CS 5412 Project 1:
Restaurant Finder

Michael Wang (mzw4)
February 10, 2015

1 Overview

This program provides an interface to find nearby restaurants within a certain distance of a given location. The input location can either be in the form of an address, as latitude and longitude coordinates. Examples of valid location inputs include:

1 Ferry building, San Francisco, CA 94111
20.3423, −80.5234
406 College Ave, Ithaca, NY 14850
33324

The user must also provide a distance threshold in miles.

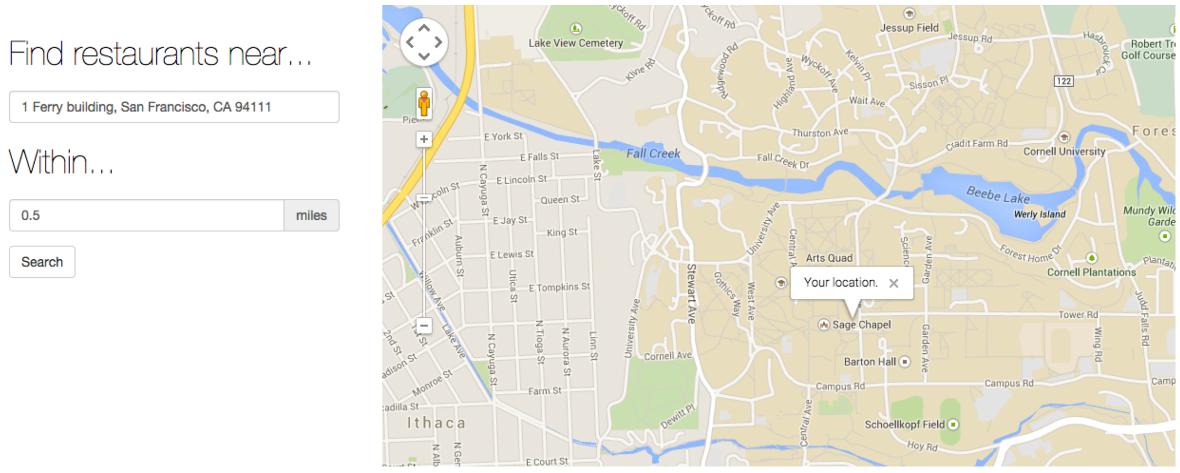
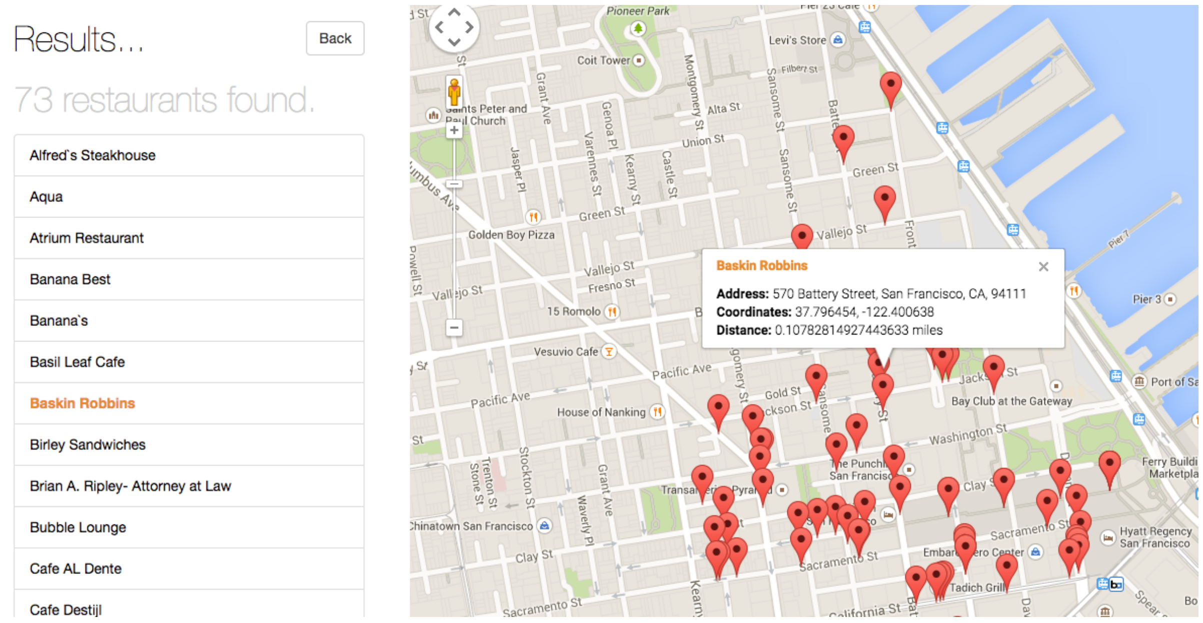

Find restaurants near...
1 Ferry building, San Francisco, CA 94111
Within...
0.5 miles
Search

Figure 1: Finder Interface

The program will compute distances for all restaurants within the same zip code region from the input location, and display the results as a list. The restaurants will also be displayed on a map, where each restaurant is represented by a push pin. The user may click on a restaurant in the list or on a push pin to view additional details such as the address, latitude and longitude coordinates, and distance from the input location.



