Teamder, A Teammate Finding Platform

Edward Zhou[1], Yangjinan Hu[1], Yuxuan Luo[1], and Yuncong Yang[1]

[1]The Fu Foundation School of Engineering and Applied Science, Columbia University
{*xz3014,yh3273,yl4524,yy3035*}*@columbia.edu*

Abstract

A successful project require people with diverse skill set and close collaboration between team mates. However, finding a team mate that match your skills can be arduous task. That's why we build Teamder, a teammate finding platform designed for Columbia University students to find group project teammates based on their programming languages and skill sets. We employed a Tinder-like swiping card function that easily visualize classmates' profile and skills. User can swipe left or right to view classmates' profile in designated class. With our trained model, user can also get personalized recommendation of potential team mates that matches their skills. Our system is entirely built on AWS, with AWS amplify to host our react frontend, API gateway to handle requests to Lambda, Sagemaker endpoint to host our trained model that gives recommendations and SES to send out emails. It's highly scalable, reliable and fast. Users get seamless transition between cards and a recommendation is delivered within 30 seconds.

1 Introduction

There are a lot of group projects for students, and forming an ideal team could be a cumbersome task in that 1) students have different programming language preferences as well as various skill sets, and 2) group projects often require a combination of programming skills. Successful project need students from different backgrounds, hence motivating the birth of our platform.

Teamder is a Tinder-like App for forming teams for course projects. Teamder users could view other students' skills and profile in the same course and determine whose got the most suitable skill sets with them. The core of our platform is that we use recommendation algorithm to search inside our user database to rank and display recommended teammates for students. In later sections we are going to discuss further about our algorithm.

2 User Interface

As part of the technical documentation for Teamder application, this section provides a step-by-step walk through of the user interface as well as the technical details of the frontend. Teamder's frontend is the point of human-computer interaction and communication for our application. We want to ensure scalable, easy access, high performance and across platform usability while using easy to develop tools and keeping maintenance cost low. Naturally, this led to our decision to build a web application using React and host it on AWS Amplify. AWS Amplify is a series of purpose-built tools and functions that allows developer to quickly built full-stack web application using AWS services. With AWS amplify, we can easily integrate AWS services into or web applications such as cognito, lambda, API gateway etc. In terms of navigation between pages, we used react router and for interaction between frontend and backend we used Axios.

2.1 Signup/Login Page

As shown in figure 1 and figure 2 The first time you enter the page for Teamder webapp, the app will prompt you to create an account with your phone number and email. After successful creation of your account, you can switch to sign in page and sign in with your phone number and password.

