

Design Report

for

TastyTable Restaurant Recommending App

Version 1.0

Prepared by Justin Pacella

12/17/2023

Contents

1 Introduction.....	2
1.1 Purpose.....	2
1.2 References.....	2
1.3 Document Overview.....	2
1.4 Summary & Conclusions	2
2 Design	2
2.1 Overview.....	2
2.2 Research.....	4
2.2.1 Mobile Development Framework	4
2.2.2 Restaurant Search Service.....	4
2.2.3 Geolocation Service	5
2.2.4 Access Token Security	5
2.3 Verification.....	5
2.4 Implementation	7
2.5 Testing.....	11
3 Logistics	Error! Bookmark not defined.
3.1 Project Timeline.....	Error! Bookmark not defined.

1 Introduction

1.1 Purpose

The purpose of this document is to detail the design process for my mobile app, tentatively named TastyTable, for recommending local restaurants. Also, since research took up much of my time, I included sections detailing my findings and conclusions.

1.2 References

- Yelp Fusion API Documentation
 - <https://docs.developer.yelp.com/docs>
- Yelp Display Requirements
 - https://terms.yelp.com/developers/display_requirements/
- Abstract IP Geolocation API Documentation
 - <https://docs.abstractapi.com/ip-geolocation>
- Flutter UI Framework Documentation
 - <https://docs.flutter.dev/ui>
- Dart DotEnv Package Documentation
 - <https://pub.dev/packages/dotenv>

1.3 Document Overview

The introduction section of this document intends to give the reader an overview of the document. It contains the purpose of the document and each subsection, references, and personal conclusions I drew from my design process. The design contains an overview of my design and its functions. It also contains information regarding the start-to-finish design process, including a section pertaining to the research I did before I could begin my design. The logistics section contains the overall timeline for my design.

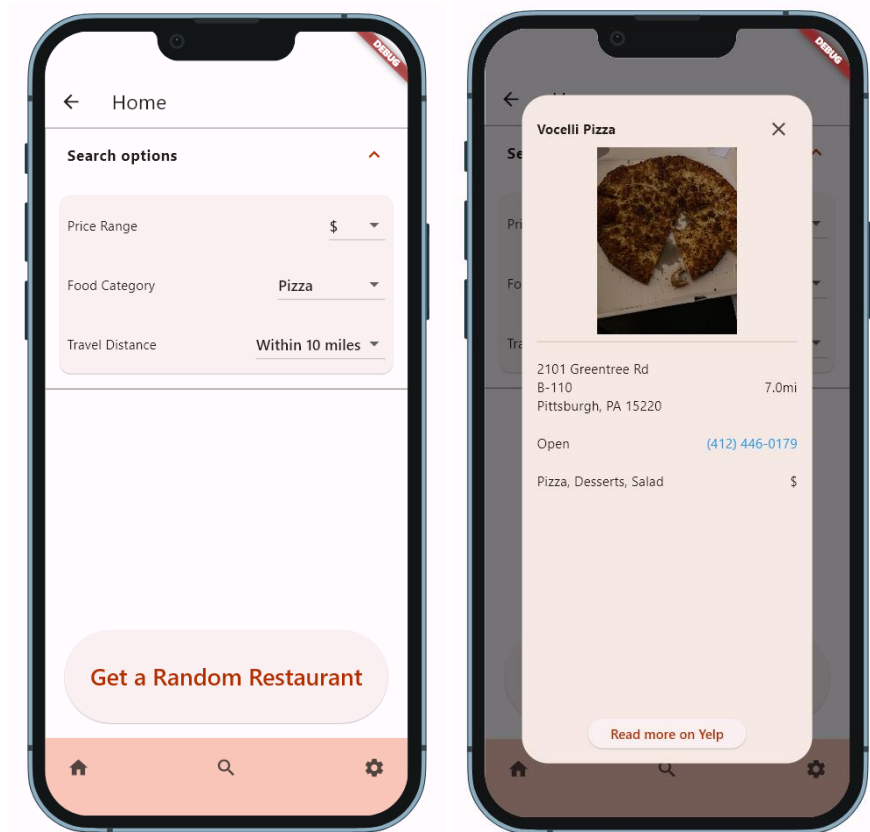
1.4 Summary & Conclusions

My main takeaway from this project is that mobile development is a very involved process that uses object-oriented programming almost exclusively. Realistically, I spent most of my time researching various aspects for my design.

