Socialize

Smrithi Prakash *(sp4135)*
Varsha Kanmuri *(vk2497)*
Soham Dandapath *(sd3596)*
Madhura Chatterjee *(mc5470)*

***Abstract*—This document is a report on the project for COMS 6998 Cloud Computing & Big Data at Columbia University (Fall'23). For our project, we have created an application called 'Socialize!', a platform which aims to make it easier for Columbia freshers to socialize with their peers. We offer users of our app the ability to explore ongoing events, meetups, and study groups. Additionally, our platform includes a sophisticated recommendation system, allowing users to receive personalized suggestions. Furthermore, users can engage in polling activities and easily view the number of participants interested in a particular event or meetup. In this document, we talk about the architecture, key design details, code structure, results, and conclusions of this project.**

I. Introduction

Usually students new to campus try to get to know new people by planning to meet somewhere and ask people to join on whatsapp groups. However, this could be a challenging task for some, while for others there could be a mismatch of common social or chatting platforms. A majority of students come to Columbia from far away and the entire experience of leaving home is a bittersweet experience for most. Lack of connections and a social circle could be detrimental to students' mental health. Our motivation for this application is to provide a secure platform for all students of Columbia University to connect with one another.

A. Existing Solutions and Limitations

Meeting new people is a well-established concept of forming bonds between humans with platforms like Meetup and Heylo already in existence. However, these platforms primarily concentrate on event coordination, often spotlighting event hosts and promotional efforts. While they offer opportunities to forge new connections, they may not fully cater to the specific needs of forming study groups, connecting with classmates, or simply fostering friendships within the Columbia community. Presently, our approach to forming teams for group projects relies on Edstem, while for study groups, connections are predominantly established through informal means such as word of mouth, whatsapp or chance encounters during lectures. This method may inadvertently curtail opportunities for broader interaction and collaboration within a larger community.

B. Novelty

In this paper, we present a novel platform for the students of Columbia students that will facilitate them to meet, attend networking events and form study groups. Our platform goes a step further by facilitating the planning of these gatherings through the use of polls. Additionally, it is exclusively accessible to Columbia students, fostering a heightened level of trust among participants.

The app allows students to create and participate in various **activities** like meetups, events and study groups.

1) Meetups: A meetup is an informal meeting arranged by any individual (Columbia student) where a group of students with similar interests meet to have casual conversations. Meetups can happen at a cafe, restaurant, on campus, virtually, or any other location in New York City. The app allows students to conduct polls to get to know other's preferences of timings, places, food, interests, etc. The app also has an AI-based recommendation feature that recommends locations in NYC based on user preferences/poll results. Based on poll results, individual students can create meetups and post them on the website for other students to join. Students can join any meetup posted on the portal.

2) Events: Events are formal meetings organized by the university or student organizations. Events generally take place on campus and typically have a purpose, such as networking events or career fairs. We no longer need to worry about missing out on events scattered across different platforms like Facebook, Instagram, WhatsApp, or emails; our app serves as the one-stop destination for all your event announcements and coordination needs. Student clubs and organizations can post events, too. All Columbia students can sign up for any event published on the portal.

3) Study Groups: This feature helps students form study groups for courses. Most courses require students to form groups for projects. A student enrolled in a particular course can create a study group and make it available for other interested students. The app allows students to create polls to get to know students' availability for weekly study meetings, group project discussions, assignment discussions, etc. The app also recommends suitable locations on campus to meet and study.

II. Features Description

A. User Profile Creation and Update

First time users have to first create their profile or register before logging in. Once a user logs in or registers, they can view, create or update their profile or preferences.

B. Homepage

The homepage serves as a central hub, offering a personalized experience for users. Divided into three sections, it presents upcoming activities with details on time, location,

and participants. This allows users to efficiently manage their schedules. Additionally, the homepage displays activities created by the user, enabling easy access to their organized events, meetups, or study groups. The third section highlights polls created by the user, fostering engagement and decision-making within the community. This comprehensive view enhances user engagement by providing a holistic overview of their activities and contributions.

