Laura Shub
lrs2553

CalorEats

CalorEats is a restaurant focused calorie counting app that stores a database of nearby restaurants, their popular foods, and their calorie counts. The user has the option to either find a certain food directly by querying the database according to either restaurant, calorie count, or price, or they can get choices by clicking the "Generate Suggestions" button. The app maintains a diary of the previous 7 days as well as a bar chart that displays the limit and the calorie total for the past seven days.

The MainActivity of the app does little besides using the FirebaseUI to retrieve the credentials of the user. It then initializes the Firestore, the Storage classes, and the PagerAdapter, which holds the four main fragments of the app: Suggestion, Query, Diary, and Summary. Each fragment is in charge of a specific aspect of the app. The user can switch between them by swiping left and right

The main API that my app uses to generate the foods for the user to select from is SQL databases and queries. There are three tables in total: a 'foods' table that contains the _id field, name of the food, the calorie count, the price, and a restaurant id. The second table, 'restaurants', contains the names and website urls for each restaurant. The third table is 'locations', which contains the address of the restaurant, the phone number, and the corresponding restaurant id. SQL was good for the purposes of this demo, but if this were a legitimate app on the Google Play store, I would probably want to use an API that I can pull data from, rather than a database stored locally.

The results of the query are displayed in a ListView. When the user clicks on it, they are taken to the "New Food" page that shows the name of the food, the restaurant, a link to the restaurant website, as well as the calories and price. An image of the food is also displayed by fetching the corresponding image from Google firebase storage and using the GlideApp module to load it into the corresponding ImageView.

The user is able to view the restaurant locations where this food can be purchased by selecting the "View on Map" button. This takes the corresponding addresses of the restaurant of the food and uses the GeoCoder API to convert them into LatLngs and place them onto a map fragment. Similar to the flipped classroom exercise, if the Geocoder fails, a JSON request is sent and the result is parsed. If more than one location is found, a LatLng bounds object is constructed to include all of the markers and ensure that they are all visible on the graph.

If the user selects to save the food, this saves the food into their folder in the Firestore Database on the current date. The database structure stores a collection of users, labeled by their unique user ID. Inside each user document is some collection of dates, labelled with the corresponding date, and "limit" field that stores the users set calorie limit. Inside the date collections are 'food' documents, which contain the values of the corresponding foods such as name, price, and calories. When the user wants to get the foods that they have eaten by viewing

the diary, the corresponding user and current date is queried and displayed. The user can switch between dates up to one week in the past and view the foods that they ate on that day and the total.

The User's calorie limit is also stored in Firebase, which is retrieved when the views are loaded. It is updated by selecting the 'Settings' window from the top menu on any of the four fragments, and entering the new desired value in the field.

The "Summary" fragment shows the total, obtained the same way as the diary fragment, and the totals for the week. After all of the totals are calculated by obtaining the foods for the past week from firebase, the y-increment of the graph is calculated to ensure that both the limit and the tallest bar are always visible. Then some math is done on the totals to determine the corresponding height and location of each bar. The dates are printed at the bottom.

The feature that makes this unique compared to other calorie counting apps, aside from the greater focus on restaurants and their location, is the ability to get suggested foods. This is achieved by getting the user's location using the google LocationManager, then calculating the distance to the LatLng of each restaurant. The restaurants are then ranked in increasing distance order, and queried for any foods that are lower than the user's remaining calories for the day, sorted by price. The results from each of these are merged into a single MergeCursor, then displayed using the same CursorAdaptor as the query. This allows the user to click on the foods and save them in the same way as a regular query. In order to get the user's current position, the app must request permission from the user. This took me a very long time to get to work because I kept requesting the coordinates from the Network, which of course does not exist since I'm running on an emulator, so it would just hang forever no matter what location I tried from. This was easily fixed when I realized that I could get the location from the emulator GPS, which made my previous hour of trying to get the location manager to work extremely pointless, but at least it works now, I think. If I were to redo this part of the project, I might try using Google's distance matrix API to calculate an approximate time to reach each restaurant, but I had issues with my API key that I couldn't figure out how to resolve and I wanted to ensure that I completed all aspects of the project.

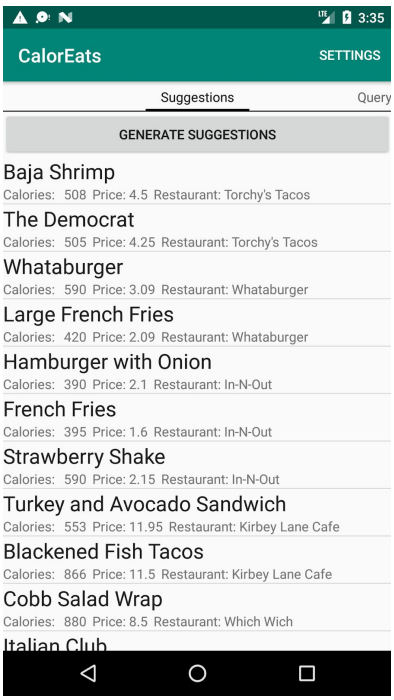
The most significant error that I encountered was dealing with the asynchronous task of getting information from Firebase, especially for the graph. This was solved by passing the Firestore the widget that I wanted to change, such as the Graph or the EditText, and then updating it when the task finishes. Another way that I solved this was passing in a reference to the current fragment, and then calling a method in the fragment on task completion. There might have been a better way to approach this, but this worked for my purposes.

Another significant problem was the SQL database I was using becoming corrupted and unusable when I hadn't pushed in a while, so I had to remake many of the entries. This reminded me of why we use version control, and why pushing to GitHub regularly is important.