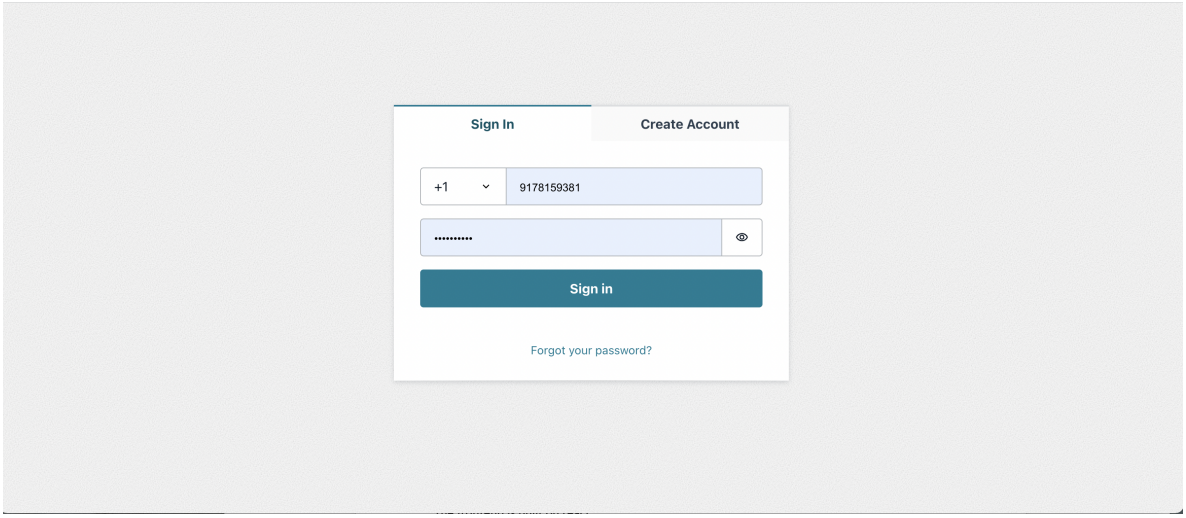

Figure 1: Login

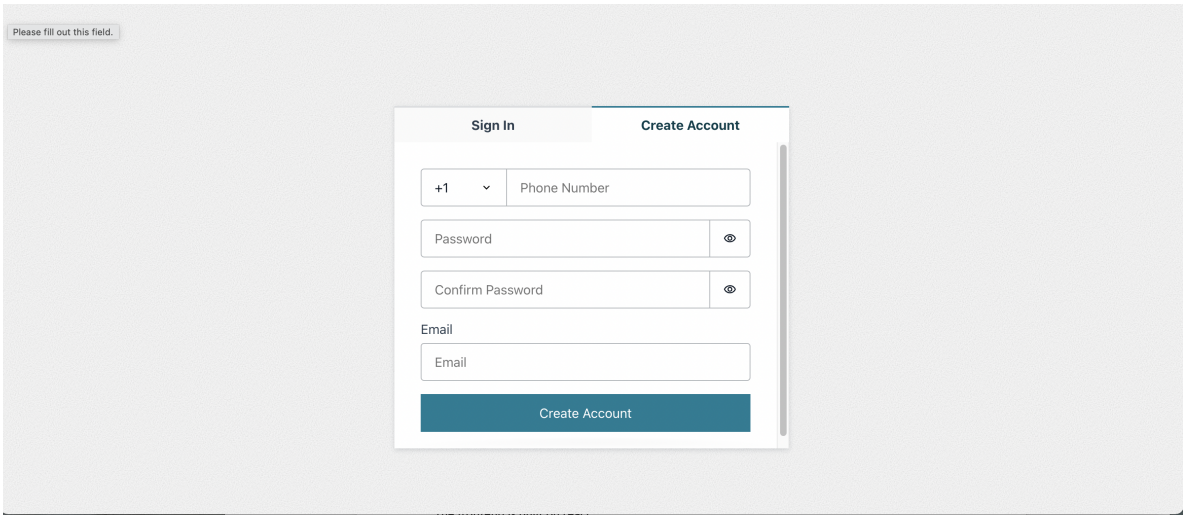

Figure 2: Signup

2.2 Home Page

As shown in figure 3, the home page contains three components, the header, swipes and the buttons. The header contains three clickable buttons: profile button, teamder button and the chat button. If you click the profile button, it navigates to the profile page. If you click teamder button, you navigate to home page, so if you are in any other page, you can click teamder button so you can go back to home page. The button located on the right end of the header is the chat button. Clicking it allow you to navigate to the chat page.

The swipe card is at the center of the home page. The swipe card displays the classmates' profile picture, profile name, skills and skill level. You can either swipe left or swipe right to view classmate's profile and skills. We employed the Tinder-like swiping frontend for two reasons. The First is that this function is widely known, users can quickly get familiar with the swiping as a way to view classmates' profile. The second reason is that this is easy to implement. There is a ready-to-use package: Tinder Card that already handles all the complicated logic of swiping left and swiping right. The swipe cards connect to a database in firestore where it contains all the students' data in a class.

There are five buttons at the bottom of the home page, three of them have specific functions. At the left end, the button with a add and minus sign let users navigate to Pool Interaction page. To the left of the button, the button with a pen on it let users navigate to Put user info page. The button

with a download sign let users navigate to get user info page.

