HW4: Analysis

NOTE: *We're unsure why, but Google Sign-In only works while connected to Firebase and not on mobile. We guess the reason is due to async task. Otherwise, it works fine. In addition, if you search on the coffee hunter page with an empty text box it will last all of the coffee types stored in the database. Besides, our remove function search for the coffee name and delete whole coffee with this name. We assume each user will only add one type of coffee to their favorite list. For the test, you can add duplicates for same coffee, but all of them will be removed once click on remove button.*

For this assignment, working with Javascript was both a blessing and a curse, especially when paired up with firebase's provided database. Most of our group was used to connecting a backend database to a website (for example, MySQL and then letting the backend handle transactions) so learning how to incorporate firebase's database into our Javascript was a learning process. One of the first things we noticed was that without a proper user authentication, our database (and consequently the website itself) would break. Every time we needed to access the database we had to authenticate the user who was logged in in order to allow proper database access. It was something we needed to get used to in order to prevent null errors from stopping Javascript function execution. This was the easy part of the project for us, actually. When we got to the CRUD part of the assignment, another set of new problems cropped up. In addition to learning syntax and figuring out how firebase dealt with queries, we came across issues in how to handle database fields. Some of our html files were either too ambitious in this case (too often we'd go to a new page and confirm a transaction there) or simply incorrect with how our original idea went. Our coffee_hunter.html page has a note field when in reality it was unneeded since the notes field is also meant to be used by users who

have favorited a particular type of coffee. Fixing this error was simple, but it also forced us to think about how our data should be retrieved from the database and displayed.

Our original plan - which we kept following - was to use a database to store various types of coffee that only developers could modify. Users could view the contents of this database, but not actively modify them. They could store its contents and add notes to it, but that would simply add the data to a separate database meant to store that information which would also be retrieved when the user wanted to view their favorite types of coffee. We encountered peculiar errors here as well in addition to trying to nail down a way to retrieve and display data that was not too terrible in its performance. For one thing, there were race conditions in our Javascript. Since firebase takes time to retrieve data from its database our Javascript would execute before the data was ready and cause the webpage to completely break down. The user would see nothing and all we'd see is a bunch of null errors on the console! We tried various things to prevent this from happening (one of which was a really silly sleep function to delay the Javascript execution) but in the end we found that the best performance we could get was simply passing in the information through the URL. It was, admittedly, a hack job and likely insecure but it was an issue we had gotten stuck on. It solved our race condition issue and we were able to load the needed information, however.

As was mentioned, we passed IDs through the URL in order to properly retrieve information from the database. When retrieving information from the database, we realized that a potential issue with the amount of information being retrieved and/or displayed could prop up. At this point in time we were able to avoid the issue, but if the database grew extremely large then we'd probably crash something attempting to display all of the information. As a result, we were debating on retrieving the entire contents of the database once and simply parsing it, but that also led to another problem - would parsing it be quicker than repeatedly retrieving queries

from the database if we were only finding a few queries each time we accessed the database? Unfortunately we weren't able to check this in time due to other responsibilities, but repeatedly accessing the database seemed to load properly on the old Android phone we were using to test our site's performance. We did, however, have to pay special attention to the types of images we were uploading (again) and using since the older Android phone we were testing couldn't handle files that were larger than 1 meg in size - it would unfortunately cause the page to load incorrectly.