2 Design

2.1 Overview

My restaurant app's primary feature is to recommend restaurants to the user by picking a random nearby restaurant based on user-provided criteria. The user may select a maximum price range, maximum travel distance, and food category using dropdown menus on the **Home Page**. When they click the **Get a Random Restaurant** button, the **Business Page** appears showing relevant information about the business.



Figures 2.1 & 2.2: Final Revision Home Page and Business Page

While the randomization button is the primary feature of the application, I also included a few other features to enhance the user experience. The **Search Page** allows the user to manually search Yelp's database for restaurants by name, category, or locale. The user may tap on one of the results to view the same business page that appears on the home page. The **Settings Page** allows the user to toggle between automatic and manual location. If the user enables manual location, they may enter an address in the corresponding text field. The settings page also includes a developer options tab so the developer may test certain aspects of the software. This tab would be removed in the final release.

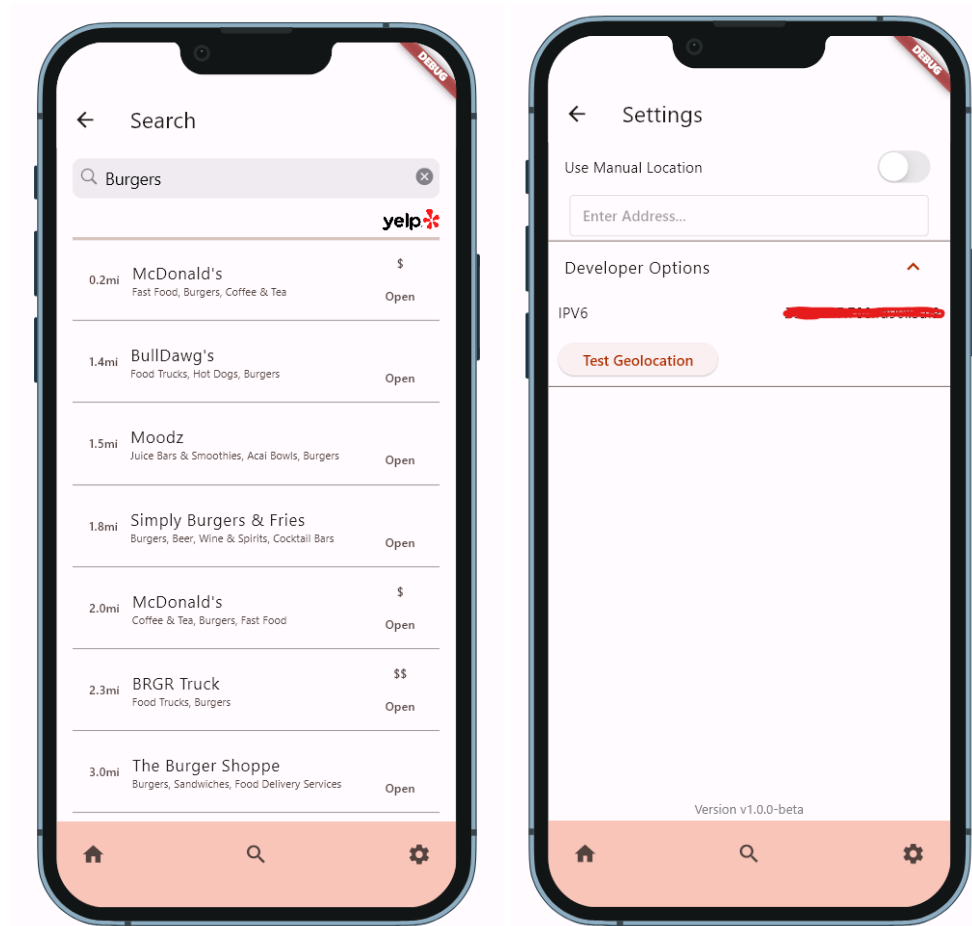


Figure 2.3 & 2.4: Final Revision Search Page and Settings Page

2.2 Research

2.2.1 Mobile Development Framework

My research for this design began with learning the ins and outs of mobile development. Since my personal device is an iPhone, I started by looking at Apple's Swift programming language. However, I quickly realized that without a system running macOS, I would be unable to properly verify my software using this approach. Instead, I investigated using a more versatile framework so I could develop and verify my software in a Windows environment. I settled on using Google's open-source Flutter¹ framework since it offers cross-platform support and a means of verifying the software via the Device Preview plugin. This framework employs the Dart programming language, which is a compiled object-oriented language. I found Dart easy to learn due to its similarities to C++ and Java. After this was done, I was able to create my Flutter project directory and begin the development process.