Results...

Back

73 restaurants found.

- Alfred's Steakhouse
- Aqua
- Atrium Restaurant
- Banana Best
- Banana's
- Basil Leaf Cafe
- Baskin Robbins
- Birley Sandwiches
- Brian A. Ripley- Attorney at Law
- Bubble Lounge
- Cafe AL Dente
- Cafe Destijl

Baskin Robbins

Address: 570 Battery Street, San Francisco, CA, 94111
Coordinates: 37.796454, -122.400638
Distance: 0.10782814927443633 miles

Figure 2: Example results.

Additionally, the user may download a file "queries.txt" containing a formatted output as specified in the project description.To do this, click the "Download formatted results" button at the bottom of the results page. One thing to note about the formatted output is that each line starts with the name of the restaurant plus its address, rather than just the address. This makes it a bit more informative and readable, and the instructions didn't specify whether or not the name could be part of the address.

2 Design

2.1 Program Architecture

The program is structured as a web app running **Flask**, on a Python web framework. When the user makes submits a request, the client forms an ajax call to the server, which handles the bulk of the computations. To perform geocoding operations, the program uses the **MapQuest geocoding API**. This API can perform batch geocoding requests of up to 100 locations To display results on a map on the frontend, the program uses the **Google Maps API**.

The server provides ajax endpoints for the client to call. It essentially serves as an ajax wrapper for the `finder` Python module which provides the interface for the following core restaurant-finding functions:

`def connect_db():` Downloads the database given at the url "http://www.cs.cornell.edu/Courses/CS5412/2015sp/_cuonly/restaurants_all.csv" and stores it in server memory. This is performed only once when the server is started so the user doesn't have to wait when they open the app.

`def find_nearest(input_addr, distance_threshold):` Performs the database lookup and distance calculations to find the nearest restaurants within `distance_threshold` miles. This function does the following:

i. Geocode the input address to get its latitude and longitude coordinates and its zip code (or if it is already given in coordinates it reverse geocodes it).

ii. Search the database for entries matching the zip code of the input location.

iii. Batch geocodes matching restaurant entries and computes distances from input location. If the distance is within the threshold, it adds the restaurant to a list. Also, repeated database entries are eliminated, since the program uses a dict that maps address to distance. To be considered a duplicate the restaurant must have the same name and address. This provides a more refined output.

iv. Formats and returns the matching restaurant list with additional data such as coordinates and distance values.

Additionally the server saves the formatted results of each query to a document "queries.txt". The client may make a call to download this data, which calls the function `def ajax_get_results()`. This file is truncated each time the server is restarted. To save a file from the client, the program uses the library **FileSaver.js**.

2.2 Design Decisions

Designing the app with this web app structure provides a good separation of logic between the server and client. The app server simply makes calls to the `finder.py` module and relays the respone data to the client which presents the results in a user friendly way. In fact, the module `finder.py` provides all of the core operation needed for the app, and can be operated by a separate command line interface by running the script `cmd_test.py`. Serving the program as a web app also makes it easy to host and distribute.

The decision to use the MapQuest geocoding API revolved around its ability to make batch calls of 100 locations at a time. This drastically reduced the number of requests per query, improving runtime and avoiding the problem of exceeding query limits for other cloud resources.

In computing the nearby restaurants, the program only considers restaurants with the same zip code as the input location. This design may sometimes overlook restaurants that lie within the distance range if the input location is close to a zip code border. However, matching zip codes improves performance by reducing the number of restaurant entries in the database that must be geocoded. One of the constraints of the program was that it had to compute nearby restaurants by accessing the database in memory, without caching results and without geocoding its entries beforehand. Thus, there was no other feasible and better option to reduce the number of geocoding requests other than filtering by zip code. In a related issue, geolocation would sometimes transoform the input location such that the zip code changed, and addresses that should have matched did not. To deal with this, the program manually checks to see if a zip code was provided in the input location using regex, and if so, uses that zip code to match against database entries.