that this schema causes redundancies in data storage, given our limited user base and the possibility of adding non-standard coin information, we maintain the position that is better to have this redundancy than not to have it.

The security of CoinFlip is delegated to FireBase's authentication and security ruleset. Specifically, we check to see if Firebase's authentication token (synchronously retrieved via getAuth()) has been issued to the user to determine if the user has access to internal pages. Not only do we check for authentication, we also make sure when the user is performing update, creation, and removal operations on the database that their UID on the authentication token matches the UID defining the database entry their are modifying with the following rule:

```
"users": {
  "$uid": {
    ".write": "auth !== null && auth.uid === $uid",
    ".read": "auth !== null && auth.uid === $uid"
  }
}
```

If the user invokes any security issues, we boot the user to the index login page to authenticate him/herself or obtain authentication credentials.

To obtain authentication credentials, we support Google+ authentication, Facebook authentication, and Firebase-based email/password authentication. For users of email/password authentication, we maintain the options for the user to change their email or their password upon logging in (via the left side bar). We also integrate a "Forgot Password?" link on the front page to send them via email a token password by which they could login to CoinFlip and change their password. These functions can be found in main.js. Login credentials are set to expire after 24 hours.

Mobile support is handled by clever CSS design that checks the screen's size and reloads certain elements in a mobile-friendly form. This mechanism can be found towards the end of style/style.css.

IV. Analytics / Build

Error analytics: as for tracking errors, we decided to use a library called Rollbar, which offers free logging. We wanted to capture all types of errors, therefore did not specify any specific error level. To install Rollbar, we signed up on their website and created a project on their dashboard. Consequently, we were given a unique token for our app, along with a minified Javascript snippet wrapped in a pair of <script> tag. We then placed this in the head section, before all other script tags. The rationale for this is because Rollbar will recursively track errors in all Javascript files the current HTML file links to; it makes sense to capture them all. We repeated this process for all HTML pages in our repository. When an error in the scripts occurs, the Rollbar API method will fire a logging event, sending the type, location, timestamp, etc, to Rollbar's backend. We then could look at the updated Rollbar dashboard to see exactly where the errors occurred.

Action tracking: instead of tracking page views, we wanted to track which actions users frequently use the most to better understand the user flow and performance. We used a service called Mixpanel, which offers a very simple API to track events. We figured that the most valuable actions would be the button clicks. Almost all of our buttons serve a unique purpose, therefore we would like to track each of these events. To install Mixpanel, we again signed up for a free account and received a unique token for our app, along with a minified Javascript snippet that we inserted at the end of our <head> section. This installation allows us to call the simple Mixpanel API by using mixpanel.track("some event"); we targeted actions such as sign up, login, create new items, update new items, delete items, etc. Each of these action are sequences of Javascript functions, so we placed the tracking code upon successful calls. Lastly, to view the events, we look at the dashboard which can sort the events by type, timestamp, and frequency.

Build process is very simple. The build task is built with Gulp and is a one step build process. The developer only has to run ‘gulp build’ to get a distribution folder called dist. In this folder is the app packaged and ready for production. HTML files were left unminified and unconcatenated. CSS and JavaScript file resources were packaged into a single combined.css and combined.js respectively. Using build blocks in HTML we replaced the assets of CSS and JavaScript with combined.css and combined.js so that the developer doesn’t have to fix asset references himself. The packaged app mimics the file structure of the original development structure so files are easy to find. To run the newly created distribution app the developer runs ‘gulp serve-dist’ to ensure

quality of the app locally. Each run of ‘gulp build’ deletes the dist folder so the developer doesn’t have to; only after deletion does the task build the new app.

V. Development Discussion

Onboarding Plan

1. Contact an administrator of the CoinFlip repository at aywang@ucsd.edu for developer access to the repository.
2. Clone the repository by running
   git clone https://<username>@bitbucket.org/cse134bteam3/hw5.git
3. Install nodejs/npm at https://nodejs.org/
4. Install firebase by running npm install -g firebase-tools
5. Read this documentation and the comments detailing the engineering of the product. If any questions arise, feel free to ask CoinFlip developers at any time!
6. If you wish to deploy to the public application at cse134bteam3-hw5.firebaseapp.com, run firebase deploy in the hw5 folder.