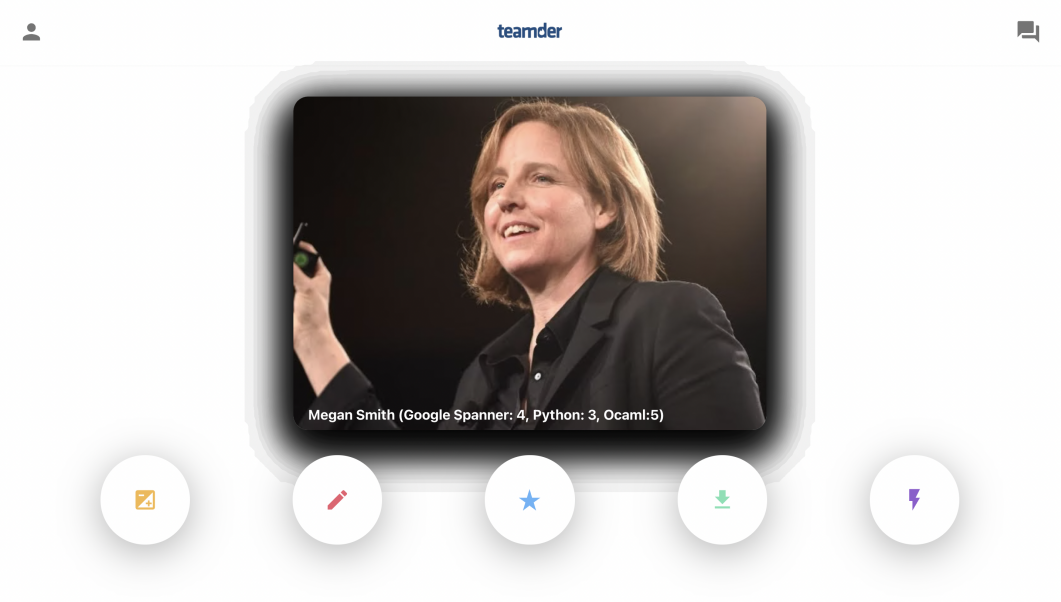

Figure 3: Homepage

2.3 Chats/Chat page

As shown in figure 4 and figure 5, the chats page contains the header, individual chat. Clicking the area of each chat navigates to the individual chat where you can view all the messages with the matched person and send message to them. When users are in the Chats page, the profile button located on the left end of the header automatically turns into back button. Clicking the back button, user can return to the home page.

