

FoodFinder

Table of Contents

[Overview](#)

[Process](#)

[1. Read the Assignment Description Thoroughly](#)

[2. Market Research on Similar Products](#)

[3. Interview a Few Others on Their Experiences](#)

[4. Draft Design/Mockups of the Product](#)

[Overall](#)

[Search](#)

[Filters](#)

[Result Card](#)

[5. Build FoodFinder](#)

[Technologies](#)

[Why use Yelp Search API instead of Yahoo Local API?](#)

[Why Node?](#)

[Why Not a JavaScript Framework?](#)

[Why Bower, Bootstrap, and Font Awesome?](#)

[6. Review code and Find Bugs](#)

[**Yelp Search API Bug](#)

[**Update on Yelp Search API Bug](#)

[7. Write Documentation and README](#)

[If I Had More Time](#)

[Conclusion](#)

Overview

This document covers the technical details of *FoodFinder*, including the full design process, decision making, and features I would add if I had more time. If you are looking for directions to getting *FoodFinder* running on your local machine, check out the ***README***.

FoodFinder is a web application for finding places to eat based on location, type of food, ratings, and distance. The search feature also supports other businesses besides restaurants (e.g. haircut, clothing, car repair), but because the scope of the project was for finding restaurants, I chose *FoodFinder* as the name.

Process

In this section I will walk through the steps I took when building *FoodFinder*. I treated this project with the same approach of a software team that is trying to appeal to a market segment and potentially have millions of users. I essentially took 7 steps while building *FoodFinder*, and they are all explained below.

1. Read the Project Description Thoroughly

It's always best to make sure that you build what is actually asked. In this project I was asked to build an interface that finds a restaurant based on cuisine type and location using the Yahoo Local API. My tool does find restaurants based on location and other factors but it doesn't use the Yahoo Local API, but I will explain later why I made that decision.

2. Market Research on Similar Products

After I realized that I wanted to build a web tool for finding restaurants based on cuisine and location, I then did some research to see how the product was currently implemented in the market. I wanted to get a base on what the current products do well,

and also issues with their solutions. I then wanted to see if I could find a niche to build my product differently so that it does address some of the problems with the current products. For the sake of time, I decided just to analyze three products, and they are all listed on the next page. Not all of them are specifically targeted at restaurants, but each business is fundamentally built on information gathering for different products and market segments, which is what I am trying to build. I chose:

- **Places to Eat Near Me** - <http://places-to-eat-near-me.com>
- **Yelp** - <https://www.yelp.com>
- **TripAdvisor** - <https://www.tripadvisor.com>

After using all 3 of these products, here are some of the notes I took that I used as feedback for *FoodFinder*:

- **Searching is the product**: having a great search experience is crucial
- **Result filtering definitely helps**: makes finding what you want simple
- **Don't clutter the results**: including too much information can confuse the user

The three takeaways above really resonated with me while trying Places to Eat Near Me, Yelp, and TripAdvisor, and had a major hand in building *FoodFinder*.

3. Interview a Few Others on Their Experiences

To filter out the bias in my opinion, I asked a few friends about their experiences using these products. Because Yelp and TripAdvisor are much more popular in the market, the feedback I got was just on those products. A summary of the feedback I received from a few of my close friends were:

- **Searching on vague key phrases really makes the experience pleasant** (e.g. *sushi in san mateo* or *coffee near me*)
- **Using a map of the region as a visual to see the results**
- **Searching based on distance from a specific location**

After hearing this feedback, I decided on a few principles for the design of *FoodFinder*:

1. The search fields needs to be text input

- a. Much faster for the user and gives them options for more powerful queries that are not predefined
- 2. The search fields must be able to handle vague input**
 - a. The input for the cuisine type and location is completely dependent on what the user types
 - b. Search should still produce great results even with typos (e.g. *piza* instead of *pizza*)
 - c. Categories of food and locations that are not predefined by my interface (e.g. *brunch in sf* should yield breakfast places in San Francisco)
- 3. Filtering results on a few key categories**
 - a. Cuisine, Distance, Reviews, etc
- 4. A simple interface that only gives necessary information**
 - a. Any extra information should be eliminated
- 5. Responsive website that can be viewed from any device**
 - a. Currently any website that isn't responsively designed is really far behind

These principles were a major factor into some of my feature decisions and the overall design of *FoodFinder*. I wanted the interface to be extremely simple, and only provide necessary information to find a place to eat. I also wanted the search capabilities to be powerful and able to handle ambiguous input from the user.