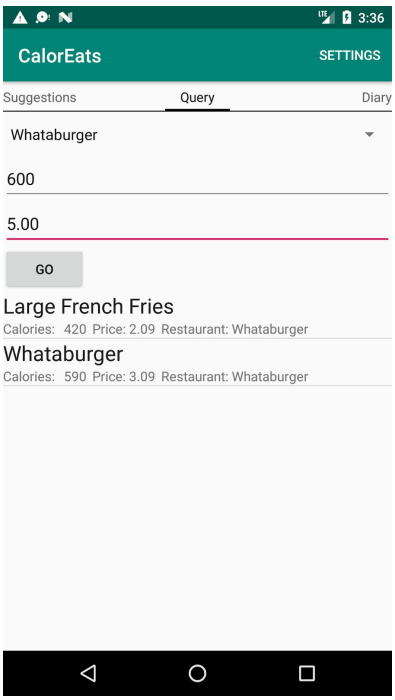
In order to run the project, the account that I have been using to test everything is "shublaura@gmail.com", password "123456".

Screenshots:

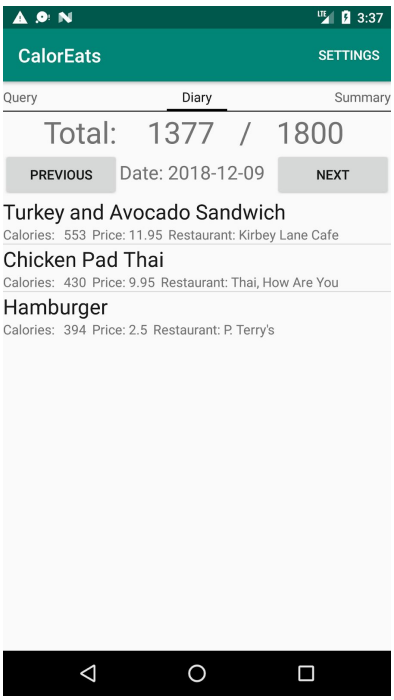
Suggestions



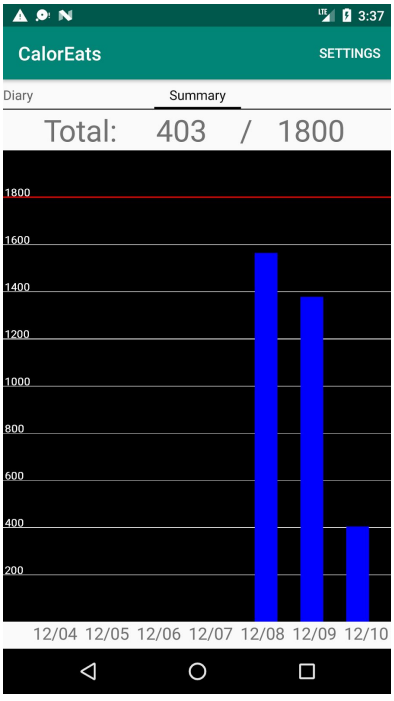
Query



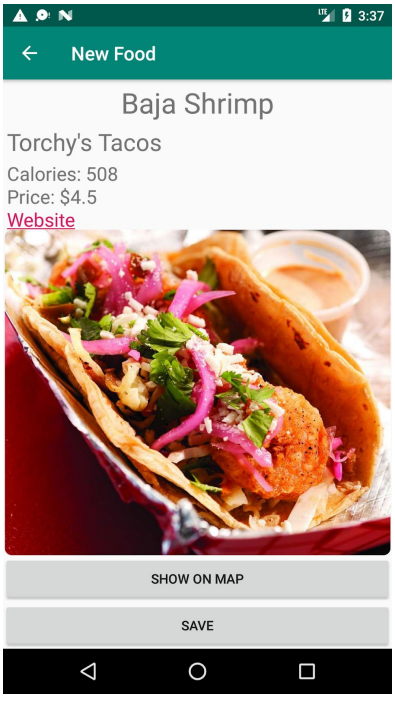
Diary



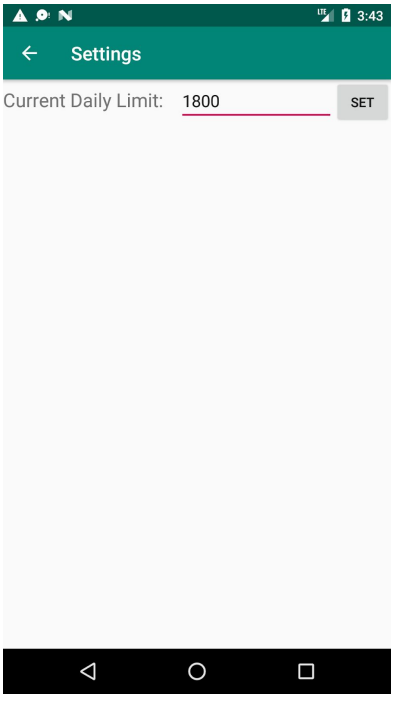
Summary



New Food



Settings



Code Totals:

From cloc

Language	files	blank	comment	code
Java	18	462	81	1570
XML	19	84	16	796
SUM:	37	546	97	2366

However, I did not write DatabaseHelper.java, Storage.java, FirestoreGlideModule.java, ic_cloud_download_black_24dp.xml, styles.xml, colors.xml, google_maps_api.xml, ic_launcher_round.xml, ic_launcher_background.xml, and ic_launcher_foreground.xml, so subtracting all of these gives:

Language	files	code
Java	15	1443
XML	12	560
SUM:	27	2003

The fill cloc result is in “line_totals.txt” in the main directory.

GitHist