C. Activity Creation

The Activity Creation feature empowers users to create their own events, meetups, or study groups effortlessly. Users can input essential details such as titles, descriptions, locations, and categories seamlessly during the creation process. This feature empowers users to take charge of the event creation process, cultivating a dynamic and user-driven platform for organizing diverse activities.

D. Activity Registration

Activity Registration simplifies the process for users to express interest and participation in various activities. The straightforward registration process enables users to quickly and efficiently sign up for events, meetups, or study groups, fostering an inclusive environment. This feature enables users to effortlessly join a diverse range of activities based on their availability and interests.

E. Email Notification

The Email Notification feature ensures timely and personalized communication by leveraging AWS SES for reliable and efficient email notifications. Users receive personalized confirmation messages for successful activity registrations.

F. Activity Search Functionality

The Activity Search Functionality harnesses Elasticsearch's robust capabilities to provide a comprehensive search experience. Efficient textual searches based on activity title, description, location, and category of interest enable users to easily discover relevant activities. This implementation also incorporates fuzzy search, returning documents that contain terms similar to the search term, even when there isn't an exact match.

G. Recommendation using AWS Personalize

We use AWS Personalize to obtain recommendations for Meetups, Events, and Study Groups based on user activity history. AWS Personalize allows us to create highly personalized recommendations for users. By analyzing historical interactions, the service understands individual preferences and tailors recommendations accordingly.

H. Interest based Recommendation

To enhance interpretability in our recommendation system, we consider user preferences expressed as interests, such as "Community & Environment" and "Fitness". Users add tags to activities like meetups, events, or study groups during creation. Employing string-based filtering, similar to music streaming services asking for preferred genres and artists, we tailor recommendations to the user's specified interests. This approach ensures a personalized experience, aligning with the user's indicated preferences in various categories.

I. Polling

Our application integrates a polling feature which allows users to gather opinions and preferences of participating students regarding events, meetups or study groups. Users can seamlessly create polls to gauge the response of students for events, meetups or study groups. This makes it slightly easier for new students to attend events which they are interested in as well as has a high volume of participation. A participant can mark their interest for any event by clicking on the 'interested' button. A user can view the polls they created and their respective results in the homepage. In our application, we assume that the creator of the poll is by default interested in the event/meetup/study group.

J. User Authorization

The app ensures security by only allowing authenticated Columbia University students/affiliates to use the portal. We ensure this by allowing the users to register using only their Columbia Lionmail email. Exclusivity to Columbia students fosters a heightened level of trust among participants. To further enhance the security, a verification email is sent to the users to their Lionmail which redirects them to creating their profile.

III. Architecture

The architecture diagram is shown in Fig 1.

IV. Key Design Details

A. Frontend

Employing React for our frontend empowers us with component reusability, a pivotal asset in our design strategy. This proves particularly advantageous for shared UI elements like Cards across Event, Meetup, and Study Group sections. React's capacity to declare and import packages further elevates usability; for instance, 'aws-api-gateway-client' streamlines API Gateway requests without the need to download the SDK for each route modification.

B. Web Hosting

Choosing AWS Amplify for deployment underscores our commitment to efficiency. Continuous deployment becomes a breeze, seamlessly updating our website with each repository commit. This ensures our platform remains agile and reflective of the latest changes.

C. API Gateway

User-backend interactions are channeled through the API Gateway, featuring diverse routes redirecting frontend requests to the corresponding Lambda functions. The API Gateway not only optimizes this flow but also brings forth essential features such as Cross-Origin Resource Sharing (CORS), access control, and request throttling, fortifying security and

