Lion Buddy: A Meal Buddy Pairing System

Frances Cao, Hanxiao Lu, Liping Hu, Yixing Wang
Computer Science, Electrical Engineering
Columbia University
New York, USA
COMS 6998 Cloud Computing Spring 2022

I. Project proposal

Motivation Behind Project

Many students find it strange or awkward to dine out at a restaurant alone. They may be labeled as someone who does not have any friends, and they may simply not want to be at a restaurant alone. There are often restaurants around campus that students want to try out badly, but it's difficult to find someone else to join. Even if students do not find it strange to dine out at a restaurant alone, perhaps the restaurant serves large portions of food, or they want to try more than one dish. Having someone tag along would be perfect so they can split the food with one another. Most importantly, this gives students a chance to bond and make new friends with similar interests. Best of all, finding a meal buddy does not equal dating!

Description of Project

Our application aims to solve the above mentioned problems. **LionBuddy** is a platform that helps Columbia students find meal buddies. It will allow students to fill out their personal profile, and get matched to other students with similar interests, hobbies, or majors on demand.

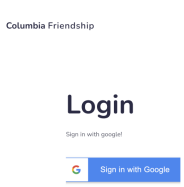
Some features of our application include the following:

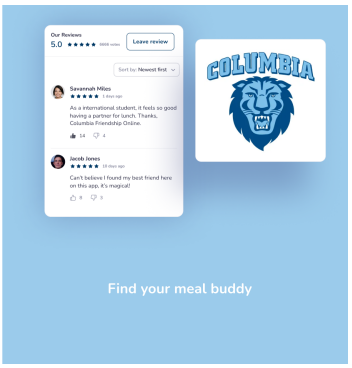
- LionMail authentication (Google API SSO)
- Match Tab that allows students to find a similar match.
- Restaurant Tab that allows students to rate restaurants (Later to be used for restaurant recommendations).
- Request confirmation page that allows students to approve or reject another student's request.
- Notifications sent to the user about their requests and match approvals via email.

II. Project Prototype

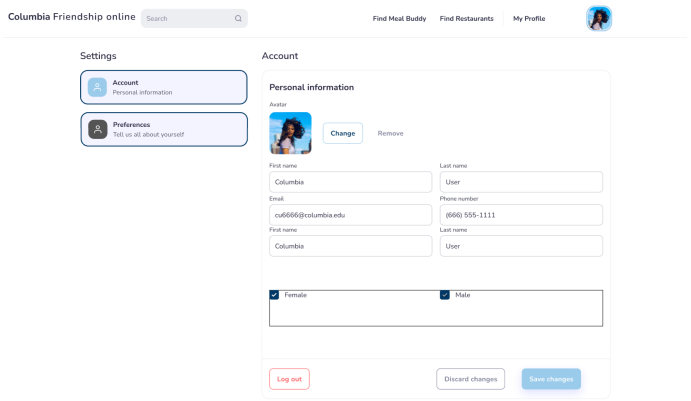
https://www.figma.com/proto/FkDq6ni9Bhzd9Lt39GPqsl/6998-Meal-Date

