Final Report: Study Yelp

An application to help students find the best study spaces in their surrounding area based on their preferences.

Alan Trinh , Meerza Ahmed , Melissa Perkins

DSCI 551 - Foundations of Data Management
Prof. Wensheng Wu
Spring 2024
5/3/2024

Website: https://dsci551-finalproject-756e6.web.app/home

Github: https://github.com/MeerzaA/DSCI551_Project

Table of Contents

1. Introduction

Our goal for this project was to create a Yelp-like website specifically catered to students. As current USC students in the DEN program, we recognized the need for a convenient solution to locate ideal study spots in our area. Unfortunately, for some students who do not live near a university, finding a space that offers the same amenities or the same quiet environment that a library would offer can be challenging. For example, cafes do not always offer WiFi or outlets for charging devices, and some cafes play music that can be loud and distracting. We also recognize that it can be refreshing for students who live near universities to take a break from the typical study spaces and find somewhere new that offers a similar environment.

Our Study Space Finder website aimed to identify traditional study locations such as cafes, restaurants, quiet corners in local neighborhoods, and other co-working spaces that students can access. Our website will help students, regardless of their mode of learning, find a comfortable and productive environment in which to excel in their studies.

2. Planned Implementation

Our initial implementation plan was to use 3 Yelp datasets—one containing business information, another with reviews, and one with photos. We planned to store the data in 2-3 Firebase databases and host the website using Firebase SDK but to implement the data manager portion of the project using Python queries. For the backend development, we planned to create a hash function to create smaller data sets to make data retrieval more efficient and avoid crashing. For data managers, we planned to give them Command Line Interface (CLI) access to the data. For front-end users, we planned to create a user-friendly interface where users could use a search bar to make calls to the Firebase RealTime Database.

While we were able to keep the essential functionality of our plans, we had to make some changes due to certain limitations, such as time and storage space. The details of these changes will be outlined throughout the report.

3. Architecture Design

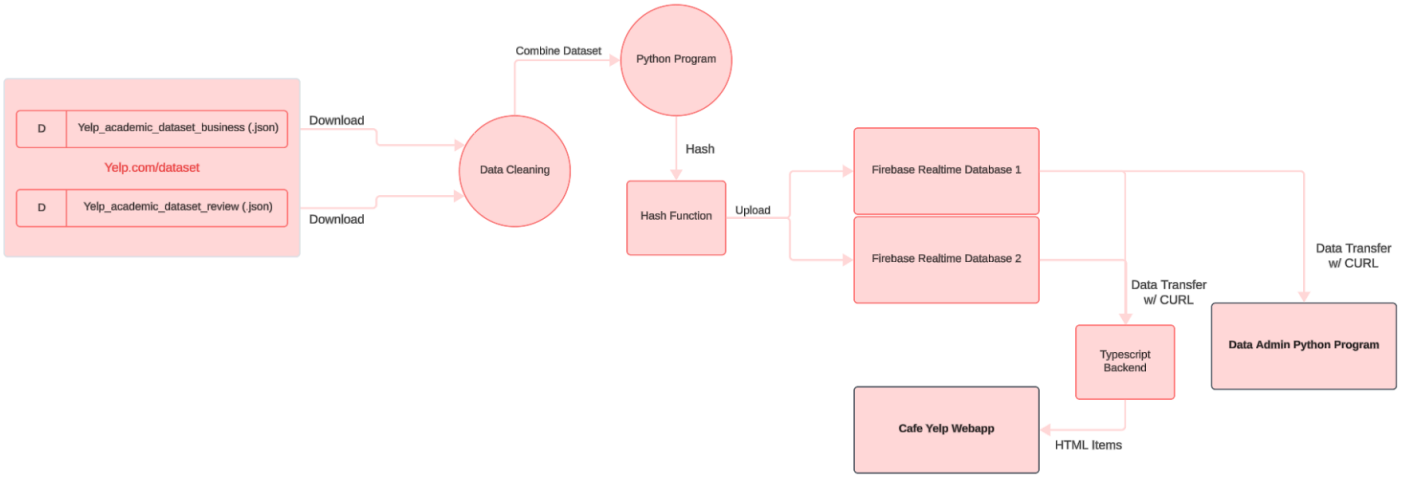

Fig 1. Data Flow Diagram

The data flow diagram (Figure 1) follows our process of downloading the dataset and piping it into our two endpoints. We first downloaded our datasets from Yelp.com/dataset in JSON format and combined them into one extensive dataset during data cleaning. After data cleaning, we processed the dataset through a Python program and uploaded it to Firebase using a hash function. The data is then accessed and processed using the REST API to send it to our two endpoints: a live web app and a Python program.

