BruinYelp Final Report

Nicole Ju

Our team created a review service for UCLA's dining halls: Epicuria, Bruin Plate, and De Neve. The purpose of our app is to provide users information about each dining hall during the current meal period; this allows students to make a more informed decision when choosing where to eat. Furthermore, it provides a platform for students to share their opinions on Hill food to both peers and UCLA staff for potential improvement. Through our app, users can view and write descriptions about menu items, upload images, and provide star ratings. We used Javascript (React) and CSS to create the pages and frontend features of our site, Node to run our app on a local host, and Firebase to store, search through, and retrieve client data.

One feature of our application is Google authentication. If a user is not signed in, clicking on the "Write a Review" button or trying to upvote a review will cause a sign in screen will pop up, prompting the user to sign in to a Google account before they can make any changes. We used a protected route; users cannot access the review form or the user dashboard without signing in. An extra feature we added is providing a dashboard for each user to view all their previous reviews and statistics. Next, we implemented an upvoting feature, where users can interact with other users' reviews. When clicking the upvote button, we update the entry in the database to increment the upvote count for that review. Lastly, we included a sort feature that allows users to change the way they view a dining hall page. They can sort the reviews by most recent, highest rating, and most upvotes. This was done by using Firebase's "orderBy" function to alter the collection that we looped through when displaying the reviews.

I believe I contributed a significant amount to our project. The first area I worked on was allowing users to input data to the backend database; my work allowed for other team members

to later retrieve this information and display it on our pages. I created the “Review” component and gave it various attributes like the number of stars and text description. The function returns a form that asks users to input the menu item, description, image, and star rating. I made a handleChange function that is called whenever the user adds information to the form, updating the Review’s corresponding state field. For the submit button, I set the onClick attribute to a function that writes to the database. This function will add entries (which reflect the Review’s state fields) to Cloud Firestore. Because we needed to display the reviews on both the respective dining hall page and on the user’s dashboard, I wrote to two different collections: that particular dining hall’s collection and a large collection with all the reviews ever submitted. I also included form validation; I had an error screen pop up if a user tried to submit with missing fields (this was similar to Zinnia’s work on the sign in pop up screen, so we collaborated on that).

I experienced a lot of obstacles with uploading to the database, most of which stemmed from timing issues. Initially, upon clicking the submit button for a review, the page would refresh before writing the review to the database. I tried to address this by using a function that prevented the refresh on default, but then I realized that it made more sense to refresh the page after submitting the review so you could see your review on the page. Ultimately, I worked with Andy to turn the function that wrote to the database into an asynchronous function; we used “await” when writing to the database, then used the result of those statements to determine if we should refresh the page. I also had to incorporate Chris’s work on the star ratings to make sure updating the star rating would update the database as well, which took passing props around since the star ratings were a separate component.

Furthermore, I worked on the sort feature. We allowed users to sort the reviews by most recent, highest rating, and number of upvotes. Implementing this was relatively simple; I used

Firebase's "orderBy" function to sort the collection of reviews by a particular field (for example, "orderBy("stars", "desc")" when ordering by the star rating). I used a handleSort function to update the sort option when a user made a sort selection. However, I did experience some difficulties. I was unable to use Firebase's "orderBy" function with its "where" function since I specified different fields for each of these. To provide context, we used the "where" function to find all the reviews for that particular dining hall. Due to the issue, we had to create different collections for each dining hall to eliminate the need for the "where" function entirely.

Lastly, I helped create the dashboard after Zinnia had implemented the Google authentication. This page allows a user to see all of the reviews they've posted and provides statistics like number of reviews and upvotes. Displaying the reviews was similar to displaying it on the dining hall pages, so I worked with Andy (who had done the previous displaying); we just needed to find all the reviews in the collection that were made by that user. An issue we had was making sure we were uploading images and reviews to two different collections (as mentioned earlier), and upvoting updated the review in both of these collections.

If we had more time I would have liked to add in a comment feature, where users can add their own thoughts to other people's reviews. I think this would be a similar process to implementing the write review feature since it is also adding data to the database and later displaying it (there just might be some styling issues involved). Another idea is allowing users to view reviews from previous days or meal periods. Personally, I would like to see reviews from the past in case a dining hall repeats a certain item. Since we already added a folder in the Firebase to store past reviews (for the dashboard), I think it would not have been too difficult to implement. Lastly, I know Yelp has a point system where users can become verified if they make a lot of reviews or receive a lot of upvotes; it would have been interesting to incorporate this into

our project as well. We were able to keep track of a user's number of reviews and upvotes, so we could have possibly implemented such a point system.