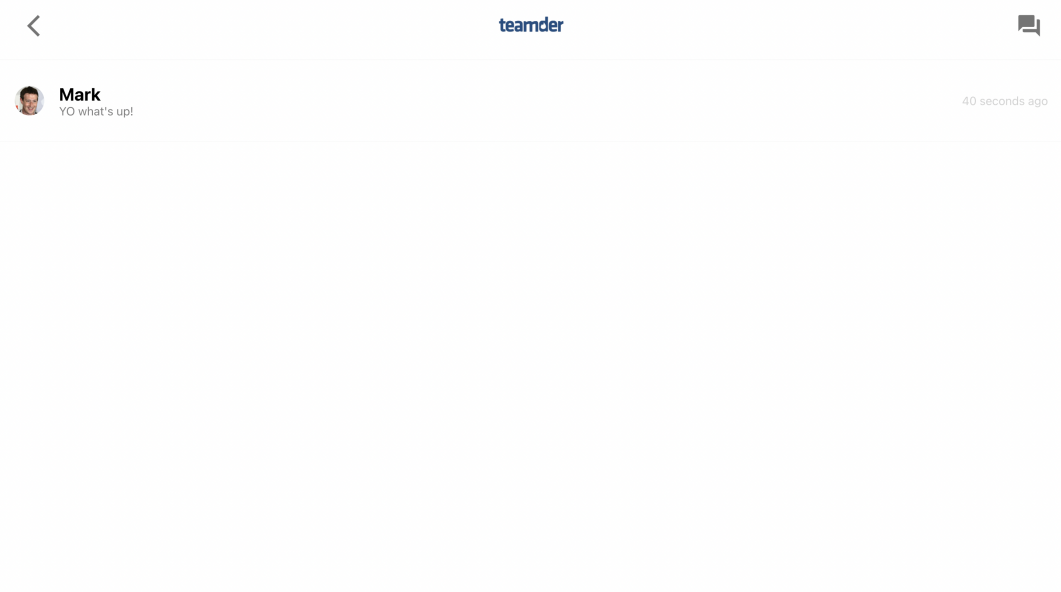

Figure 4: Chats

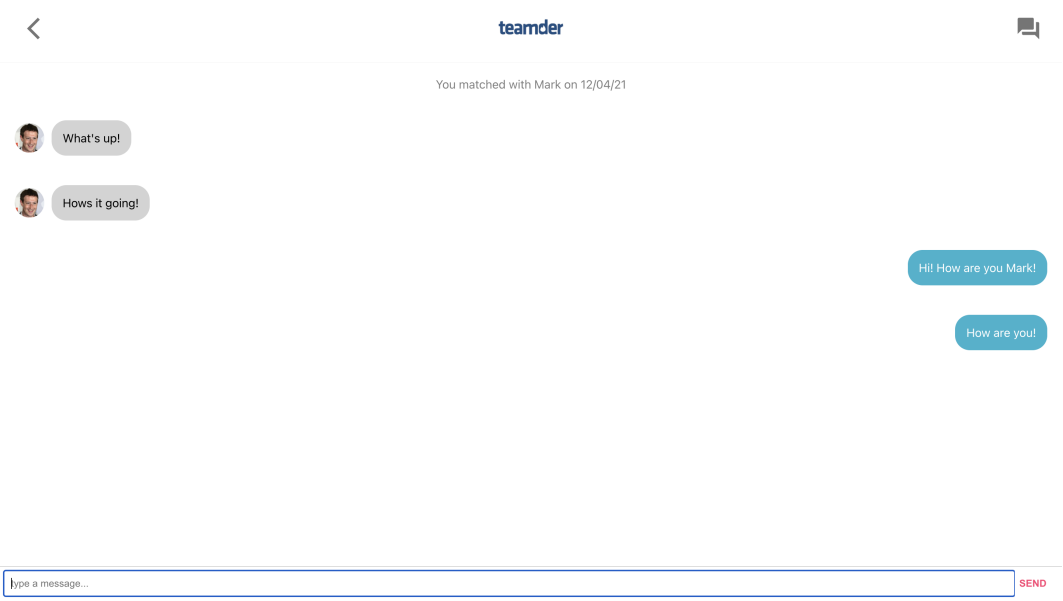

Figure 5: Chat

2.4 Get/Put User Info function and Pool interaction Page

As shown in figure 6, Get User Info function contains one button, clicking it allow you to view the skills and skill level that you put in the system. It retrieves the token generated in a session in AWS cognito and sends it to API Gateway and then to the Lambda function where it can query the relational database and fetch the user data and send it back to the frontend. As shown in figure 7, Put User Info function contains a list where it asks you about your name, email, online/offline status, Uni and skill level from 1-5. Once you click the submit button, the information is sent to the backend and handled by a Lambda function and stored in a AWS RDS. As shown in figure 8, User Pool Interaction allows a user to be added or deleted in a course. User can enter Pool Action (Put or Delete), course_id and email. Once user clicks the submit button, the information is sent to the backend and handled by a Lambda function.