3a. Data Processing

Dataset

We used two Yelp datasets: one containing all the business data and one containing the reviews. There are 7 million reviews, 150,000 businesses, and 11 metropolitan cities. The Yelp business dataset contained 150,000 rows of data with 14 columns: business ID, name, address, city, state, postal code, latitude, longitude, stars, review count, open, categories, and attributes. There are 39 attributes, including alcohol, suitable for kids, outdoor seating, wifi, etc.

Data Cleaning

We used the Python Pandas library to clean all of the data. For the business dataset, we first filtered the categories on “food,” “cafe,” and “restaurant” because all of the study spots we wanted fell into these broad categories. Then, we removed unwanted attributes and columns. Next, we removed the

null values and normalized the Unicode to replace them with their ASCII equivalents. Finally, we put all the data into .json format by nesting all the data under the business_id and sub nest if needed.

Since the review dataset was so large, we had to split it into chunks for faster computing power. We did this by breaking the entire dataset into four groups of 7.5 MB, with the last group having the rest of the data. This significantly helped with computing power. Next, we merged each group with the business data on the primary key (business_id) with an inner join. This left us with five merged tables containing all the business data and reviews. We then merged the four merged tables from the split groups and put those into our Firebase Database 1 and the last merged table into Firebase Database 2 (Figure 2).

Fig 2. Screenshot of the Two Firebase Databases

3b. Website Building

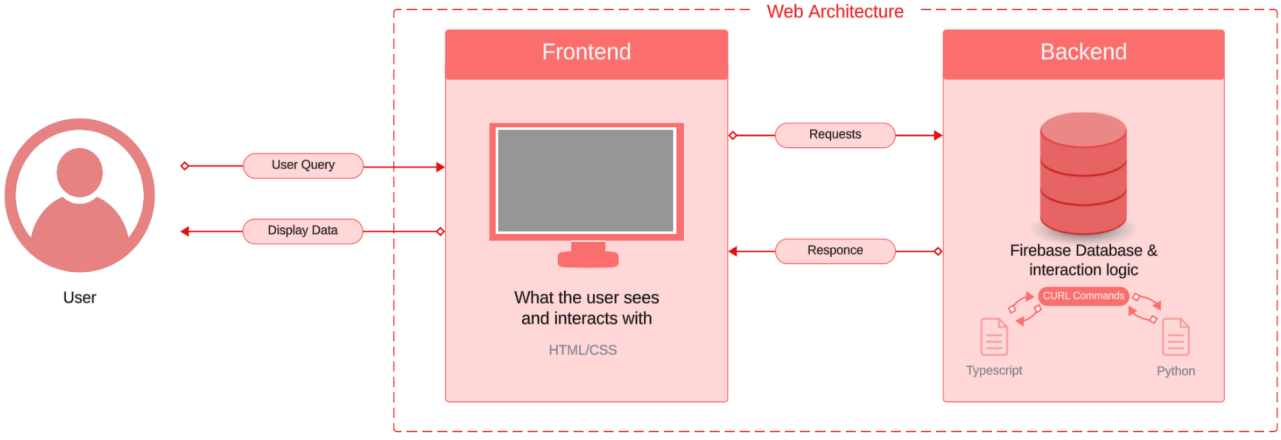

Fig 3. Webapp Architecture Diagram

Our web app architecture is shown in Figure 3; we used the Firebase SDK to host and deployed our website to a live link that can be accessed online. The front end has a UI where our users can enter a search query; this then generates a request to our backend using Typescript and REST API. The query responds with the queried data without downloading the whole dataset and sends it to the front end. The data is then processed into HTML elements, which are nicely formatted and displayed to the user.

Deploying Website

Our website was deployed using the Firebase SDK. This was the best method for us to host and publish our web app to the Internet, as it handled data hosting to a live URL. We had the flexibility of choosing a frontend framework that we knew Firebase could deploy before writing HTML/CSS code; this was also great for picking libraries in our code and not worrying about conflicting versions. Our database was also distributed and hosted online in a Firebase real-time database, which meant we could access all of our project's individual components in one project file. This was great for version control, among other benefits.

