Brandon Huang, Chiraag Prafullchandra, Daniel Ku, David Lee

7 April 2016

CSE 134B

Homework 4 CocktailDex Analysis

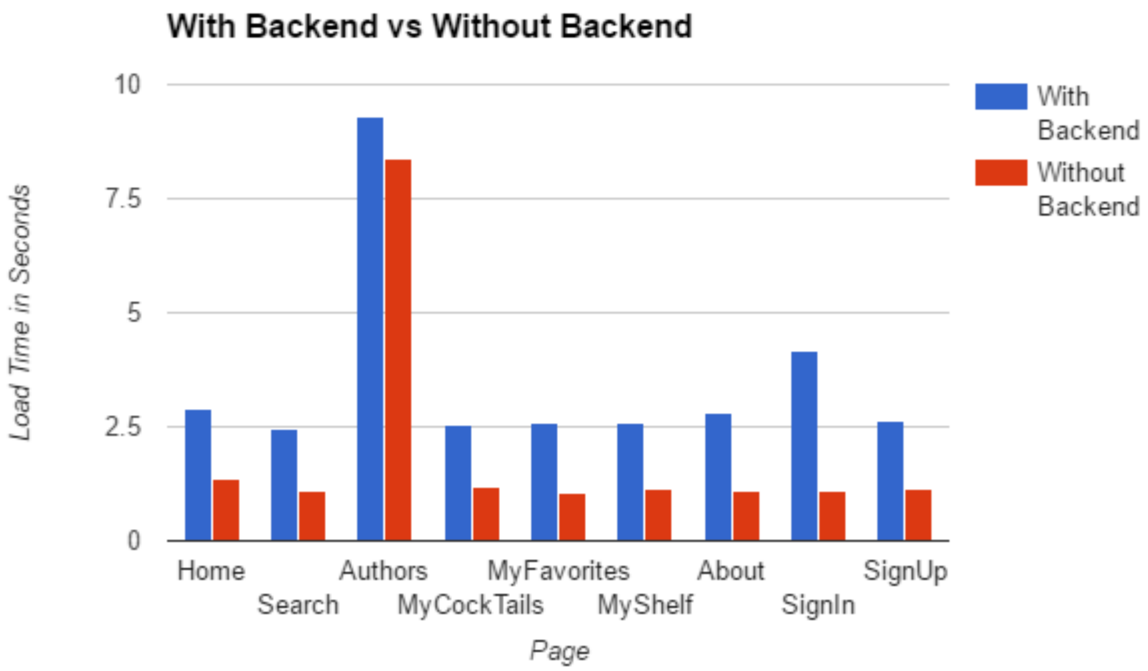
In our web application, we have implemented signup (create), login, create cocktail (create), updating the shelf(update), removing ingredients from shelf (delete), searching (read), and filtering.

As with any code base, we faced some issues determining how best to structure our javascript code, how best to design it with the interests of brevity, re-usability, readability, scalability, portability, and modifiability. Rather than slap everything into one javascript file, which is arguably better as less files need to be fetched, we have temporarily broken our code out into multiple files that reflect the code that is necessary to execute for that specific page. We have done so conscientiously, noting that we do have to fetch more files per page, and that there may be elements of our code that could be refactored to be more reusable, but for now we focus on prototyping and ease of development. To avoid namespace conflict in the future, we choose our variables precisely and try to avoid declaration of global variables.

There were difficulties that our group encountered while executing CRUD operations because implementing certain functionalities were unclear in the firebase documentation . For example, figuring out whether it was possible to have certain nosql entries while manually initializing data. Which didn’t seem possible based off the firebase documentation. Also, while trying to add ingredients to myShelf we encountered a problem where posting ingredients to the database was working but the delete gave us a lot of abnormal bugs such as, adding more of the same ingredient or deleting and re-inserting ingredients we already deleted. The problem was due to using .on(‘value’) rather than .once(‘value). The .on(‘value’) made the value update itself repeatedly which caused multiple entries to appear in the database. Another documentation problem was using .set rather than .update because firebase automatically deletes paths if there are no subvalues under that path. The documentation was also overly simplified in many respects, with only one use case provided in the documentation, leaving it up to the user to extrapolate the best API for a different use case. Simple things such as using the right function calls or figuring out whether a certain aspect of firebase is allowed or possible gave us a lot of problems which in result added more time to our project.

User authentication using email and password was rather simple but time consuming. However google login gave us a lot of problems and in the end we weren’t able to fix it. When clicking the google sign in button a blank page would appear then just crash. There was an auth/network-request-failed error and after surfing the web and trying multiple tests the error was unable to be found. After testing the login in multiple ways we concluded that either our firebase project account was faulty or there was a problem in other files that weren’t affecting the login directly. Since effecting other files may affect the whole web app, we decided to wait till next assignment to continue to debug it.

Our team found that the programming done in this project was significantly more than that of previous assignments. There were multiple concepts and technologies that we needed to combine in order to properly make a dynamic and data-driven web application. Our team was not used to the asynchronous nature of database calls. In order to simply parse the data that we requested, we had to learn about how callback functions worked as well as how to synchronize multiple calls especially when they were called sequentially and were asked to be concurrently processed. Moreover, the team had to learn how to dynamically build HTML/DOM elements using javascript to render dynamic data per user.



In our findings we found that drawing data from our back end significantly increased the time in which it took our pages would load. On average, our pages took nearly twice as long to finish loading as rather than receiving all of the data inside of the html file. Our website would have to also query the real time database for the data required on any given page. This meant that for data intensive pages such as the search result, we were often left waiting for result to come back. The search algorithm in particular drew all of the drink data and parsed it to match any of the search terms.