Figure 6: Get User Info

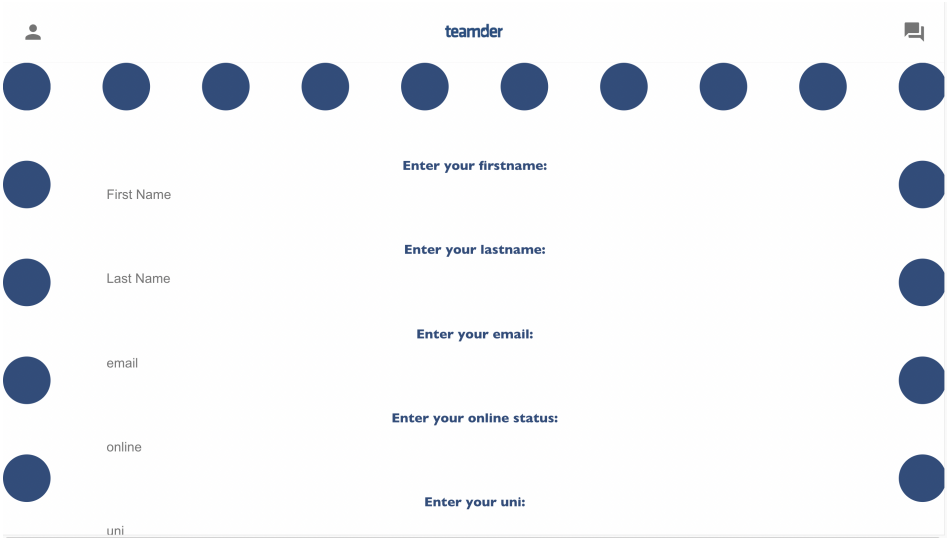

Figure 7: Put User Info

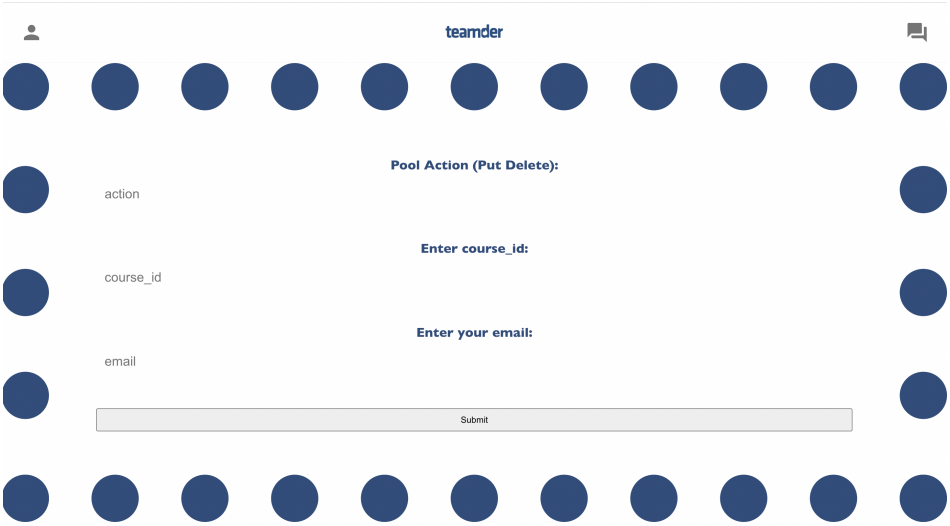

Figure 8: Pool Interaction

3 System Architecture

3.1 APIs

There are three APIs that's exposed to the frontend: /PutUserInfo, /GetUserInfo, and /PoolInteraction. We use axios for frontend and backend interaction. /PutUseInfo supports post method. We include user input information in the payload of a post method and send it to this endpoint. /GetUserInfo supports post method. We include user session token generated by AWS cognito in the request payload and retrieve the information user put in the RDS. /PoolInteratction supports Post and Delete method. User can specify methods they want to use in the form and enter the class number and their email to be added to or deleted from a course. Details of how to interact with the three APIs are included in the architecture flow section.