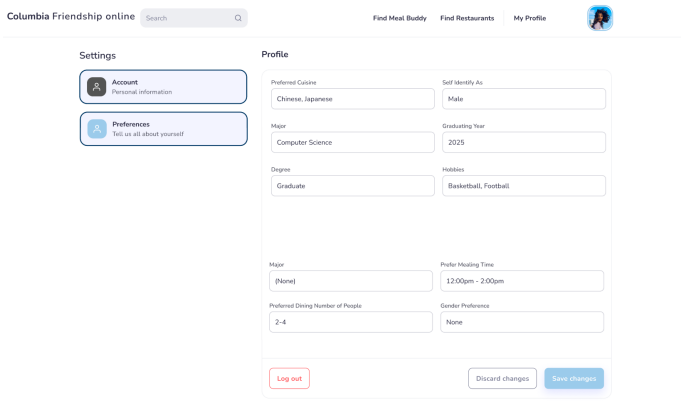
Login Page



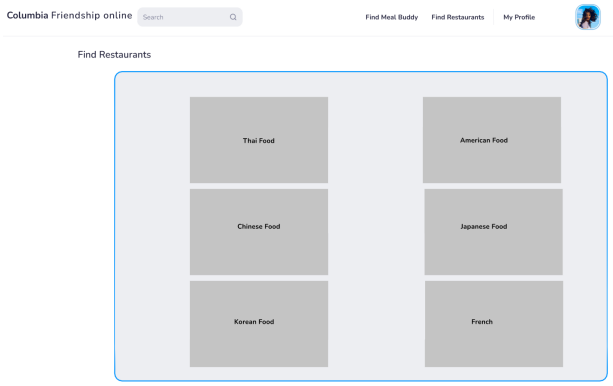


Profile Pages





Restaurant Page



III. Data Model

User Information

email	age	cuisine	degree	firstname	gender	graduating_year	hobby

img_url	lastname	major	nationality	phoneNumber	prefer_gender

Restaurant Information

business_id	cuisine	address1	city	coordinates	display_address	state	zip_code

image_url	insertedAtTimestamp	name	phone	rating	restaurant_url	review_count

Restaurant Like History

business_email	business_id	email	date_liked

IV. API DESIGN

GET /users/getuser
Parameters: email
Description: Get user profile data from DynamoDB users database.

POST /users/adduser
Parameters: None
Description: Add user profile to DynamoDB users database.

PUT /users/updateuser
Parameters: email
Description: Update an existing user's profile in DynamoDB users database.

PUT /users/imguser/{bucket}/{filename}
Parameters: None
Description: Upload a user's profile picture to S3 bucket storing user images.

GET /users/matchuser
Parameters: email
Description: Run the user through our Match model and retrieve data from DynamoDB user database.

GET /restaurant/{cuisine}
Parameters: numrest
Description: Run a user through our Restaurant model and retrieve data about the restaurant from DynamoDB restaurant database.

POST /respemail
Parameters: emailbody, recipientemail, recipientname, sourceemail, sourcename
Description: Triggers a Lambda function that uses SES to send an email to the recipient.

POST /command_restaurant
Parameters: business_id, date_liked, email, user_feeling
Description: Sends user like history data to DynamoDB like history database.

Implementation Detail

Restaurant recommendation is invoked through GET /restaurant/{cuisine} from API Gateway. The {cuisine} could be overridden by the cuisine that the user would like to search for. The header contains 'numrest', which indicates the number of restaurants to be returned to the frontend. API Gateway invokes the Lambda function in the backend. The Lambda function first retrieves the restaurants' business_id from OpenSearch according to the cuisine type. Then we use the business_id to get more information, such as rating, review count, and valid address, from DynamoDB. Finally, sorting and returning the restaurants' list to the FrontEnd according to the users' background and rating history.

Send invitation email is invoked through POST /respemail on API Gateway. The headers contain recipientemail, recipientname, sourceemail and sourcename which are used to configure the email. The body contains an email body. The backend Lambda function utilizes SES with information transferred from FrontEnd to send email to their interested buddy.

For the image upload function, we rename the image with the user email. Since S3 does not provide renaming for an object, what we do is firstly copy the image and upload it with a new name, and then delete the old one.

As for user matching, we mainly consider matching on the user's preferred cuisine matches the other user's preferred cuisine; user's preferred gender buddy matches the other user's gender;user's gendermatches the other user's preferred gender buddy. Once that filtering is complete, we further process it with our recommendation model. After that, we get the top 10 results. To avoid the user himself being chosen, we still check the emails before sending them to DynamoDB. We get all the data from DynamoDB and process the numeric data with min-max standardizer and the discrete data with one-hot encoder, then choose the top-3 users with the highest cosine similarity with the current user as the recommendation result and send it to the current user to choose male buddy from.