2.2.2 Restaurant Search Service

After I verified that the Flutter framework was adequate to build my design, I began looking into the backend implementation for searching a restaurant database. The first (and final) option I tried was using the Yelp Fusion API.² Using API was almost completely painless; all I had to do was make an

¹ See Flutter UI Framework Documentation in [1.2 References](#)

² See Yelp Fusion API Documentation in [1.2 References](#)

account, generate an API access token, and use Python requests to search the database. The research I did for that point on was learning how to properly utilize the search parameters offered by Yelp’s database and Yelp’s attribution requirements³.

2.2.3 Geolocation Service

Setting up a script to experiment with geolocation was a different story. I began by trying the Google Maps API, but quickly found out that it was a paid service. I didn’t want to pay for geolocation because I knew that free services existed for it. My next step was Amazon Web Services (AWS), which offers a free geolocation service. I spent a lot of time learning the basics of the AWS user interface and management console. However, the information I found on the AWS geolocation service used AWS JavaScript exclusively. Unsure of whether this was viable option in the Dart language, I continued my search. Eventually, I settled on the Abstract API⁴ due to its similarity to Yelp’s Fusion API. However, experimenting with this API was more difficult than it was with Yelp. After continuously receiving the wrong location from the service, I discovered that providing the device IPV6 address to the search parameters fixed the issue.

2.2.4 Access Token Security

Due to the sensitive nature of API access keys, I had to research a way to properly hide keys. Luckily, dart had built-in package, DotEnv⁵, that suited my needs. The process was more involved than I expected but was relatively easy to implement.

2.3 Verification

Like my Bop-It project, I verified my design by component. First, I used Flutter to create a navigation scheme for my software. All I implemented for this were page titles and the navigation scheme using Flutter’s Navigator object. I also employed the Device Preview plugin to see what these pages would look like on an iPhone.



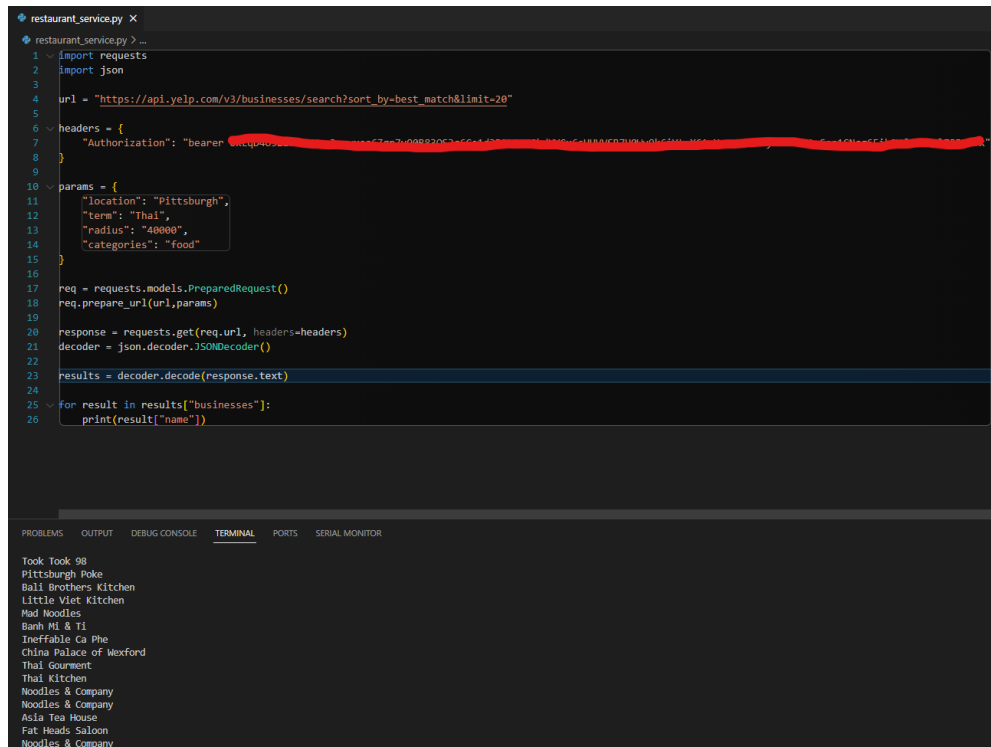
Figures 2.5, 2.6, & 2.7: Preliminary Navigation Scheme and UI