Front-End Development

The front end was developed using Angular, a TypeScript-based framework. Angular utilizes modular components to build the webpage, and any changes made in one component do not necessarily affect the entire webpage unless they are connected in some way. When initializing the project, Firebase generated the App components. We then developed a Layout component, a Header component, a Home component, and a Content component.

Our App Component is where we created the routes using the app.routes.ts file. The user enters their search on “/home,” and the results are displayed on “/search”. The App component also renders the Layout component in app.component.html. The Layout component is used as a grid to hold the Header and the Home component together. The Layout component renders both the Header and Home components. The Header component contains the sidebar and the title “Study Space Finder.” The Home component contains the search bar, saves the user entry as a variable, and passes it to the Content component. The content.component.ts file is where the queries are made to the Firebase Real-Time Database. The content.component.html file renders all business information that meets the search criteria. The content.component.html file also renders the Header- looking back, we likely could have avoided using the Layout component. However, it was our first time using Angular to develop a web application.

4. Implementation

4a. Functionalities

Data Managers

Data managers can use the command line interface to interact with both Firebase databases. Data managers can add a study spot where a unique spot_id will be generated and outputted to the user and a return 200 message if successful. Similarly, data managers can modify information for any spot_id and delete any data point by spot_id.

Our search features allow the data manager to find study spots in many ways, including by city, state, postal code, and spot_id. Data managers can also search by nearby distance by inputting their latitude, longitude, and radius in kilometers.

Front-end Users

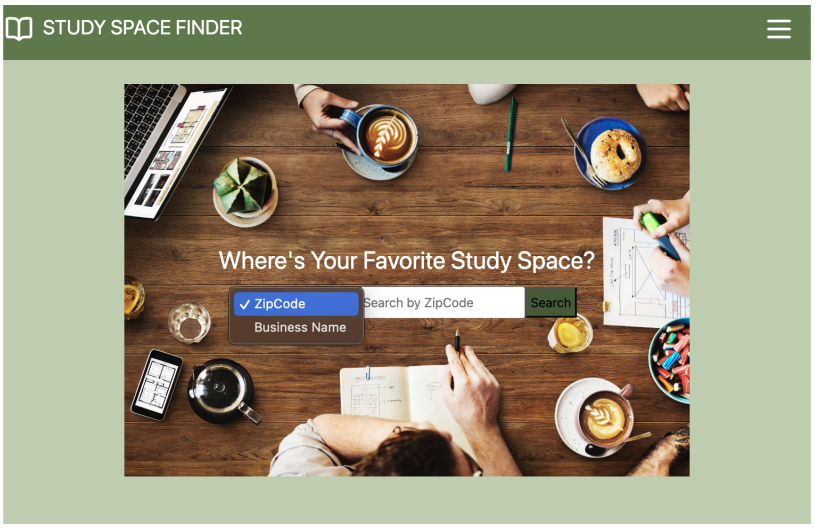

Fig 4. Home Page with Search Bar

Front-end users can access the Study Space website and search by either business name or zipcode (Figure 4). If there are businesses that match those search criteria, companies will populate the page. Below the business's name, we have included attributes such as “WiFi.” “Group Friendly,” “Dog Friendly,” “Outdoor Seating,” and more (Figure 5). These attributes were chosen because they were available from the Yelp dataset, and we believed students might want a study space. We created a filtering option on a small selection of these attributes (Figure 6). For businesses that fall within the search criteria, users can also use a filter to select which options they need.

