HW4 Writeup - Team Foodex

Dylan Uys A12800306, Mauricio Panelo, Chang Gwoun A13193910

The process of adding prototypal user authentication, Firebase, CRUD operations and asset management to our app was by far the biggest step our group has taken as beginner web developers working on Foodex. We encountered many situations where we were required to weigh the tradeoffs of a development decision and go with what seemed to have the most pros and the least cons, usually without being certain that we had made the optimal decision. However, we treated this first run-through as an exploratory effort - trying different approaches, reading up on best practices, and making plenty of mistakes.

Firebase seemed to be the issue with most of our problems. Besides authentication and how to structure our data, it took a lot of time to understand some of the things in Firebase. Google's documentation for Firebase was helpful but there was also a steep learning curve on how to implement a specific method or feature into our own TopicDex app. For example, until the TA posted a demo that showed us how to easily retrieve the user's unique uid from Firebase, with the bind method(), we were wrapping our methods with an observer because Firebase doesn't allow the Auth object to in an intermediate state, like initialization. And so not knowing what we don't know was a huge hurdle in trying to accomplish our goals and task for the assignment.

Thankfully, Firebase makes user authentication pretty straight forward. The biggest limitation on progress in adding authentication was lack of familiarity with JavaScript, which affected the decision process for how to structure the code, where to include the .js files, etc. Additionally, one of our concerns was the flow of the Google login, which takes you back to the login page after selecting your Google account before redirecting to the home page. This issue, though not core to functionality of our app, currently remains unresolved.

We ran into more authentication related issues down the line, such as how to easily identify the currently logged in user so as to associate the correct entries with correct users (which was solved after a TA posted a helpful link on how to accomplish this). This decision was also affected by how we chose to structure our database, and we went through several iterations before deciding. Initially, as our group has SQL experience, we initially wanted to structure the database in a way that would reflect easily queryable data (via SQL queries that we are familiar with). However, we soon saw how different Firebase's database system is from anything we had dealt with before, and we ended up storing users, each of whom owned a list of recipe keys. This is different from our initial attempt of having recipes, each of which had a UID for the user who owns it.

As mentioned above, we went through various approaches of where and how to include our javascript code in our html pages for optimal performance. The first decision pertained to where to include the script tags. Initially, we included them at the bottom of the page so that our page loads wouldn't block. However, with scripts whose contents are required for basic functionality of its respective page (such as firebase/authentication code), this may cause dependency problems. So, the next approach as to include them at the top (in <head>) with the async attribute, again with the intention of not causing blocking. However, this caused a similar issue, as we must ensure that firebase dependencies are included before any firebase authentication code is executed. Thus, we ended up just including the scripts in the <head> without any special asynchronous compensation for page load blocking.

The next javascript inclusion related hurdle that we faced was if/how we should consolidate the code for optimal load times. One group member suggested that we break up the javascript files into chunks of related functionality, placing each chunk in its respective html file in a <script> tag. Another idea was to put all the javascript into one huge minified file.

Optimization was a little more difficult than expected. We definitely wanted to minimize load times to maximize UX but DX was so much better when we had a larger selection. And so we really had to choose what to exclude from the libraries we were downloading. For example, we customized our bootstrap to only include things we needed for it such as the nav bar. This ultimately helped us reduce the time required to write css code and allowed us to spend more time to understand Firebase. We also realized that most of our load time was coming from the images of all the recipes for the page. We read about how some image compression programs could speed up websites but most of them didn't support any other browsers besides chrome and opera. Therefore we decided that we didn't want to let our beloved firefox or edge users to be left out and in the dust by forgoing image compression. We might however resize user image uploads later on so that images sizes will be a little more consistent, in some cases much smaller file sizes, in our TopicDex app.

Additionally, we minified a lot of our files. We initially tried minifying all of our code into one monolithic file in order to take advantage of caching (include it everywhere, first load gets cached for later access). We accomplished this with the firebase authorization code we wrote by guarding blocks code with if statements that check whether the element they want to operate on exists on the current page. This turned out to be too troublesome for the rest of our javascript code (we ran into dependency issues that were too cumbersome to solve), so we ended up minifying individual files for their respective html pages. No caching advantage here, just minimally better load times. Below are some statistics on the saved byte count from our minifying efforts.

	Original Script(s)	Minified Script	Gain
vue-2.2.1.js + vuefire-1.3.1.js → vue-and-vuefire.min.js	9033b	4793b	4240b
app.js + auth.js → app-auth.min.js	4553b	3315b	1238b
main.js → main.min.js	408b	306b	102b
addRecipe.js → addRecipe.min.js	4345b	2107b	2238b
editRecipe.js → editRecipe.min.js	3160b	1744b	1416b
personal.js → personal.min.js	1218b	798b	420b
recipe.js → recipe.min.js	330b	255b	75b

Obviously, some of these files were not worth minifying, as they include such little code. However, we still minified all of our javascript so as to follow good practice and to experience some of the issues that you can run into while trying to optimize load times.