V. Architecture Diagram

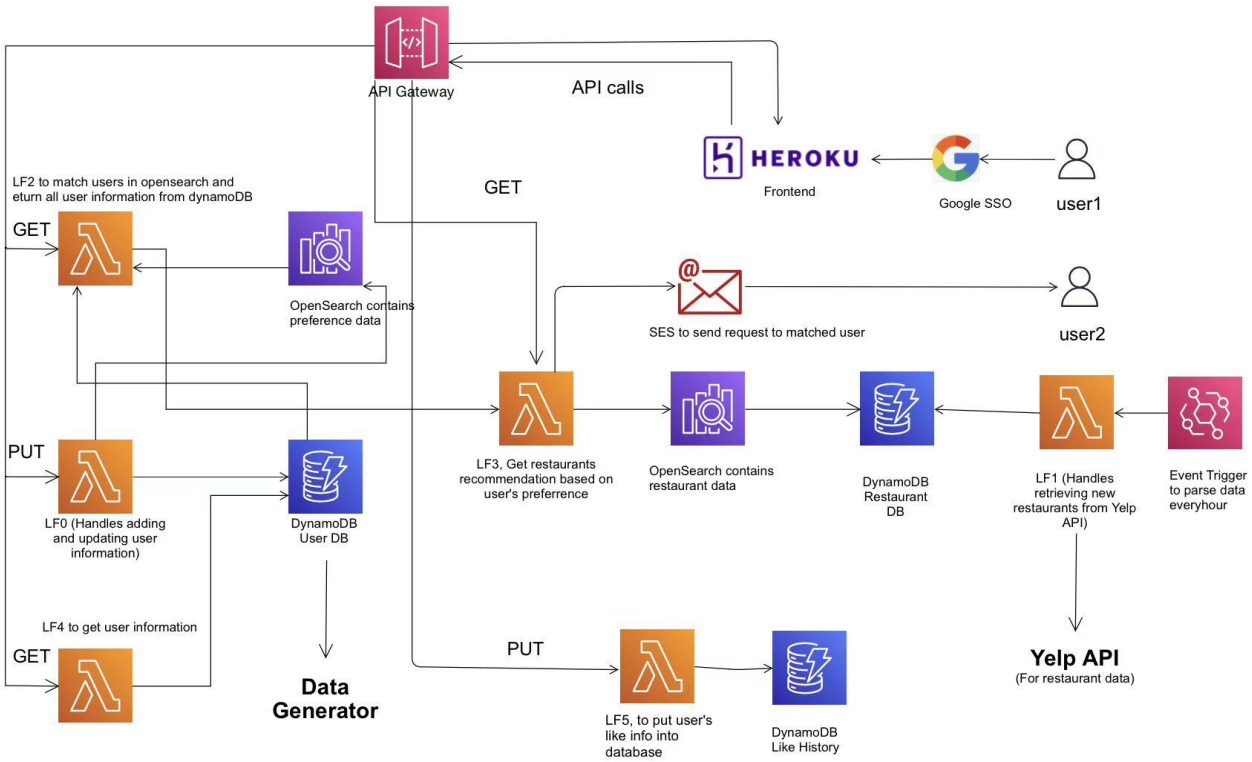


Fig. 1. System Architecture

VI. Architecture Flow

Frontend and Sign In

We use Heroku to host our FrontEnd, and Google API for single sign-on to our application. When a user signs in with their Google email address, our application sends a callback request to get an authorization response from Google and the user's data is stored in the session. If a user successfully

passes authentication, they are redirected to their user profile.

We maintain the user's token returned from Google in the session while the user is signed in. We also destroy the session details once the user hits Sign Out. The profile page will only show the authenticated user's profile page, and no one else can make changes to it.

User Data Update

When a user logs in, we send a GET request to the user database to get their data. If it is a new user, then when the user finishes filling profile data, we push their data to DynamoDB and also a part of the data related to matching phase, into Opensearch. If an existing user updates his or her profile, we also make changes to Opensearch and the database.

Moreover, the user can upload his or her profile image, we save the image in S3 and save the link to the image in DynamoDB.

User Match

When a user clicks on the frontend to request to find a meal buddy, we firstly search the matching user using features saved in OpenSearch and use our recommendation models based on cosine similarity(more details in Recommendation Model) to sort the results and send the top 3 emails to DynamoDB to get the users full profile.