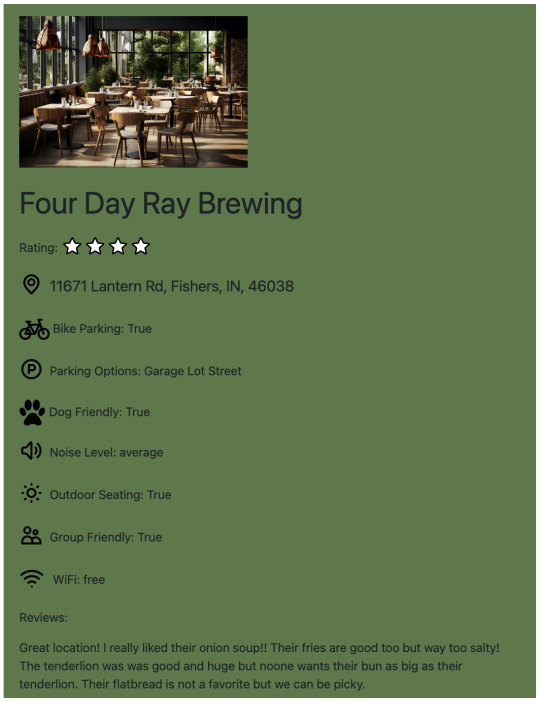

Fig 5. Business Information and Attributes

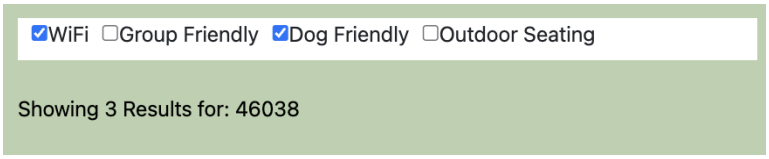

Fig 6. Filtering Options

The web page's header has an option for a sidebar that opens a dropdown menu (Figure 7). While we did not have enough time to implement this sidebar, we included a few options that may be useful for future implementation. Finally, in the header, if the users click on the “Study Space Finder,” it will take them back to the search page.

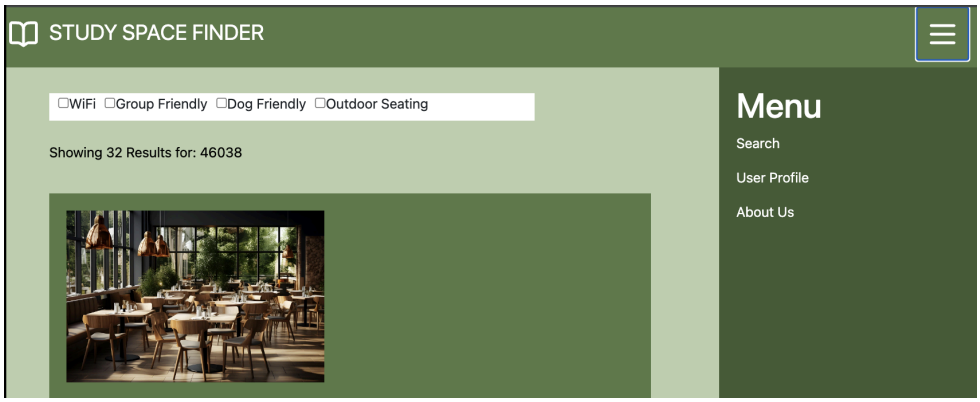

Fig 7. Menu Sidebar

4b. Tech Stack

Frontend	HTML/CSS, **Hosting**; Firebase SDK, **Framework**; Angular
Backend	Python, Typescript, **Data Interaction;** REST API / CURL

Frontend

For the front end, we developed our web application using the Angular Framework. Angular uses TypeScript files, HTML, and CSS in each component. HTML files were used to render the various components. Typescript files were used to create functions to query the Firebase Realtime Database. Typescript files were also used to create the functions within the webpage to register which buttons or checkmarks the user is selecting. CSS files were used to help with styling the web pages.

Backend

We used Python for all data manager tasks utilizing libraries such as JSON, requests, sys, uuid, and geopy. We created a Firebase URL dictionary to house our two Firebase database URLs. Our hash function works by taking in any location element, such as city, postal code, or state, and returns the absolute value of the hash value. UUID creates a unique spot ID and converts it into a key element under which the rest of the data is stored. We created a filtering option into a string. The data will go into whether it is even or odd. After determining the hash value, we use rest calls to add the data to our database. Our search functions will take in parameters (city, state, postal code, or ID) and loop through both databases to find matching results. A get request does this and outputs 200, and the matching results are successful.

Our search by distance function takes in parameters: latitude, longitude, and km (radius) and uses our calculate distance function, which calculates the distance between two coordinates using geopy. After taking in the origin location and the maximum distance in kilometers, the function loops through both databases and requests to receive the data.

Our modified study spot function searches through both databases to find that specific spot ID and does a request put command to update the data. The delete function does the same thing but does a request delete command.

Data Interaction

We utilized the REST API for all database interactions, including generating HTML elements, and all functions used by the data administrator. For the frontend, we used the axios library in Typescript to

create CURL commands that query our Firebase databases. This returns an array of elements which we iterate through to create div elements and store the business attributes within them. This is how our frontend displays data to users directly from our database without the need to download the entire dataset.

We employed a similar approach for the data manager by using Python. Each Python function uses CURL commands to query our databases and prints elements in JSON format. The program runs on the terminal, where you can write commands that directly query any of our Firebase databases and return only the queried attributes.