4. Draft Design/Mockups of the Product

After getting feedback from some peers and looking at products in the market, I then drafted on paper some simple designs for the layout of *FoodFinder*. I don't have any pictures of the original designs but screenshots of the final application will still show my intent for the experience.

Overall

I wanted the overall layout to be a very simple application. Just a one page tool that has search capabilities at the top, and results rendered below. Here is a screenshot *FoodFinder* when first loaded by a user:

The screenshot shows the top section of a web application titled "FoodFinder". Below the title is a subtitle "Search below to find your next meal." There are two search input fields: "I want ..." with the placeholder text "sushi, brunch, coffee, cheap food..." and "Around" with the placeholder text "address, city, state, zip, or neighborhood". Below these fields is a "See more" link with a downward arrow. A "Search" button is positioned below the "See more" link. The bottom of the page features a footer with the text "Created by Brad Ware. © 2016 FoodFinder. All rights reserved."

As you can see, there are just two search bars and a small description of the application at the top. The bottom is empty because the user has not searched yet. This layout is simple and follows suit to most other sites with search functionality.

Search

For search, I wanted the user to input as little or as much information as they desired. *FoodFinder* only requires a location to be specified for rendering results. The original two options are search inputs for the type of food and the location.

This screenshot is identical to the one above, showing the "FoodFinder" search interface. It includes the title, subtitle, two search input fields ("I want ..." and "Around"), the "See more" link, and the "Search" button. The footer text "Created by Brad Ware. © 2016 FoodFinder. All rights reserved." is also present.

I also display a *See more* option which renders more ways to filter results. This is provided to give any user the ability to make powerful queries but is also hidden to give the search component a simple look.

FoodFinder

Search below to find your next meal.

I want ...

sushi, brunch, coffee, cheap food...

Around

address, city, state, zip, or neighborhood

Radius (mi.)

35 mile range

☐ Use current location☐ I'm feeling lucky ⓘ

Search

Using the extra search component, a user can specify their location, a distance radius, or *I'm feeling lucky* (explained in the next section). If the *Use current location* filter takes too long, it will alert the user and timeout the query.

Filters

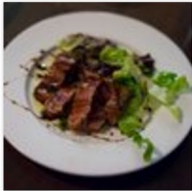
After the user has initiated a search, the results are rendered below with filtering options at the top. The default option is *Best for you*, which the Yelp API factors in distance, ratings, and search preferences. The other filtering options are *Distance*, *Ratings*, and *I'm feeling lucky*.