3.2 Architecture Diagram

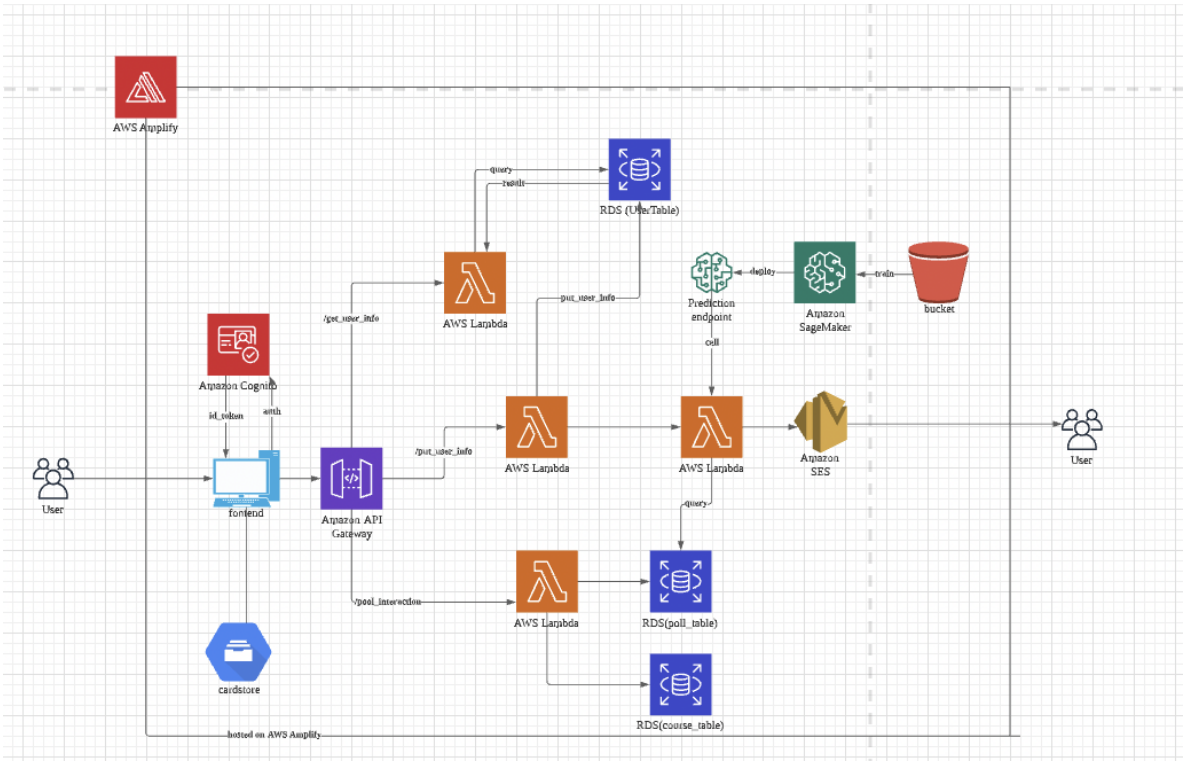

Figure 9: Teamder System Architecture.

3.3 Architecture Flow

All Actions | Browser → *

Users interact with Teamder's services through their browsers. All user sessions begin with a sign in or sign up process through a portal that is hosted by AWS Cognito. After the completion of this process, the browser saves the Access Token and ID Token for the user and takes the user to our frontend written in React and hosted on AWS Amplify.

Create/Update User Profile | Browser → API Gateway → λ → RDS

Our frontend uses Axios with React to create and submit REST API requests. The requests are sent to each respective endpoint hosted by API Gateway. For this operation, the resources this action interacts with is `PUT /user_info`. API Gateway triggers a Lambda function and forward the `PUT`'s payload to this Lambda function. The Lambda function first checks the user's identity and extract their email address from the ID token. The Lambda function then establishes a connection to the database instance running on RDS, and create or update the entry on the `USER` table with the information provided in the payload by issuing a SQL query.

Get User Profile | Browser → API Gateway → λ → RDS

The "Get User Profile" flow is very similar to the flow outlined above. With the SQL query changed to a `SELECT` from `INSERT`.