5. Learning Outcomes

5a. Challenges Faced

Data cleaning was a big issue since we initially thought the datasets could be uploaded to Firebase directly as they were in JSON format. However, the dataset collected individual JSON items instead of one extensive collection. It took a lot of work to properly nest all the data under the business ID and subnet categories and attributes. A large portion of the data was stored as unicode which needed to be converted to ASCIII. The dataset was extensive, adding multiple complexities and challenges relating to computing power.

One of the biggest challenges for the front end was being unfamiliar with Angular. There was a learning curve associated with building the website and learning how to implement the different components. Many hours were spent researching and reading documentation (we included the references in our code), and ChatGPT was used to help debug errors.

Initially, we planned to use a template. However, this did not work due to conflicts in versions for angular and other dependencies required by the templates. We also attempted to implement several features, but we needed more time to get them all to function. For example, we wanted to implement buttons that hide and show business hours and reviews. This would have been more efficient for the user to focus on only a few attributes simultaneously. However, this feature needed to be fixed, likely due to how the business information cards were added. It would not allow for the buttons to update.

Another feature we wanted to include was the ability for users to see how far a business was from their current location or within the zipcode they searched. To determine the user's current location we would have had them enter their city and state. We would then use an api to get the coordinates for

that location and use Turf, a JavaScript library for spatial analysis, to calculate the distance. Unfortunately, we did not have enough time to properly implement this and we had to remove the turf dependencies because it caused an issue when we tried to deploy the website.

Finally, we were unable to implement an inclusive search when searching business names. For example, if a business has the name “Joe’s Bagels,” a user cannot search “Joe” or “Bagel.” Our current search function must be exact. If you are searching for “IHOP” or “Starbucks” it must be exact. These are features that we would like to improve on in the future.

6. Individual Contribution

Meerza Ahmed, mhahmed@usc.edu	Data collection, Firebase Implementation, Data Modeling (ER / EER), Typescript Calls
Melissa Perkins, perkinsm@usc.edu	Data Visualization, UI Design, Firebase Implementation, Website Component Implementation
Alan Trinh, alantrin@usc.edu	Data collection, Data Cleaning, Python queries, CLI Implementation

7. Conclusion

In conclusion, we successfully implemented a website where users can search for potential study spaces stored in our Firebase Realtime Database. This website has features for data managers and front-end users and navigates a large dataset. We believe this could be a valuable tool for students to assist them in finding a study space that works that caters to their study needs, but additional features might make it a more effective tool.

Some key differences between our planned and actual implementation included not utilizing the picture dataset and not implementing a sorting feature for businesses on the webpage. Unfortunately, the dataset with pictures was a very large file, and we were concerned that we would exceed the Firebase free storage quota. In place of the photo dataset, we decided to use a generic set of photos for the study spaces, which saved us storage space. In other words, a cafe would have a generic coffee picture, and a pizza shop would have a pizza photo. We could implement a filtering feature on the web page, but we needed more time to implement the sorting. This would be a feature, along with many others, that we would like to include in the future.

8. Future Scope

As previously mentioned, the webpage can display business information for businesses that match the user's search criteria. With more time, we believe certain features could be added to this project to make it more user-friendly, efficient, and comparable to the YELP website. A critical component that we would like to improve is the search function. It is currently limited because the name must be exact, and the zip codes do not show businesses in the surrounding area. A map would have also been helpful to show the business locations relevant to the users location or within a radius.

Additionally, a user profile would be helpful to give users an option to save locations, add reviews, and add new attributes. One attribute we would want to see on an application like this is outlet availability. Most assignments are done using laptops or iPads and if there is no access to an outlet, that study time will likely be interrupted. This is an amenity that can often be found in libraries but may not be available in cafes or restaurants. Furthermore, additional attribute filters might be helpful for users to find spaces that fit their needs, such as filters on type of study space (cafe, restaurant, library, bookstore) or "Noise Level."

For the backend, we decided to use a simple hash function to reduce complexity and still achieve appropriate data distribution for two databases. We believed that only two databases were needed but could have expanded our hash function to include more databases for the future.

For our data managers, we planned on implementing a search by category feature to allow for data managers to search by categories such as "italian", "restaurant", "coffee". We had planned code for this feature but could not fully implement it since our database had categories that are nested with "0","1","2", etc. To fix this we would have to redo part of the data cleaning phase to have the categories nested correctly.