Search

Sort by: Best for you ▾

Ryoko's

Sushi Bars



★

★

★


★

★

+1-415-775-1028
619 Taylor St
Lower Nob Hill
San Francisco, CA 94102

Saru Sushi Bar

Sushi Bars



★

★

★

★

★

+1-415-400-4510
3856 24th St
Noe Valley
San Francisco, CA 94114

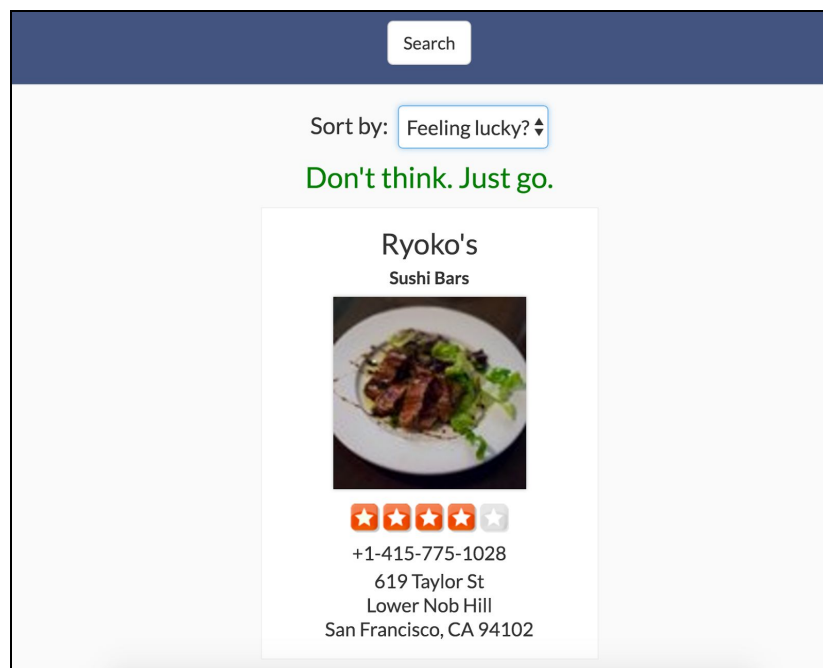
Akiko's Restaurant

Takoba

I'm feeling lucky has the exact same functionality as it does for Google search. It renders one result, so that the user does not have to spend time comparing every

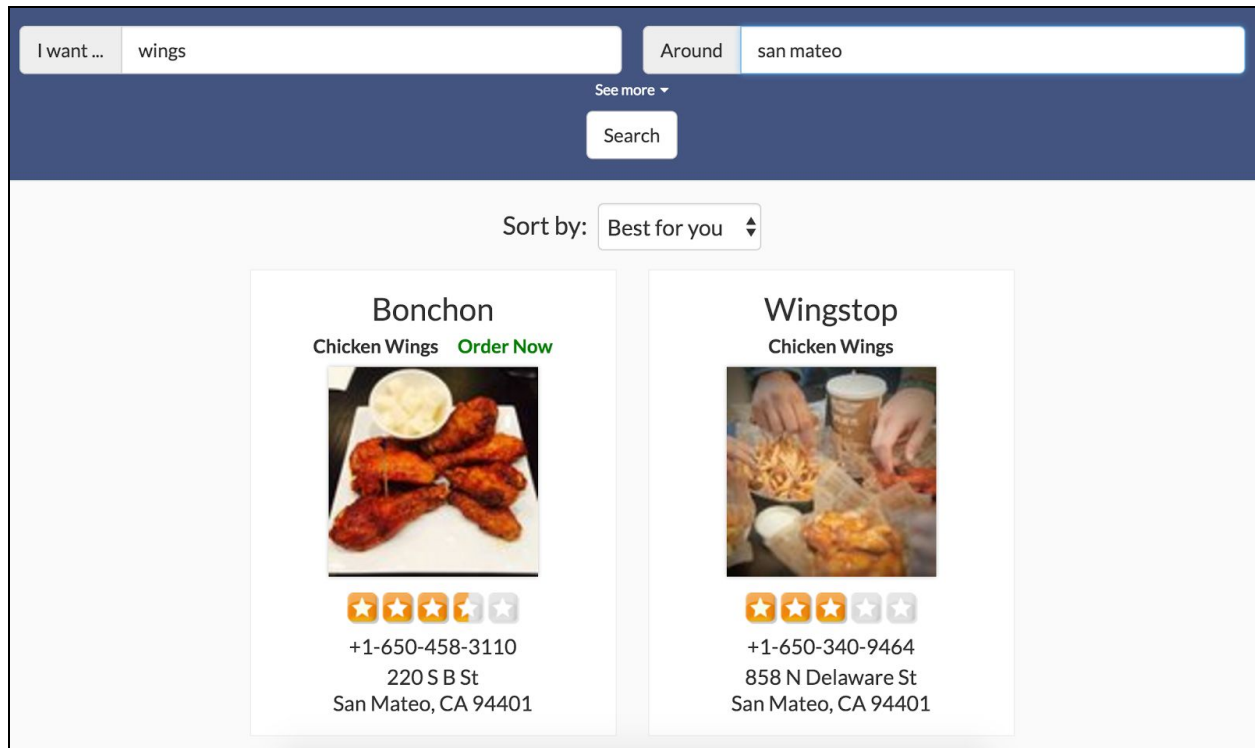
option. Many times my group of friends have argued which place we should eat simply because we had so many choices. One of my main goals for *FoodFinder* was to provide a simple experience for the user, and I feel that this feature definitely helps the user find a place to eat without hesitation or debate. The paradox of choice limits my decision making all the time, and I wanted to give the users of *FoodFinder* the option to just be given a place to eat (screenshot on next page). The other filter options reorder the results based on the ratings they have received, and also their distance from the location specified.