Development Recap

This project uses mostly native APIs from HTML/JavaScript. Majority of the CSS was written in SCSS and then compiled to a native CSS file. Our team made use of Dream Team's wireframes by adding to the CSS file directly. To add JavaScript functions we wrote them in main.js and directly in our HTML files. At first glance it is tricky to understand the JavaScript code in main.js mostly because it's in an unfamiliar format. The document.write() function to build partials is difficult to debug because of a lack of editor formatting available. Many escape characters or lack thereof caused hard to track errors.

We built our backend service using Firebase. The data we collected came from COMEX/NYMEX as well as from Quandl APIs. This gave us a wide range of data to share with our users. Live market times counting down to closing time lets the user take his time or hasten his efforts. To implement a login mechanism we used Firebase for user authentication. This was made easy using Firebase APIs and did not require extra schema implementation. We implemented Facebook and Google third-party authorization as well as rolling out a simple email and password authentication system. Using the auth APIs from Firebase we can limit access to our app and redirect users not logged in to the login/register screen.

Redundant tasks were taken care of using gulp tasks. We implemented a gulp task to listen for file changes on our HTML/CSS/JavaScript files so that our browser refreshes automatically. While this may seem trivial at first it saved us time and effort when checking our work. We took advantage of a JavaScript linter to reduce coding errors and several other gulp plugins like minify and uncss to save on file size.

Improvements from HW4

- Test subjects reported that a lack of knowledge of existing coins/bullions presented a formidable challenge towards building a portfolio effectively.
    - We populated a database of frequently traded coins and bullion bars and redesigned our creation/update page to be centered around these coins and bars
    - Overhaul of update page resulted in it now dynamically loading default data into the fields based on the user-selected coin
- Account management system
    - We now support not only Facebook authentication, but Google+ and email/password combinations as well
    - Implemented full account management capabilities - changing email, changing password, and password reset email functions
- Build/Analytics
    - Added Rollbar error tracking and Mixpanel action tracking to gathering information regarding our application's performance
    - Wrote build automation script to compile production release of application
- UI/UX
    - Implemented many fine tweaks to the CSS and layout in response to test subject feedback to improve aesthetics. For example, we moved the optional account management system to the left sidebar and replace it with the Logout button due to users finding the Logout functionality more worthy of the top-right corner real estate.

Testing

Testing was conducted on the latest versions of Chrome, Firefox, and Internet Explorer through distributing CoinFlip to a select group of alpha testers. Having over ten testers actively combing through our application, providing valuable feedback on UI/UX, and catching bugs/loopholes, we were able to produce a well-tested, stable platform for

CoinFlip. We also developed a logging framework to aggregate errors in order to diagnose where users are coming across errors.

What we could have done better

Had we more time to learn React.js many of the reusable components in main.js to load partials could've been wrapped up cleaner in React. Code in React would have been much easier to read and logically follow when creating the UI. Not only would code have been nicer to look at the React engine comes with incredible debug statements in the console that aids in tracking and fixing bugs.

If we knew SCSS syntax then building more complicated CSS may have been possible. Not because SCSS is necessary to build high quality UI but it's easier to read and manage when extending the work of Dream Team's SCSS file. Nonetheless we got by writing CSS only for our styles.

What went well

Our team members were strong users of git so merging our work was effortless. We did not run into common pitfalls like bad code mergers which broke code. We could have used more branches to distinguish the work each individual was implementing but that would have been extra complexity for building a simple web application. Development discussion was made easy using a Facebook group. While it's not Slack, it is a more common platform among college students and was easy to track on our mobile devices as well as work machines.

Future Improvements

- Rewrite document.write() statements into React components or just standard HTML. Partials might be overkill for this simple project.
- Add user asset field to maintain a scarcity-driven portfolio
- Add game mode and invite friends from social circles to compete
- Add theme selector to brighten app
- Add welcome user message to nav
- Replace alert() messages with nicer pop-up/fade modals.
- Wrap project into mobile app using framework Ionic/PhoneGap