Summary

Team Pentapod approached the creation of the website from an outside perspective in order to fully examine and critique it. We broke our analysis down into sections of appearance, functionality, browser compatibility, and implementation to fully capture the strengths and weaknesses of our website. Our group came into this class knowing little to none about web development, and we believe we made great strides in trying to make our website as great as possible.

Appearance

index.html - We wanted to give the home page a simplified yet clean look and feel for the users. We purposely made the signup and login buttons appear similar to buttons found in more popular websites. The only purpose of index.html is to allow the users to signup or login so we made it just that. We unfortunately came across one issue in index.html. When resizing the browser, the login/ signup and Facebook buttons resize with the browser (sometimes overlapping). This issue is a result of the CSS styling of the buttons--using percentages instead of pixels.

Top Navigation Bar - We put the logout feature on the right corner to give the users a sense of familiarity to other websites. The “Logout” text is purposely made large to make it noticeable to the users. The navigation bar also has a refresh button on the left side in case the user wants to refresh their graphs or page. The refresh feature was implemented using javascript. Lastly, the user is able to toggle the theme/ styling of their webpage by clicking on the cog wheel. Currently there are two possible themes for the users--the default dark theme and a facebook-esque theme. The alternate theme was implemented using HTML5 local storage and javascript.

Side Bar - The Side Bar was put into place to allow the user to easily navigate between their most important pages--the home screen and their respective bullion overviews. When the user resizes the window to a certain length, the Side-Bar is minimized.

dashboard.html - The user’s dashboard is ordered and split in a way to display the most relevant information to the user--totals and metal prices on the right; a graph of the metal prices for the last 30 days on the left. The 1/3 to 2/3 split utilizes the real-estate of the page beautifully--a good balance of empty space and content. When the user resizes the window, they are able to choose to display either their totals and metal prices or the graph of the metal prices. When the graph is chosen, the user is unfortunately unable to click the links to their metals’ pages. The slight inconvenience this may cause was sacrificed for a cleaner looking interface. Unfortunately, when the window is resized to a small pixel density, the items on the Top Navigation Bar begin to overlap and collide.

gold(silver/platinum)overview.html - The gold/silver/platinum overview pages closely mimic the style of the dashboard (albeit, with a little bit more empty space). However, in addition to the user's totals and the graph, we added a table on the lower half of the page. The table was created to display the user's respective gold/ silver/ platinum collection. The overview pages exhibit a similar feel and interface as the dashboard. When the window is resized, the user is unable to access the other metal pages, and can only go to the dashboard via the top left arrow in the navigation bar. Additionally, the user is able to add an item to their collection with the "+" button that takes the place of the cog wheel and logout.

updateitem.html - Same appearance as the add item but with the option of deleting the item.

myitem.html - Same appearance as add item but with set information.

Functionality

index.html - We used javascript linked to a Parse database in order to implement the sign up and log in on our page. Users have the option of choosing to create an account with us or use their login credentials from Facebook.

Top Navigation Bar - The refresh button reloads the page for the specific page. The "User name" line will take you back to the main dashboard.html every time. "Logout button" signs you out of the page and automatically send you to index.html. We also created a feature that changes the colors to white by changing css templates when the wheel is clicked by the corner. However, due to the nature of HTML5 Localstorage, switching the pages have a high latency.

Side Bar - The links will take you to the appropriate metal pages. The page you are specifically on will be highlighted as well. This option will be minimized if the page is minimized to a smaller view.

dashboard.html - On the left hand side, there are links to the metal pages when you click on the text. To reset that timer to the correct open or close time, refresh the page using the top dashboard. The graph was provided by Chart.js and data is parsed from a JSON file from Quandl. The Quandl API is modified such that performance is prioritized. In handling the parsing, users are collectively parsing the data such that if they are idle, the data will be updated on our servers sending out new data to everyone. Doing this allows faster server times of our range data to our users and loads the data in the background. The dashboard datas are generated collectively as well and calculated to ensure our users receive the latest details of their assets.