³ See Yelp Display Requirements in [1.2 References](#)

⁴ See Abstract API Documentation in [1.2 References](#)

⁵ See Dart DotEnv Package Documentation in [1.2 References](#)

Next, I verified the functionality of the Yelp Fusion API search function using python. I used a simple script constructs a http request, attaches the search parameters and API key, and prints the names of the resulting restaurants. I also needed to use the *json* module to decode the http response.



The image shows a code editor window titled 'restaurant_service.py' with a Python script. The script imports 'requests' and 'json', sets a URL to the Yelp Fusion API, defines headers with an API key (redacted), and parameters for location, term, radius, and categories. It then makes a GET request, decodes the JSON response, and prints the names of the first 20 businesses. Below the code editor is a terminal window showing the output of the script, which lists 20 restaurant names.

```

restaurant_service.py X
restaurant_service.py > ...
1 import requests
2 import json
3
4 url = "https://api.yelp.com/v3/businesses/search?sort_by=best_match&limit=20"
5
6 headers = {
7     "Authorization": "bearer [REDACTED]"
8 }
9
10 params = {
11     "location": "Pittsburgh",
12     "term": "Thai",
13     "radius": "40000",
14     "categories": "food"
15 }
16
17 req = requests.models.PreparedRequest()
18 req.prepare_url(url, params)
19
20 response = requests.get(req.url, headers=headers)
21 decoder = json.decoder.JSONDecoder()
22
23 results = decoder.decode(response.text)
24
25 for result in results["businesses"]:
26     print(result["name"])

```

PROBLEMS OUTPUT DEBUG CONSOLE **TERMINAL** PORTS SERIAL MONITOR

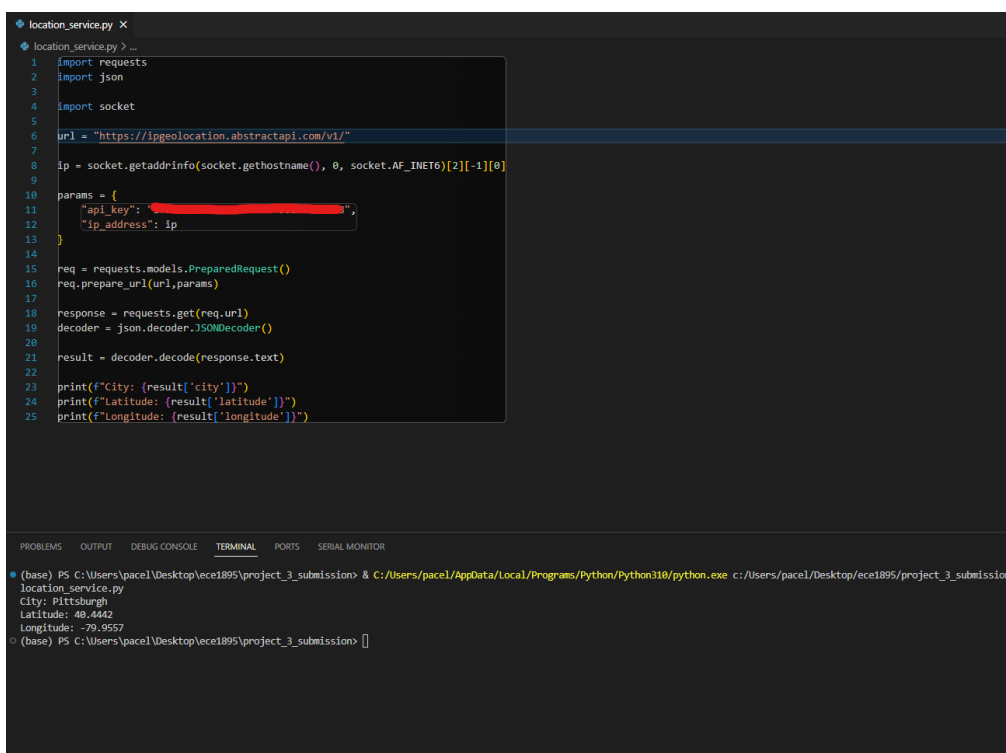
```

Took Took 98
Pittsburgh Poke
Bali Brothers Kitchen
Little Viet Kitchen
Mud Noodles
Banh Mi & Ti
Ineffable Ca Phe
China Palace of Moxford
Thai Gourment
Thai Kitchen
Noodles & Company
Noodles & Company
Noodles & Company
Asia Tea House
Fat Heads Saloon
Noodles & Company