Joining/Leaving Matching Pool | Browser → API Gateway → λ → SQS

The frontend submit a `PUT` REST request to the API Gateway endpoint on `/pool_interaction` with the user's ID token, a course ID selected by the user on the frontend, as well as a boolean variable to

either join or leave the matching pool for that course. Once again, the API Gateway triggers a Lambda function that verifies the user's identity and adds or removes an entry of the `(email, course_ID)` pair to the `POOL` table in the database.

Finally, if the user is joining the matching pool. The Lambda function leaves a message on an SQS queue with `(email, course_ID)` as the message.

Finding Matches | SQS $\rightarrow \lambda \rightarrow$ RDS $\rightarrow$ SageMaker $\rightarrow$ SES

Messages left in the SQS queues are processed asynchronously. SQS triggers a Lambda function with `(email, course_ID)` as the payload. This Lambda function fetches the classmates of this user and their skills by joining the `POOL` table and the `USER` table. The user's and their classmates' profiles are passed into the SageMaker endpoint where our XGBoost model gives a compatibility score for each classmate. The Lambda Function sorts the classmates by this score and sends an email to the user through SES with the top ten profiles.

4 Database

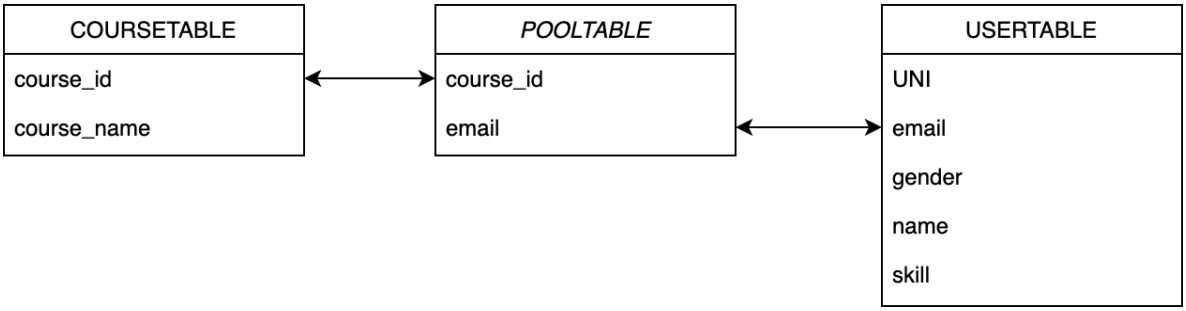

Figure 10: Database Schema.

Since our project requires sensitive user data that could be difficult to acquire from any database, we randomly generated fake user information (uni, email, gender, name) along with their skills.

uni	email	gender	first name	last name	skills
xz3014	xz3014@columbia.edu	1	Edward	Zhou	{"Networks": 3, "FrontEnd": 4, "C++": 3}

couse id	course name
COMS 6998	Cloud Computing

user email	course id
xz3014@columbia.edu	COMS 6998

5 Recommendation System

Since we did not find any labeled data on the internet, we had to adopt the most arduous way. We manually labeled over 2000 samples of randomly generated data to train our recommendation model offline. The trained machine learning model is able to assign scores to all the other users in the same user pool and then rank them. The information of the other users with top 10 scores would be sent to the user by email as recommendations.

3128	('Database', 4)	('Java', 1)	('Python', 5)	('NLP', 5)		
3129	('C++', 4)	('Database', 3)	('Machine Learning', 3)			0
3130						
3131	('JavaScript', 4)	('NLP', 5)	('C++', 5)	('Java', 3)	('BackEnd', 1)	
3132	('C++', 3)	('Security', 3)	('Python', 5)	('FrontEnd', 2)	('Java', 4)	1
3133						
3134	('Computer Vision	('FrontEnd', 5)	('JavaScript', 4)			
3135	('Latex', 2)	('Database', 2)	('C++', 4)	('Java', 4)		0

Figure 11: Data samples.

