Laboon Capstone Fall 2017

Project Proposal: Yelp

Randyll Bearer and Zachary Barlotta and Raj Patel

September 21st, 2017

**Introduction**

The project proposal we have been assigned to fulfill is the Yelp industry project. This project entails building a purely voice reliant user interface for users to communicate with Yelp’s public API. Our person of contact for this project was initially Dean Thompson, however, upon our first meeting our collaboration with NoWait/Yelp Pittsburgh expanded to multiple team members. Our direct liaison with the company is still being selected on their end.

Our project proposal was unique from all the others in terms of open ended-ness. After meeting with Mr. Thompson and his team in order to elicit requirements and further brainstorm ideas, it became clear that only true design constraint for our user interface is that it must solely rely on verbal communication. While our initial research in the domain of verbal user interfaces and our brainstorming session with the Yelp team led to a number of ideas as to *what* to build, the *how* to build it was a decision left very much up to us. More detailed plans for development are specified further in this project proposal, but we will first provide here a brief summary.

We had many software choices, such as Amazon’s Alexa, Siri, and Google Home/Google Assistant, to pick from to complete our task. During our research, we found that Google Home’s were projected to be the home device of the future, so we decided to implement our program using a Google Home. Our goal is to create a piece of software which will run on a Google Home smart device which will enable users to query the Yelp database. This will allow for users to search for nearby restaurants based on a wide variety of criteria including food type, wait times, and others all hands-free from the comfort of their homes. The software will emulate a natural conversation, not relying on overly long dictation but rather requesting/confirming information in multiple brief sentences. The public Yelp API allows for a high level of specificity in its queries, which makes us confident that this software will be very effective in handling user demands.

**Project Details**

Undoubtedly the most integral part of this project is the Google Home device itself. Without its built-in Google Assistant API which will handle the parsing of spoken language, we would face a significantly larger roadblock to starting our project. But as it is, all we really need to create is a Google Action and Fulfillment on the Google Assistant platform. An “Action” can be thought of as a skill that the Home possesses, for example “Check the weather”, “Play this song”, or in our case, “Search Yelp”. The Action is not the actual code which will handle the request (that is the “Fulfillment”), but rather an intermediary which facilitates communication between the user and the code. The Fulfillment is directed by the Action to handle whatever the user requests.

The way the Google Home handles user requests is actually rather simple. A user makes a verbal request to the Home device, which is then parsed into string format by the Assistant AI. The Assistant AI then searches through its list of Actions to determine which one would be the best fit for the request it just parsed by matching keywords. The selected Action then confirms the user request, asks for any additional information, and then sends that request to its corresponding Fulfillment. The Fulfillment code will then handle the request, and send a response string back to the Action. The Action will then send the response string to the Assistant which will verbally communicate that response back to the user. A hopefully elucidating diagram is below:



As for the programming languages/tools required to develop our project, we will be utilizing Google’s API.ai to build our Action and the Node.js JavaScript runtime to code our Fulfillment. We plan to use Google’s API.ai simply because Google has left us no other choice since they require API.ai to be used for all Action development. Node.js was our decision however, as communication and interchange with the Yelp public API will be done through the use of JSON (JavaScript Object Notation) which is well supported by Node.js. This implementation, due to it being ran on the Assistant platform, should be compatible with all devices which utilize Google Assistant.

While the Yelp public API has been mentioned quite a bit by this point, we have yet to fully explain it. A more specific identifier would be Yelp’s Fusion API, their newest publicly released API and the sequel to Yelp 2.0. The Fusion API allows for six basic calls which can then be passed further arguments for more specific results. These calls are: Search, Phone Search, Transaction Search, Business, Reviews, and Autocomplete.

For our purposes we can disregard half of these API calls, namely Phone Search, Reviews, and Autocomplete. As far as expected use cases and pushing out a minimum viable product goes, we and the Yelp/No Wait team do not see a dire need to implement the ability for users to search for restaurants by phone number- it is simply a rarely used feature. The Reviews call returns up to three user review excerpts about a specific business. While this is a potentially useful feature, we are wary of presenting the user with any arbitrary paragraph written by some random person on the internet. Lastly, Autocomplete will not be implemented as it is used exclusively for text-based user interfaces and displays. Autocompleting a half-typed word simply cannot be applied to a pure voice user interface.