Recommendation Model

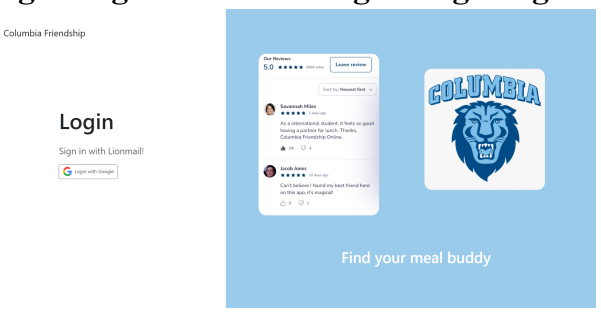
When a user requests to find potential buddies for the meal, the recommendation model utilizes the user's gender preference and matched history to give a list of suggestions. Here we mainly use a cosine similarity based method, since the cuisine and gender is already matched, we sort the result based on their age, hobbies, degree etc. We process the numeric data using min-max standardizer and the discrete data using a one-hot encoder. When a user requests to find potential restaurants, the recommendation model utilizes the users' background information and rating history to give a list of suggestions. A better way should be using content-based or user-based filtering, but due to lack of authentic data, we cannot simply rate the data randomly because the rating is related to the features of users and restaurants.

Scraping Data

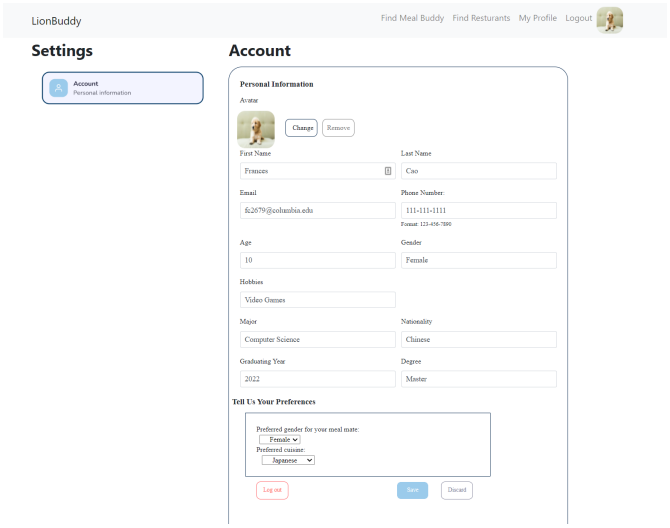
We scrape Yelp once every 24 hours for new restaurants added, we may update this to run weekly instead as there are not too many changes that we see entering our database and OpenSearch. We use EventBridge to trigger this for us. Next, we also created a mock data generator to create around 500 fake users for us, so we have some data to play with.

VII. Features Implemented

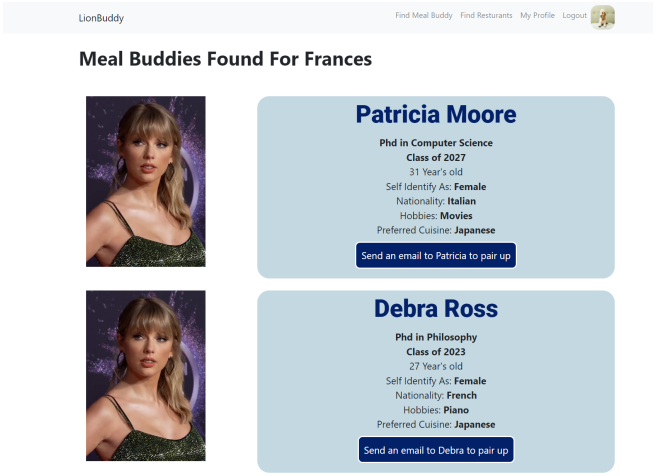
Login Page that uses Google Single Sign-On.