Every time a filter is selected by the user, a new query is sent to the Yelp API to retrieve the results. A loading icon is provided as feedback to the user so that they know the current state of the application. This is important because sometimes the query can take a few seconds if the user has opted to use their current location. However most queries are extremely fast when a text location is provided.



Result Card

The information received back from the query is the most important part of the application, and I wanted to make sure it was helpful but also easy to understand.



Above we have two search results that are displayed as result card components. Each result card displays the basic information about the result such as the name, address, phone number, reviews, and an image. The name, phone number, and address are all hyperlinks where you can find more about each if desired by the user. These categories were the minimum that I wanted to include in each search result, as the user knows enough information now to visit the restaurant's website, or call for more information. I also included an action link for booking a table or ordering online (if the cuisine supports it) through the services Eat24 and SeatMe. The design of the result card is simple, contains essential information about each result, and doesn't clutter the page with reviews from other customers. The goal of *FoodFinder* was to provide a simple interface for finding restaurants near you, and I wanted the result cards to do exactly that, while leaving out excess information.

5. Build *FoodFinder*

When building web applications now, every developer is faced with the decisions of what tools to use. There are so many different options for each level of the stack, and

everyone has a different opinion on what to use. Below I discuss what technologies I used to build *FoodFinder*, and why I chose them.

Technologies

- [Node.js](#)
- [Express](#)
- [Node-Yelp](#)
- [Bower](#)
- [Bootstrap](#)
- [Font Awesome](#)
- [jQuery 2.x](#)
- [Yelp Search API](#)

Why use Yelp Search API instead of Yahoo Local API?

I chose to use the Yelp API for a few reasons. The first being that I wanted *FoodFinder* to support keyword search to enrich the user experience. People make spelling errors all the time while typing, especially when in a hurry. Modern day software is expected to handle input like this and still decipher what the user meant to type, producing correct results. I held *FoodFinder* to the same standard, and I knew that Yelp's API supported keyword search. With *FoodFinder* querying Yelp's API, one can simply type *fancy dinner in sf* and get accurate results for expensive restaurants in San Francisco, CA. When I was researching the Yahoo Local API, I could not find this functionality implemented, and that's when I decided to try Yelp's API. I also emailed **Adam Cheyer** asking permission to use Yelp's Search API instead of the Yahoo Local API, and he said it was fine.

Why Node?

I chose to use [Node.js](#) as the back end simply because I was familiar with it, and it's very easy and lightweight to get up and running. I originally tried to query the Yelp API just on the client, but I was getting security exceptions for having my API consumer key and token available for anyone to find in the browser. I then decided to query the Yelp API from a back end request, and knew Node would be simple. It also has access to [NPM](#) and [Express](#), which makes routing CRUD requests straightforward. All of the server side code can be found in my *server.js* file, which is very small. My API is really lightweight - it only supports GET requests which fetches results from Yelp's Search API and sends it back to the client.

Why Not a JavaScript Framework?

Almost every modern day web application uses a JavaScript framework, some of the most popular being [AngularJS](#), [React.js](#), and [Ember.js](#). The only framework that I am somewhat comfortable using is Ember, and even then it includes features that I would have never needed in *FoodFinder*. Therefore, I decided to just use vanilla JavaScript and [jQuery](#) because my app is really not that complicated. If I was going to use a framework I would have used React.js, but unfortunately I don't know it, and I've heard it has a decent learning curve. For this project I decided to rely on a lightweight set of technologies that I had previous experience with in order to move fast. My full list of pros and cons for jQuery is below:

Pros

- **Lightweight** - jQuery is simple, no build tools required
- **Past Experience** - I've used it before
- **Universal** - The majority of websites use jQuery

Cons

- **No Data Binding** - Must manually update DOM elements when data changes
- **No Templating** - Must manually build HTML through strings - error prone
- **No Pattern Support** - Hard to keep the code clean and uniform

Why Bower, Bootstrap, and Font Awesome?