gold(silver/plat)overview.html - The data on the left-hand side is strictly to provide necessary information to the users and give no functionality. The graph is charted using the same methods as the dashboard.html, however, it only shows you the data of the specific metal chosen. Below the chart, there is an option to add a new coin by clicking on the "+" button. Upon loading the page, a query is sent requesting and building a table of the users stack. It was also suggested that this table be locked in and "scrollable", However a team decision decided that users would want to view their entire stack instead of a limited scope of possibly 4 - 5 items in a scrollable window.One feature we didn't incorporate was our search bar, and it currently offers no functionality.

additem.html - You can choose between the three metal types and the metal you choose will be stated in the information below. The type section allows you to choose from 3 types: US Eagle, bullion, and coin. Clicking on the purchase date, brings down a calendar where you can choose the date of purchase. Changing the Quantity, premium, and unit price will automatically change the **TOTAL** number appropriately. You also have the option of saving or canceling the changes at any time. We decided not to upload a picture since users will most often not take the extra time to capture and upload.

updateitem.html - Has identical functionalities of additem.html but with the option of deleting the item as well.

myitem.html - This page shows you the set data that you have created for the specific item. On this page you have the option of going to the updateitem.html by clicking "Edit" or deleting the item from the database and removing the data from Parse.

Themes - Coin Vault has two themes: Dark theme + Light theme. The colors chosen were specifically tailored to complement each other. In the Dark theme, we chose dark colors that, though similar, provide just the right amount of contrast to enable the users to easily navigate through our website. The Light theme was created as a way to invoke a sense of light to those that did not prefer the dark theme.

Browser Compatibility

Chrome:
We mainly developed on Chrome, so this is the basis of comparison for all other browsers. Delete on myitem.html isn't working.

Firefox:
The appearance of the webpage on firefox remained the same from Chrome. On Firefox we had a problem linking to Parse so our add, edit, and delete functionalities for the metals weren't functional.

Safari:
The functionality between Chrome and Safari remained mostly identical throughout. An issue we had in Safari was that for "platinum total" on dashboard.html and platinumoverview.html was not presented in the same line as it was in Chrome.
Another thing we noticed is that Safari renders fonts a little bit differently from all the other browsers, and especially font-size. We used font-size: 0 to bring some divs flush close to

each other but on Safari that doesn't work for some strange reason, so we compensated with some white-space: nowrap so that our mobile toggling selectors didn't overflow.

Internet Explorer 11:
The appearance of the webpage looks identical to Chrome, however, some of the functionality was lost in IE. When attempting to login or sign up, users are shown both the options to log in and sign up instead of just the chosen option which would cause major confusion for the users. When attempting to add item, the command does push the data into parse, but unlike in chrome, it does did not redirect you to the correct metals page.

Opera:
Deleting an item through myitem.html was not functional, but it was working on updateitem.html. Otherwise, Opera had the same functionalities and appearances as Chrome.

Mobile:
Due to time constraints and our limited experience dealing with the mobile applications, we decided to dedicate our focus on optimizing the browser views instead of creating a functional mobile application.

Implementation

User Management - User management for Coin Vault was done primarily through Parse. Parse allows for user management to be simple for both the users and the developers. All of the users' informations are stored in the Coin Vault Parse Database. In all of the pages that required user management--e.g. logging in/ signing up, accessing the dashboard, etc.--the Parse Javascript SDK (JSDK from here on out) is imported and initialized. Those HTML pages were connected to the database via the Parse.initialize() method. Login and Sign Up was only performed in index.html, but extra checks for each page was put into place to prevent non-users from accessing private information or reaching a page only users are allowed access to.

To add some robustness to our user management, our team implemented Facebook Login (i.e. users are able to login and sign up for Coin Vault using their pre-established credentials on Facebook). The implementation of Facebook Login was straightforward due to Parse's and Facebook's guides. In order to add Facebook Login to our user management, our team had to create a Facebook application and have it authenticated. Fortunately, Parse makes connecting the Facebook Login and its own user management an easy process. We simply added our Facebook application's ID and secret key to our parse database and some Javascript code to connect the database to Facebook. The last thing we

had to do was delegate the user management functions to specific elements in the HTML document.

Database - Users have their values stored on Parse and is retrieved when the user logs in. On initial load, we queried parse grabbing the users 30 day history of gold, silver, and platinum and then graphing that data. The table below is generated through javascript based on how many items they have in their stack.