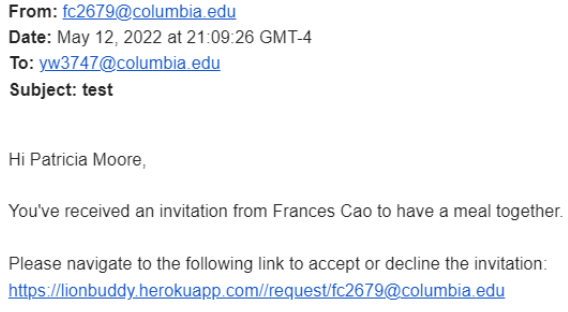
Edit Profile Page to add user details and interest, as well as update the default Google photo.



Match user page that uses the Match model to recommend users that scored the highest. Allows users to send an email to students they'd like to invite.



Email sent to the requested user.



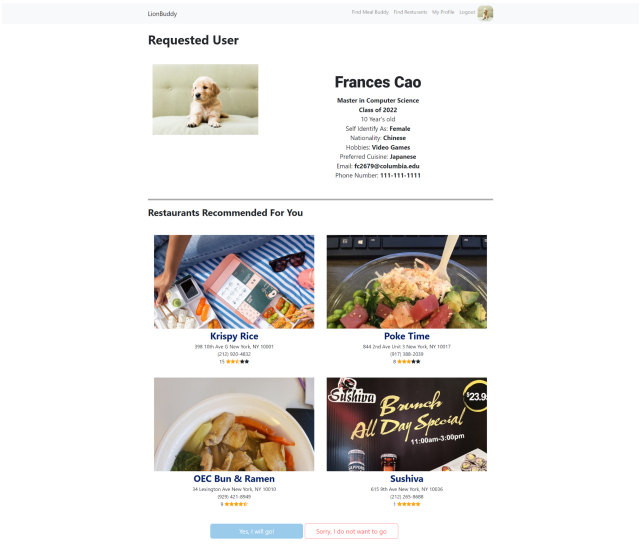
From: fc2679@columbia.edu
Date: May 12, 2022 at 21:09:26 GMT-4
To: yw3747@columbia.edu
Subject: test

Hi Patricia Moore,

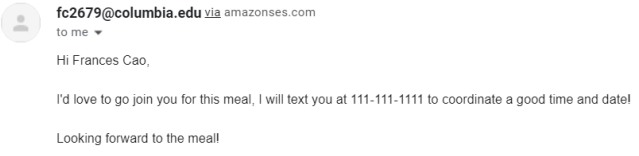
You've received an invitation from Frances Cao to have a meal together.

Please navigate to the following link to accept or decline the invitation:
https://lionbuddy.herokuapp.com//request/fc2679@columbia.edu

Requested user details page that also includes restaurant recommendations from Restaurant Model that matches the user's like history and cuisine type. Also two buttons to either accept or decline the user's invitation.



Response email after a user accepts or rejects the requested invitation



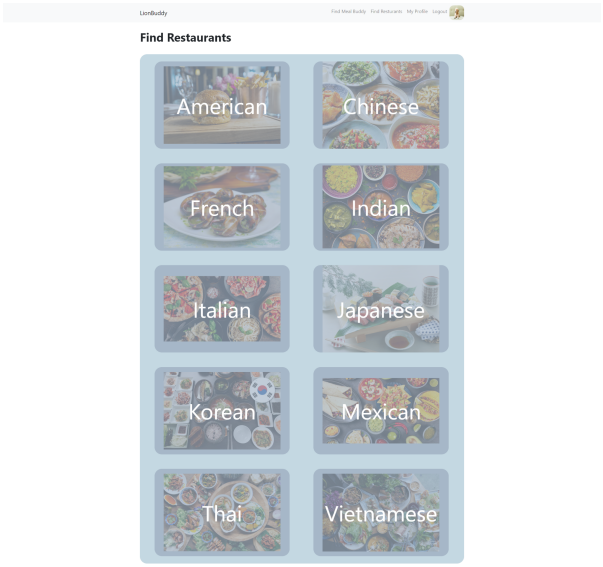
fc2679@columbia.edu via amazonses.com
to me