```

Figure 2.8: Restaurant Service Verification Script & Results

Last, I verified the functionality of the Abstract API geolocation service in a similar manner. The only differences were that Abstract needed the API key to be passed as a parameter instead of a header and that I needed to use the *socket* module to determine my IPv6 address.



```

location_service.py X
location_service.py > ...
1 import requests
2 import json
3
4 import socket
5
6 url = "https://ipgeolocation.abstractapi.com/v1/"
7
8 ip = socket.getaddrinfo(socket.gethostname(), 0, socket.AF_INET6)[2][-1][0]
9
10 params = {
11     "api_key": "XXXXXXXXXXXXXXXXXXXX",
12     "ip_address": ip
13 }
14
15 req = requests.models.PreparedRequest()
16 req.prepare_url(url, params)
17
18 response = requests.get(req.url)
19 decoder = json.decoder.JSONDecoder()
20
21 result = decoder.decode(response.text)
22
23 print(f"City: {result['city']}")
24 print(f"Latitude: {result['latitude']}")
25 print(f"Longitude: {result['longitude']}")

```

```

(base) PS C:\Users\pace1\Desktop\vece1895\project_3_submission> & C:/Users/pace1/AppData/Local/Programs/Python/Python310/python.exe c:/Users/pace1/Desktop/vece1895/project_3_submission/location_service.py
City: Pittsburgh
Latitude: 40.4442
Longitude: -79.9557
(base) PS C:\Users\pace1\Desktop\vece1895\project_3_submission>

```

Figure 2.9: Geolocation Service Verification Script & Results

2.4 Implementation

I began implementing my design by building off my Flutter verification prototype. During this time, I heavily utilized Flutter's documentation page since I was unfamiliar with the language and framework.

The first part I built was the beginning of the UI. I added a folder to my library containing object definitions for each page of my UI, consisting of the home, search, settings, and business pages. For the home page, I implemented the **Search Options** dropdown by making a backend model stub class. The dropdown contained:

- Price Range – Implemented as a combo box. The options are “\$”, “\$\$”, etc. When a higher price is selected, the results will still include lower prices.
- Food Category – Implemented as a combo box. The options are Pizza, Breakfast, Thai, etc.
- Dietary Restrictions – Opens a separate dialog menu with checkboxes for keto, kosher, vegan, etc.
- Travel Distance – Implemented as a combo box. The options are within a set number of miles with a maximum of 25 miles.

During the backend implementation, I realized that Yelp's search function offered no dietary restrictions parameters, so I omitted this option from the UI. Instead, the Food Category box contains a few dietary restrictions options, but the results' accuracies are limited.

Next, I implemented the settings page. Originally, it just had an option for using manual location along with a text field. I later added the developer options tab so I could test the backend portions that

weren't directly accessible from the frontend. This consists of the geolocation testing button and IPv6 display.