5.1 Data

For each data sample, we randomly generated two skill sets to represent two users. During the labeling process, the annotator would assume that he is the first user and he would decide whether he would like to match with the second user. Figure 11 is an illustration of our data samples.

5.1.1 Why structured data?

We thought about using unstructured data at first. However, we found that the only ways of making use of the unstructured data is through user's profile picture or profile description. Since teamder should focus more on academic or working skills instead of finding dating matches, we believe that profile image or description cannot help much, while they takes great efforts. We did not find any labeled data for such course project matching recommendation systems, which means that we have to manually collect and label all the data to a decent amount of them. Therefore, we believe that using structured data only is more reasonable in a course project scale.

5.1.2 Data generation rules

We enumerated 13 different skills as our skills enum in database. The skills includes commonly used programming languages such as "Java" and "Python" and broader "skill packages" like "Backend" and "Security". We believe that our 13 skills enumeration is good enough for course project matching. For each user, the data generator selects 3 to 5 different skills from the 13-skill enumeration and assign scores 1 to 5 for them (score 3 to 5 with higher probability distribution).

5.1.3 Manual Labeling

Our group members and friends helped with the manually labeling. Although we did not informed the annotators a golden standard during labeling, we found that the labeled data have a clear pattern. The first user tends to select the second user if they use the same programming languages, or they have complementary "skill packages", or the second user has a lot of skills with high scores. With manually labeled data, trained model would mock the behavior or users.

5.2 Model

We tried several different machine learning models on our data. Figure 12 shows the results we obtained after tuning the hyperparameters. Therefore, we picked the trained XGboost model as our recommendation model. We deployed it on our sagemaker endpoint. The endpoint helps predict each users' recommended matches.

5.3 Recommendation Results

After recommendation results is obtained via XGboost, the top results are passed into Amazon SES and sent to users email extracted from RDS USERTABLE. In the meantime, the recommendation results are passed to the front end webpage as a list, and users would have a higher opportunity of swiping right and selecting their teammates more efficiently.

	Acc
XGboost	0.91
Random Forest	0.86
Naive Bayes	0.73
KNN	0.73

Figure 12: Accuracy of different models on our manually labeled data.

6 Summary

We summarize our implemented feature in this section. In the frontend, we utilized AWS amplify to host our react web app. Using AWS amplify, we Incorporated AWS cognito for user sign-up or authentication through external identity providers to provides temporary security credentials to access Teamder's backend resources in AWS. As for the user interface, we built customized pages for each of the API endpoints so user can easily get user information, put user information to receive recommendation and add to or delete from a course. We also implemented a swiping card function where user can view all their classmate's profile and skills.

In the backend, we constructed structured training data and used them in recommendation model training. We conducted experiments comparing four different machine learning models: XGboost, Random Forest, Naive Bayes and KNN and found that XGboost gives superior performance. We deployed our XGboost-trained model in AWS sagemaker endpoint for invocation from Lambda. We constructed data-schema to store user profile, user skills and class information in AWS RDS. To exposed our API endpoints, we configured API gateway and implement four Lambda function to process requests for /GetUserInfo, /PutUserInfo, /PoolInteraction, invocation of model. Finally, we configured AWS SES to send recommendation results to the Users.

References

[1] Chen, T., He, T., Benesty, M., Khotilovich, V., Tang, Y., Cho, H. (2015). Xgboost: extreme gradient boosting. R package version 0.4-2, 1(4), 1-4.

[2] Guo, G., Wang, H., Bell, D., Bi, Y., Greer, K. (2003, November). KNN model-based approach in classification. In OTM Confederated International Conferences" On the Move to Meaningful Internet Systems" (pp. 986-996). Springer, Berlin, Heidelberg.

[3] Rish, I. (2001, August). An empirical study of the naive Bayes classifier. In IJCAI 2001 workshop on empirical methods in artificial intelligence (Vol. 3, No. 22, pp. 41-46).

[4] Biau, G., Scornet, E. (2016). A random forest guided tour. Test, 25(2), 197-227.