Changing Themes - In an attempt to accommodate a wide variety of audiences, our team decided to implement a small feature that allows the users to toggle between two themes. Implementing two separate themes required the use of two different stylesheets and some javascript. The only differences between the stylesheets, style.css and altstyle.css, are the hex codes of the different classes/ IDs (our team decided that implementing a new interface was too timely). In order to toggle between the themes, our team had to use javascript and HTML local storage. Javascript was used to change the reference of the stylesheet to either style.css or altstyle.css. To ensure the theme would remain consistent when users switched pages, we used local storage to store the stylesheet the user last used. On each page load, the theme is set to the value that was stored. Since our team did not implement the changing themes using cookies, only some browsers are able to support it.

Things that were done often and repeatedly were put into functions in order to minimize reuse of code. This resulted in cleaner code, as well as when changes needed to be made, only one function had to be changed, rather than all of the files individually.

The struggle with most database operations were the things of how javascript behaves, mostly the asynchronous aspect of it. We had trouble mostly with getting Server side and client to be in sync since most of our pages required a second refresh to show actual data. This was however solved by revamping the entire codebase such that everything from the server is loaded once and then cached into HTML5 local storage. Our performance and latency drastically improved after this switch.
Initially, we also had trouble getting data from Quandl such thing were not in sync. To solve this problem, we implemented Daisy Loading that is, when a user logs in and idles on the page, the client will person a parse call and update the current gold,silver,and platinum prices on Parse.

Validation Issue
HTML:
The HTML validation errors that we have are existing ones that came with the Dream Team's wireframes. (I fixed this - Jacob; it should only be warnings now)

CSS:
The errors in CCS validation were broken down into only two problems:

Property fill Doesn't Exist:
CSS3 validator doesn't recognize fill as a valid attribute, but as we tested it with Chrome, Firefox, Safari, and IE, we're willing to forgo solving of this validation flag to implement the fill attribute. After some further research, this seems to be a bug in the validator, much like the one that will be described next.

Calc() parse errors:
Anywhere we used calc() in our CSS, the CSS validator threw a parse error. However, with research, we determined that this was a bug with the CSS validator.
Source: https://www.w3.org/Bugs/Public/show_bug.cgi?id=18913

THOUGHTS/LESSONS LEARNED
We were not able to implement as many features as we had hoped to. We spent a majority of our time fixing problems that actually stemmed from the technologies we decided to use. Using Parse as our database presented numerous problems when feeding data into the graph and displaying it. This problem persisted throughout the weeks and consumed many hours of debugging and thought. After discussing with Professor Powell about the issue, it turns out the problem could have been completely avoided by using Firebase or Meteor which syncs client side and server side. This incident yet again reminds us that making technology decisions too early may be detrimental to a project. We should have researched more into which would be a better option for this specific project, and chose accordingly rather than just choosing which technology we had more experience with. This is, however, the great thing about classes like this. We learn valuable lessons that we can take with us into the real world.

CSS/ Themes:
Themes and CSS styling was one aspect of the project our team overlooked. At first we thought styling would be straightforward--choose a color palette and apply the appropriate hex code--but that proved to be difficult. Color palettes were difficult to choose from because our target audience (older people) is vastly different from us. Several of the color schemes and designs we found were geared heavily towards people our age as most internet users fall under that category. Additionally, we found it difficult to implement the different color palettes to our websites because of the lack of images, content, or elements available on our website. Several of the color combinations we found online and their respective examples had more robust images and more information and pictures to colorize. Moreover, customizing the CSS

(placements and sizing) became increasingly difficult as more elements and tags were introduced to the page and nested within each other.

Javascript:
Although a very powerful language, it is often troublesome based on its asynchronous features. At first this was very confusing as most codebases we have experienced with mostly worked top to bottom with the calls terminating. However in Javascript especially with server side, functions do not finish and we had to implement features such as callbacks or interval settings to keep items in check. After the project however, javascript has plenty of features that can be used to our advantage but upon diving headfirst into the project, we structured our code for failure that required revamping towards the end to achieve our goals.