That leaves us with three API calls to handle our user requests with: Search, Transaction Search, and Business. The Search call allows us to filter nearby restaurants through keywords which describe what the restaurant *is*, i.e. food category, price level, location, etc. The Transaction Search call allows us to filter nearby restaurants through keywords which describe what the restaurant *allows*, i.e. reservations, public parking, family environment, smoking/non-smoking, etc. Lastly, the Business call does not perform a search but rather returns more detailed information about a specific restaurant which the user requests by name.

Using these Fusion API calls, we intend to implement the following features which we and the Yelp/NoWait team have identified as most important:

1. Allow for “vague” search (e.g. Find me a pizza/healthy/burger place near me)

2. Allow for “utility” search (e.g. Find me a place which delivers/has street parking)

3. Allow for “wait time” search (e.g. Find me a place with under a thirty minute wait time)

4. Allow for “direct” search (e.g. Yelp Piada/Burgatory/Primanti Brother’s)

5. Allow for “random” recommendation (e.g. Yelp, where should I eat tonight?)

Footnote: It is worth noting that we were advised that the “random” recommendation should in fact not be truly random. Rather, we should try and incorporate past user searches/reviews so history storage will need to be addressed. Was decided upon as a feature to avoid “Where do you want to eat tonight”/“Oh I don’t know” scenarios. The Yelp/NoWait team was very adamant about this particular implementation.

As mentioned in the introduction, our ideal conversation flow consists of multiple brief confirmations/requests rather than singular monolithic dictations. We hope that by constantly engaging the user with brief questions that they will be more willing to provide specific details which should help our queries be more accurate. Also in our opinion, and those of our friends/family we have surveyed, no one wants to sit there and listen to a robot prattle on for minutes on end.

For example, instead of a conversation like this:

“Okay Google, search Yelp for any nearby pizza places which deliver”

“Okay, Yelp has found four nearby pizza places which deliver. They are Pizza Hut located on 123 Baker lane, price level average, reviewed 3.5 stars, reservations no, street parking yes, smoking no, children atmosphere yes...”

\*User falls asleep\*

We would much rather facilitate a conversation like the following:  
“Okay Google, search Yelp for any nearby pizza places which deliver”

“Okay, Yelp has found four nearby pizza places which deliver. They are Pizza Hut, Domino's, Papa John’s, and Little Nipper’s.”

“Alright, tell me about Domino’s”

“Domino’s is located on 125 Baker Lane, price level average, reviewed 3.4 stars.”

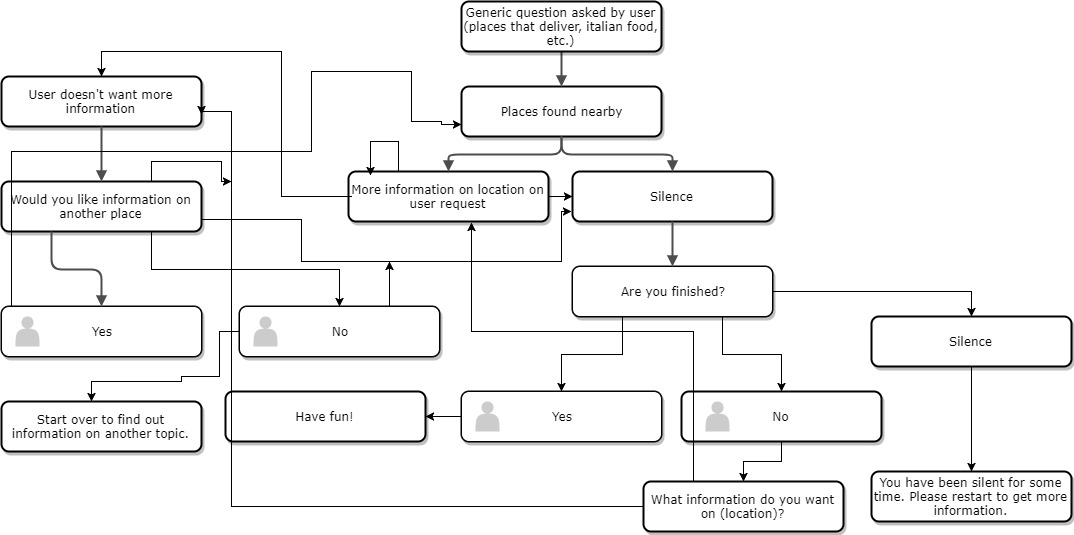
“What’s the current delivery time?”

