CSE 134B HW #4

Steve Wang A11371300
Tony Xu A10447459
Zhongting Hu A99056145
Jiangfeng Yin A99078579

Some of the performance challenges we faced is the loading time of displaying our data that we are retrieving from our database, as we create and add more data into the database and reading the data. Along with creating data, we update the favorites page with user's current favorited champions and have a delete/unfavorite function in the profile page. When the user unfavorites the champion, the favorites page will update with the deleted champion removed from the page. Loading the pictures with the file path we store in the database also has a small delay. With many clients sharing a single database, it may also slow down the loading time. However, we don't notice much of a different of the page-loading times. Each time a user creates an account, the database adds the user with a specific key given. When the person is logged in, the user may add and delete favorited champions from the favorites page and the profile page as well. As the user adds or removes champions, the database constantly updates by adding a key with the new favorited champion or removing it from the database. Overall, firebase helps us with the access and syncing of real-time data for each individual and authenticated user. However, adding the Firebase SDK with authentication adds bytes to our application and adding the real-time database adds additional bytes. We included the javascript source for the SDKs in each file of our project. In the end, it turns out that adding these libraries add hundreds of kilobytes to our application.

We notice that the FirebaseUI authentication SDK that we used provided us with many functionalities that maximizes sign-in and sign-up that may take awhile for us to implement. This includes security, error checking, account recovery, and linking each account to an individual ID for easy data access. Our firebase database that we used is accessible both on the server and offline with the SDK persisting on our disk, which makes it easier for us for testing purposes. Offline data stored in the database eventually syncs with our server once we reconnect. Using the Firebase database also allows data syncing within milliseconds, in contrast to using HTTP requests which adds more overhead.

Some development challenges we faced were finding the different functions and methods we had to use to store and retrieve data into the database. There was no set functions given online in the google tutorials to do this and we had to do some research. The relations and structure of object hierarchy was also hard to grasp when doing this. In addition, we struggled with the authentication part, for example, getting this network timeout error when we were storing new user data into the database. After doing some research, we had to make adjustments to our usual html forms and syntax structures to accommodate to the firebase SDK. In regards to performance, there is also a small delay of calling our InitApp() to initialize our application each time we go to a different page which dynamically updates some of the HTML text.

On 3G network, our index page loads on average about 8 seconds. Most of the loading comes from the loading of javascript files from the firebase database and image files. Since the testing was done on 3G, we expected the load times to take this long as we have more javascript and image files for different champions as they initialize and interact with the firebase database. It needed to contact and and grab the correct data needed to display the correct data for each user's set of favorite champions.