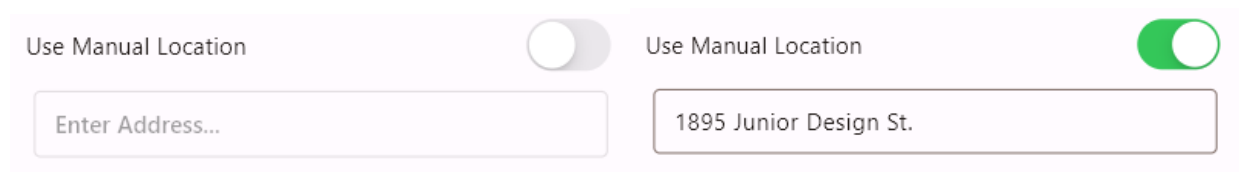


Figure 2.10: Manual Location Setting States

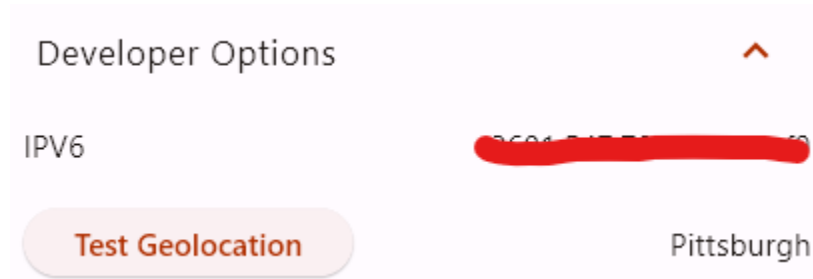


Figure 2.11: Developer Options Dropdown

After I implemented these pages, I conducted my research and verification of the Restaurant Search function. I then wrote a backend class in Dart to manage the search function. This class required me to create a `.env` file to properly hide my API key. The responsibilities of this class are detailed in the CRC Card (Table 2.1) below.

RestaurantService	
Attributes	
Name	Description
<code>_searchUrl</code>	Stores Yelp Fusion API base search URL
<code>resultCount</code>	Number of results from last search
<code>results</code>	List containing relevant data for each result
<code>_token</code>	Contains API access key
<code>_headers</code>	Contains headers for URL request
<code>response</code>	HTTP Response object
Responsibilities	
Responsibility	Collaborators
Load Access Token	App Initialization Call
Search	Search page frontend, Home page frontend

Table 2.1: RestaurantService CRC Card

Only after I implemented and verified this component could I build the search page. I implemented the search bar using Flutter's CupertinoSearchBar⁶ widget and linked it to the RestaurantService using the `onSubmitted` function. This means that the results will only update when the user taps "Enter" on the virtual keyboard. To satisfy Yelp's Attribution requirements, I added a tappable Yelp logo that launches the devices browser and opens Yelp's website.

⁶ The widgets that start with "Cupertino" have the same appearance as Apple's widgets

When the user conducts a search, the results list appears below the bar. Each item shows some of the information about the business, including the name, categories, distance, price, and whether the business is currently open.

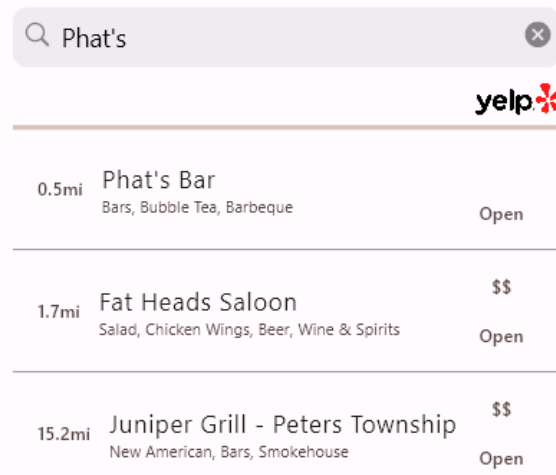


Figure 2.12: Search Bar, Yelp Attribution, and Search Result Items

Once I completed this part, I moved on to geolocation. After I finished researching and verifying, I implemented a second backend service class to handle the geolocation requests. Like before, I added the API key to my `.env` file. The CRC Card (Table 2.2) summarizes the purpose and contents of this object.

LocationService	
Attributes	
Name	Description
<code>_serviceUrl</code>	Stores Abstract API base URL
<code>_token</code>	Contains API access key
<code>responseBody</code>	Location data from HTTP response
<code>empty</code>	Stores whether the response body is empty
<code>latitude</code>	Response body latitude value
<code>longitude</code>	Response body longitude
<code>city</code>	Response body city name
Responsibilities	
Responsibility	Collaborators
Load Access Token	App Initialization Call
Update Location	Frontend
Get IPv6	Self, Frontend developer options

Table 2.2: LocationService CRC Card

After implementing the location service object, I compiled a list of places where the object is called. It is called by the search function, developer options button, and upon initializing the app. The next part I implemented was the business page for the UI. It is shown when the user taps one of the search results or taps the random restaurant button. This page includes comprehensive information about the restaurant selected by the user or randomization algorithm. This consists of:

- Business name
- Website image

- Business address
- Distance (in miles)
- Whether the business is open/closed
- Phone number
- Business categories
- Price range
- Link to business's Yelp page for further reading

The displayed phone number is tappable and launches a URL to open the device's phone app and call the restaurant. The **Read more on Yelp** button launches the restaurants yelp webpage so the user can read more information about the establishment.

