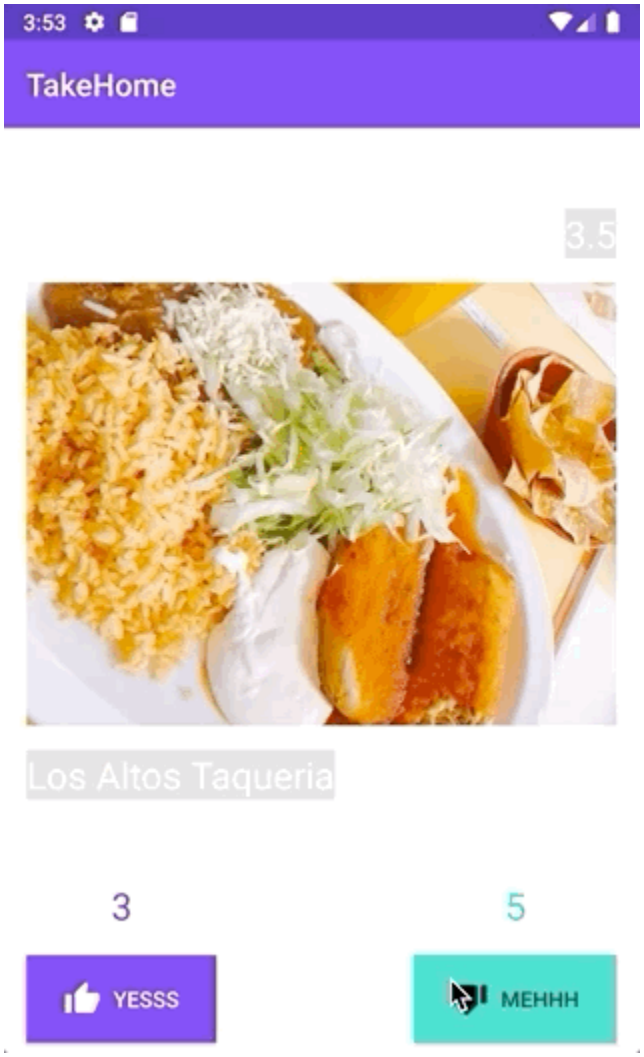
Affirm Coding Exercise

This is an exercise to test you on reading the existing codebase and adding some logic to it.

The app

The complete app is an app that loads the restaurants from two APIs (Yelp and Places). The user can tap "YESSS" or "MEHHH" to indicate if they like or dislike the restaurants. The counters on top of the buttons will be updated accordingly.



Gif is located in the “documentation” directory.

The challenge

The challenge should take 90 ~ 120 minutes to complete.

We have provided some skeleton code to you and you can work on top of it or modify the existing code as you wish.

Requirements

Complete the app

Write the logic to query some restaurants from two APIs. The data model is already implemented.

Yelp API

Places API

We load a maximum of 20 restaurants at a time. When the user approaches the last couple restaurants, we want the app to load more restaurants.

The way we load from the two resources should be alternatively. E.g. load the first batch from Yelp, second batch from Places, then Yelp, then Places, so on and so forth.

The two APIs fetch results in a slightly different way. Yelp fetches additional results using an "offset" integer value. The Places API will provide additional results (if available) via a "nextPageToken" that will be included with the search results.

Properly dealing with two different APIs is part of the challenge.

Suggested steps

1) Load the restaurants from the Yelp API first.

2) Implement the endless scroll (when the user almost reaches to the end of the image stack, load the next batch).

3) Load the restaurants from Places API, and load the restaurants in alternating fashion.

There are some "TODOs" to guide you on where you would need to implement code.

Complete a writeup

Write a simple text file about how long you spent on completing this exercise and what architecture/libraries you chose to complete the app. Also include where you think the app can improve.

Criteria

We will evaluate programs on correctness and code quality.

For correctness, we expect that the restaurants will be loaded and displayed correctly. After the first batch of the restaurants, the next batch of the restaurants will come from a different source. Once the user clicks the "Yes" or "No" button, the counter also gets updated.

For quality, we expect code to be easy to read and maintain, and have separation of concerns. We will be grading based on the architectural decisions that you made. (You can write about it briefly in the writeup)

You will NOT be graded on tests.

FAQs

Q: The skeleton code is in Kotlin, can I use Java?

A: Yes! If you aren't very familiar with Kotlin, it's totally fine! Kotlin code can be easily called by Java and vice versa. You can create your part of the code in Java.

Q: Code provided imports for "FusedLocationProviderClient", can I use something else? A: Sure!

Q: I see the skeleton code has room for improvement, can I improve it? A: Yes, we'd like to hear your thoughts!