COS 333 Spring 2015 Noms: Internals

Annie Chu, Evelyn Ding, Nathan Lam, Clement Lee, Zi Xiang (Sean) Pan

Princeton University

May 12, 2015

1 Overview

Our system is powered with an iOS app written in Swift and a Parse database backend with cloud code written in Javascript.

2 Frontend

We created the iOS app on Xcode on Mac OS X with the help of the built in iPhone emulator. Each of the views for the application is created on `Main.storyboard`. We also used storyboard to set up the components for the view, such as the buttons and images. All of the layout formatting was done with constraints on storyboard. The storyboard also shows the segues between each of the views, and a preliminary image of how the view looks like to the user.

Each view has a corresponding `Swift` file that outlines the applications response to user actions. We use some existing libraries such as Mapkit and Address in order to use Apple Maps

to provide mapping features for the user. Between different views, we save data in `NSUserDefaults` to pass information such as the restaurant user ID and the distance to the restaurant.

The provided emulator allows you to simulate running the application on any iOS device. The emulator replicates most functions of user interaction with the application, except for functions that require cellular reception such as making phone calls. The default current location of the emulator is the Apple store in San Francisco, so running Noms on the emulator will return restaurants in the closest college town, which is Berkeley, CA.

3 Databases

We use Parse's preset user class called `Parse.User` to handle user login. `Parse.User` provides the functionality to encode the password data for us. We also created two custom Parse Objects: `Preferences` and `Restaurants`. These objects can be created directly on `parse.com`, and the names and data types for each of the fields can be edited online. We save all preferences that have been created by all users of the application under `Preferences`, and we save all the restaurant data we have scraped and imported from Yelp under `Restaurants`. Some of the fields for each restaurant were provided through the Yelp academic dataset, which we imported as a `JSON` directly into Parse. However, several fields such as phone numbers, and optional information about the restaurant were not provided in the academic dataset, so we created a custom web scraper called `crawlyelp.py` to gather relevant additional information about the restaurants, and uploaded those fields to Parse.

4 Parse Cloudcode

Cloud code must be created and edited on any text editor on any operating system before calling `"parse deploy"` to upload it to the Noms Parse application. All our cloud code is under `main.js` and it handles the restaurant matching algorithm and the changing weights algorithm. The matching algorithm called `MatchRestaurants` first filters out restaurants that are not open. The int filters out restaurants within a degree of latitude and longitude of the user, this corresponds to a circle about 70 miles in diameter. It creates a four-dimensional vector for the given set of preferences, and another four-dimensional vector for each individual restaurant. The four-dimensions are distance, cost, options, and cuisine. The algorithm calculates the vector difference for each restaurant, and returns the a list of restaurants sorted by the Euclidean norm of the difference vector.

Our cloud code also contains a method called `ChangeWeights` that makes minute changes to the user's preferences based on his rejections of restaurants. If a user rejects a restaurant, his preferences are adjusted to account for that rejection by increasing the corresponding weights against the preference. This means that the next time the user makes a search for restaurants the cosine similarity scores will be computed with the new weights.