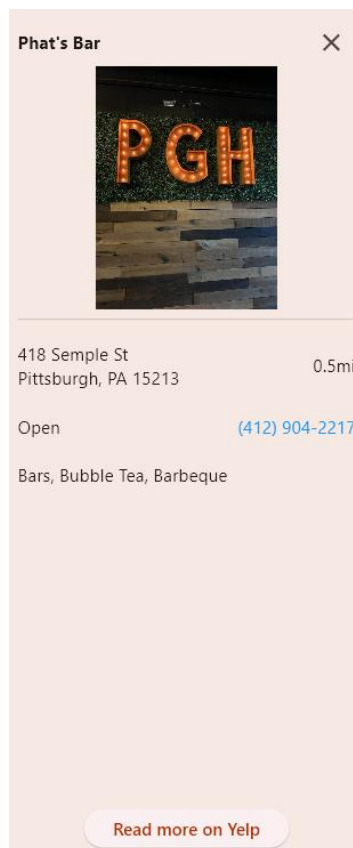


Figure 2.13: Business Page Example

This part of the UI was the hardest to create. Properly scaling the image and implementing text-wrap for the address and categories was challenging.

The last feature I implemented was the random restaurant button. Since this is the main feature of the app, I enlarged the button and placed it at the bottom of the home screen. When the user taps this button, the application re-calls the restaurant service search function with the selected parameters. Then, it selects a random integer within the length of the results and opens the business page corresponding to the restaurant.

Get a Random Restaurant

Figure 2.14: Random Restaurant Button

2.5 Testing

Unfortunately, I was unable to port my app to my Apple device. The process necessitated the use of Apple hardware, which I do not have. When I tried the computer lab in Benedum hall, the computers' versions of Xcode were out of date and therefore completely unusable. Tech support took too long to get back to me about updating the software. I also tried using AWS EC2, but I was unable to change my service quota in time. Despite this, I was able to test my application using the device preview plugin and test most of the app's functionality. The only portion I was unable to test was the phone number URL.

To test the randomization button part of software, I put myself in the shoes of the user and used the app to recommend a restaurant. For the parameters, I entered medium (\$\$) price range, Italian food, and within 10 miles. The app recommended Buca di Beppo Italian Restaurant. I drove to the restaurant and got takeout lasagna. It was yummy. This, along with a few other trials in which I didn't travel to the business, convinced me that the main feature of my app functions as intended.



Figure 2.14: Test Results for Randomization Button

I tested the search screen by entering several terms into the field and tapping enter. I noticed a few issues in the results. For instance, when I search by category, the results often don't contain the category I searched for. This is due to a mismatch in Yelp API call query parameters: when the search term is entered by the user, the API is called with the search term string as the "term" argument instead of the "categories" argument. Without a way to dynamically check if the user input is a category or a general

search term, there was no way to differentiate between the two and I was unable to rectify this issue. Since the responses from the Yelp API vary, this issue is difficult to reproduce.

I repeated the above tests with the manual location setting enabled. I tried using San Diego, CA and Munich, Germany as my locations. The results were similar to what I tested before, however I was in both cases unable to validate whether restaurant exists or not. I was only able to check that restaurant exists on Yelp.