“The current delivery wait time for Domino’s 125 Baker Lane is 25 minutes”

“Okay, what’s the phone number?”

“Domino’s 125 Baker Lane phone number is 111-222-5555”

To achieve this latter conversation, we intend to provide a very wide branching conversation pattern. This will require our Action/Fulfillment to be able to handle a *large* number of requests, but they are all fairly similar and should be not too difficult to implement. If we need to handle requests which are not handled by the API as well as we would like, Yelp also provides another framework called GraphQL. GraphQL allows for far more detailed queries at the cost of usability, but it is always an option for difficult problems. The diagram below is a flowchart of how most conversations would go.



**User Stories**

Our user stories are centered around users who we feel would need services relating strictly to dining. The stories we want to focus directly correlate with the core features we seek to implement. The following is a list of prioritized user stories:

1. **“As a general user, I want a ‘direct’ search feature, so that I can get information about a certain location.”**

We feel this is the most important feature we should work on. The average user of a location based search app already has a good idea of the eating options around them, but are generally unfamiliar with every single detail about them and the utilities they provide. This feature allows them to obtain more specific information about the restaurant, hours, address, reviews, etc.

1. **“As a local resident, I want a ‘vague’ search, so that I can review eating options in my area based on my appetite.**”

This would be our next priority. As mentioned before, someone who lives in a certain area is going to have a good understanding of the nearby eating locations. However, the resident is likely to be in the mood for a certain kind of food, so they would want to narrow their search for places to eat to places that fit their current appetite, i.e. pizza places, Mexican restaurants, Asian-style buffets, etc.

1. **“As a traveler, I want a “random” search feature, so that I can find a place to eat in an unfamiliar area.”**

This user stories prioritizes the handling of users that are foreign to their current locations. If someone is on vacation or on a business trip for a few days, they may not be familiar with any of the restaurants around them. If they were to forgo the direct and vague search, they would be in a place where they just wanted *some*thing to eat, and so they would perform a random search. This feature would give them a list of a few popular places to eat in the area.

1. **“As a business manager, I want a “utility” search feature, so that I can know which eateries around me offering delivery/catering services for company events.”**

For this user story, we considered what other kind of information is useful to someone who has found a place they want to eat. In a corporate environment, a business manager can organize events to meet with sponsors and other companies, and for these events, they would probably provide refreshments to their guests. An excellent way to do this is to get catering services from a certain location. This feature is useful for obtaining many piece of information about a location. If it’s raining, a user may want to know if the place they want has a delivery option.

1. **“As a student, I want a ‘wait time’ lookup feature, so that I can manage my schedule more efficiently if I want to eat out.”**

Finally, we thought of a feature that is useful in daily scheduling. Say a student has a very busy schedule but he would still like to meet up with friends later for dinner. The student hasn’t made a reservation but would still like to know how much time he should allot in his schedule for dinner, including the wait time. Another utility this feature would work with is delivery time. When people order food, they will always want to know how long it will take to get there.

**Going Forward**

Going forward, we will be communicating with Yelp on a regular basis as well as each other. In the first meeting with Yelp/NoWait, it was determined that we would meet in person every other week to discuss the progression of the project as well as other ideas for the project. As for us as a group, we will continue to communicate regularly using our group texting thread, and we will meet at various times during the semester to discuss our progress with the project as well as test the project. For the scrum master, the rotation will be as follows: Randy, Zack, and Raj. This rotation will stay consistent throughout the semester.

To start off, we will implement the minimum key features discussed earlier. Once these features are successfully implemented, we will implement some of the features Yelp/NoWait suggested, such as the user reviewing the restaurant they just left using the voice system, adding a language filter, expanding to more than just restaurants, etc. We hope to implement as many of these features as possible, but our main focus is getting the minimum features completed and working in full.

When we test our program, we will be testing as we go. Most times, we will run down a basic conversational dialogue to confirm the program is returning the correct outputs. Depending on which feature we are testing, we may tweak our basic dialogue to fit the parameters of the feature we are planning to test at that given time. As for our planned testing device, we will either use a physical Google Home or Google’s online testing program they have specifically to test programs. This will depend on if we can get our hands on a Google Home. With that being said, Google’s online testing software will get the job done just as well as having the physical device, so our options are open as for which we prefer to use to test our program.