All three of these technologies are really popular and easy to get running on a website. [Bower](#) makes managing all of one's client side assets really simple. [Bootstrap](#) is probably the most popular styling framework, and I've used it in many projects before. [Font Awesome](#) allows me to use icons as svgs instead of images so that they render quickly and can be injected as DOM elements into my HTML. These technologies allowed me to build *FoodFinder* responsively without spending all my time on tedious CSS styling.

6. Review code and Find Bugs

After building *FoodFinder*, I went back through the interface and tried to find bugs. As always, there were plenty and I'm positive that more bugs still exist. I eventually had to cut myself off from adding any more features due to time constraints. There is a bug in

the Yelp Search API that I emailed the team about, but they never responded. Details are below.

****Yelp Search API Bug**

In the Yelp Search API, one can pass latitude and longitude coordinates instead of a text location and get back results. However, when I make a request for coordinates positioned at a *Philz Coffee* in San Mateo, CA the API always returns results in Ireland and the UK. I emailed the team about the bug, but they didn't respond. You can reproduce the bug doing this below:

1. Visit Yelp Search API [console](#)
2. Take out San Francisco, CA in the location input
3. Put any type of food you want to search for in the Find input
4. Copy [&location=near&cll=37.5671463,-122.323721899999998](#) into the query box
5. Google maps [page](#) result of *lat* = 37.5671463, *long* = -122.323721899999998
6. Results from Yelp API Console: **2 Port Road, Letterkenny, Co. Donegal, Ireland**

****Update on *Yelp Search API Bug***

I fixed the bug in *FoodFinder* so that *Use current location* returns results near the user's location instead of Ireland and the UK. The steps above still reveal results in Ireland, where according to their [documentation](#) they should yield in San Mateo, CA. But for the sake of *FoodFinder*, the functionality now works and finds results near your current location. I just now use *ll=* instead of *cll=* in the GET request to the Yelp Search API.

I also reviewed and cleaned up my code, putting comments above functions and different components of the logic. I decided to encapsulate Result as a JavaScript object with it's own set of private methods and properties.

7. Write Documentation and README

Finally, I wrote the README for getting *FoodFinder* up and running, and also this document explaining why and how it was built.

If I Had More Time

I would do a couple of things if I had more time to work on *FoodFinder*. I have listed them below, and they are NOT ranked in any order of importance.

- **Build *FoodFinder* in [React.js](#)**
 - I've wanted to learn the library/framework for a while, and if I had more time I would learn React and then build *FoodFinder* using the library
- **Add a maps feature to the results component**
 - It would provide a better visual for the user and it was also cited in feedback from my user research
- **Add logic to deciding the *I'm feeling lucky* result card**
 - Right now it just takes the first result and renders it on the client
 - I would like to implement a weighted algorithm that incorporates randomness when deciding what card will be rendered for the user
- **Produce better UI feedback on search errors**
 - Right now there is just one error message when no results are returned, but I want to add more detailed feedback based on the error code returned from the server
- **Iterate on user feedback**
 - I would love to test *FoodFinder* on many different types of users and get feedback on what they like, dislike, ease of use, etc
- **Refactor and Rewrite**
 - I tried to keep the code clean, but it could always be refactored. Rewriting *FoodFinder* in React.js with a templating markup would make the code base more modular and reusable

Conclusion

FoodFinder was a fun project and I enjoyed building it to completion. Thanks for reading this documentation, even though it was long I hope it gave you better insight into my decision making and thought process while building *FoodFinder*. If you have any more questions you can find my contact information on my [website](#) or send me an email at bwarenohs@gmail.com.