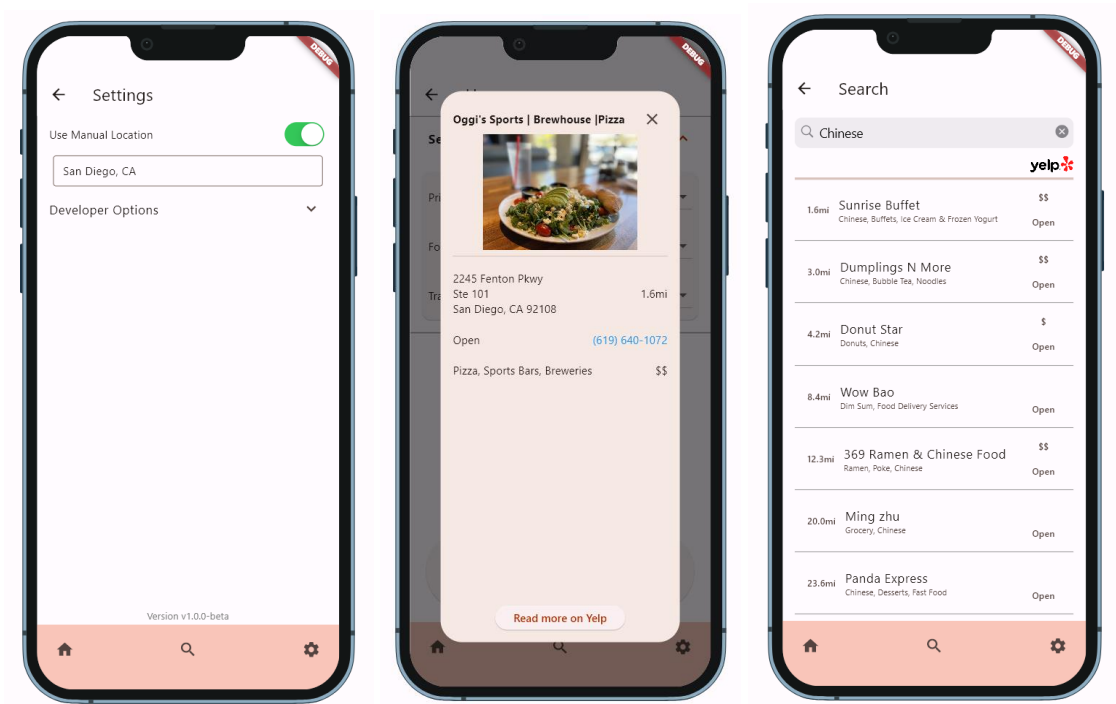


Figure 2.15: Test Results from San Diego, CA

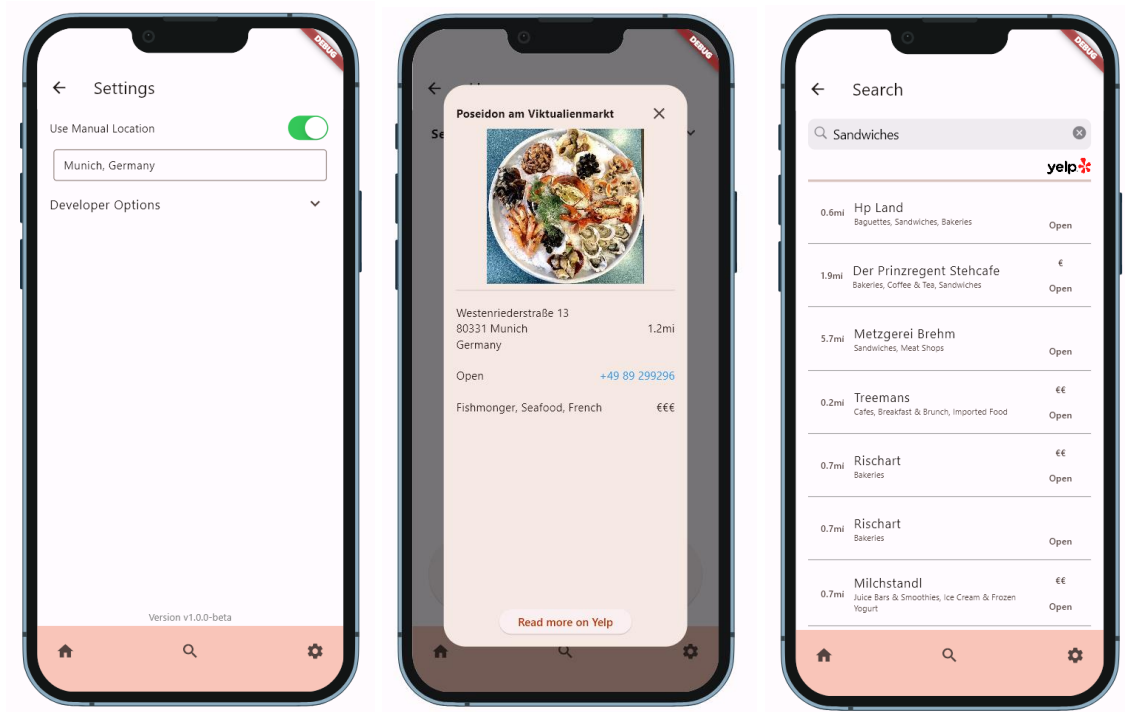


Figure 2.16: Test Results from Munich, Germany