Hi Frances Cao,

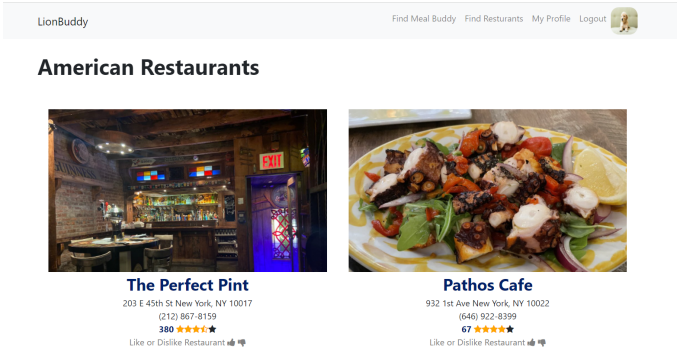
I'd love to go join you for this meal, I will text you at 111-111-1111 to coordinate a good time and date!

Looking forward to the meal!

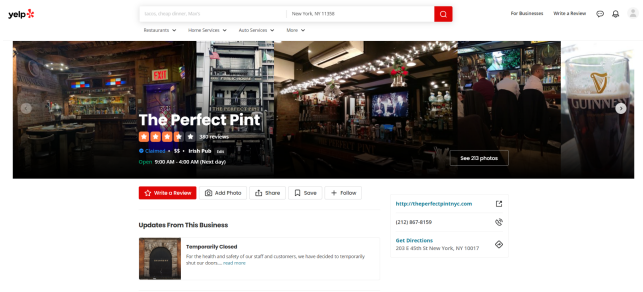
Finding Restaurant page by cuisine type.



Restaurant detail page that lists some basic information about the restaurant, as well as an option to like or dislike the restaurant.



Redirection to Yelp website if a restaurant is clicked.



VIII. Conclusion

The decrease in human interaction due to the pandemic has contributed to the decrease of friends made by students on campus. Social interaction is imperative to maintaining study-life balance and protecting overall mental health. Our LionBuddy app aims at providing Columbia students a safe and convenient opportunity to get connected with each other.

To summarize the main components of the app, we have deployed multiple AWS services including but not limited to S3 bucket, API gateway, Opensearch, DynamoDB and Lambda function. Heroku is used for deploying the frontend so that it's public to everyone. HTTP request and response messages were used for frontend and backend communication. First, HTTP GET method is used to get user profile data from DynamoDB users database and retrieve data about the restaurant. Second, POST method is used to add user profile data to users database, to trigger a Lambda function to send an email to the recipient and send user like history data to DynamoDB to history database. Third, GET method takes various information from databases. We built 3 dynamo databases to store 3 types of data, the user profile data, restaurant data and user like history. In terms of restaurant data, we utilized Opensearch to help store restaurant ids based on the cuisine type collected from the frontend. This way, the backend can search through OpenSearch to get suggestions of restaurants and then find more information by querying the DynamoDB database using the ids gained from Opensearch. Yelp API is utilized for scraping restaurants ids and other information through Yelp database.

In conclusion, more features can still be included in our design. Factors such as time and location wern't being considered because of the time constraints.

IX. Future Scope

Student location Component

Prof. Sahu brought up a great point about adding location functionality. What we can do is use the Google Map API to get the current location of the user, and add that value to our matching algorithm. Students closer to their potential matches distance wise would have a higher score compared to someone who is further away.

Time Matching Feature

Originally in our prototype but cut out due to time constraints. This feature will take into account a user's preferred dining time. We will add an additional step that checks whether the dining times are within the allotted time for each of the matches.

Restaurant Matching Feature

Currently we recommend the restaurants to the user based on the recommendation model. However, it is possible that students may just want to try a restaurant. In that case, they may not care who joins them for the meal, and would just like someone who shares a similar taste or desire to dine at the same restaurant. We would add an additional tab to support this feature.

X. Important Links

FrontEnd Url: https://lionbuddy.herokuapp.com/

FrontEnd Repo: https://github.com/FranCao/lionbuddy

Presentation Deck Video: https://youtu.be/4UutoWAEvQQ

Presentation Slides: PDF version here

Live Demo Video: https://youtu.be/w5NRRjYBUEA