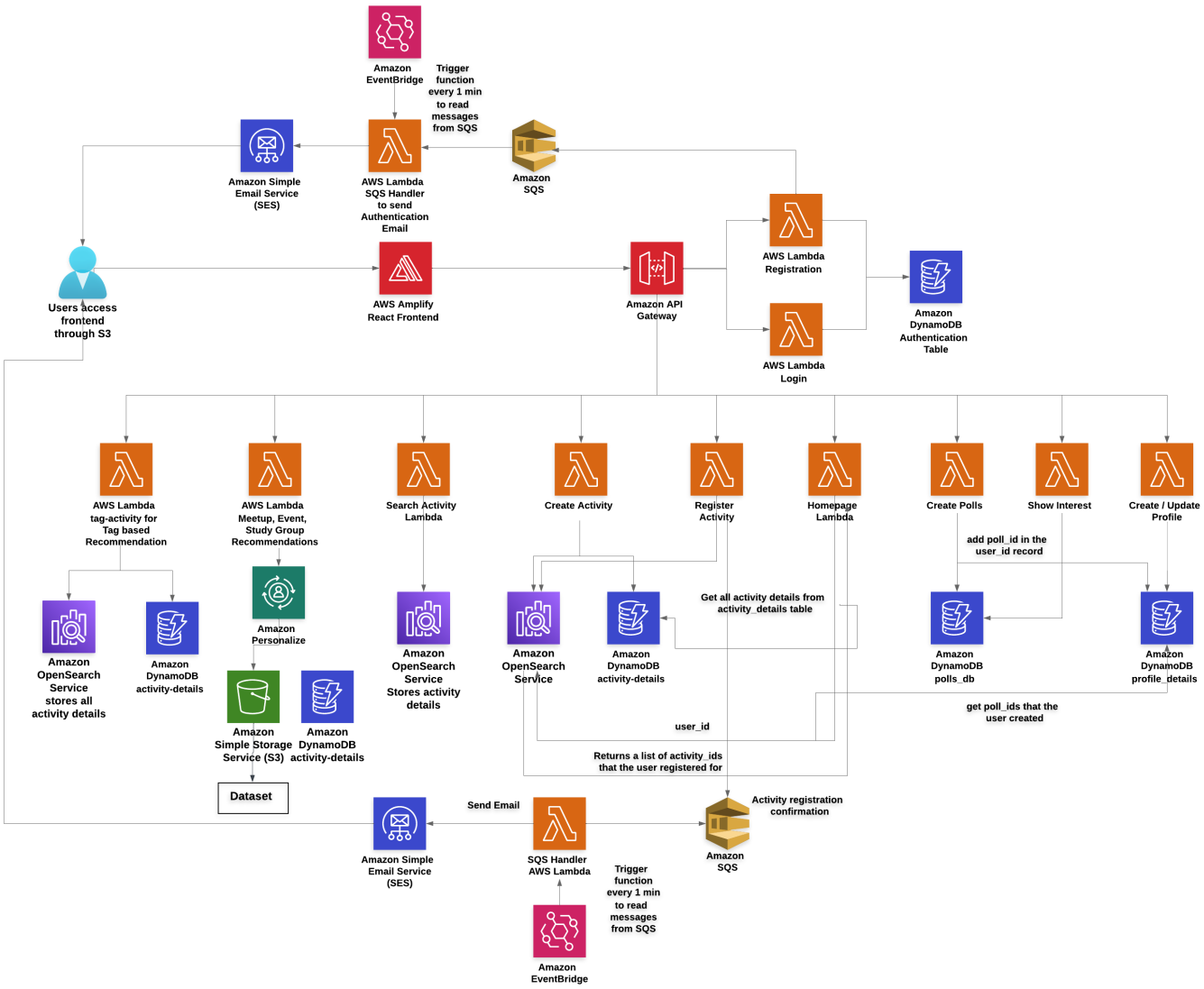


Fig. 1: Architecture Diagram

performance. The flexibility of the API Gateway extends to future scalability, allowing us to effortlessly expand infrastructure for planned features.

D. User Profile Creation and Update

For user profile creation and update, we use two lambda functions (profile-create, profile-update) and one dynamoDB table (profile-details). The DynamoDB table 'profile-details' stores the basic information about each user registered on our app. It contains the following fields: 'uni' (partition key), 'emailId', 'interest', 'location', 'name', 'phoneno' and 'poll_ids'. 'poll_ids' is a field which contains a list of all the poll id's of the polls the user has participated in. It is initialized to an empty list at the beginning. Profile-create is used to create a user profile for first time user. It creates a DynamoDB resource with the required fields and inserts the item into the table 'profile-details'. We allow the user to update their 'location', 'phoneno' or 'interest' using the 'profile-update' lambda function. This function updates the respective row in the profile-details table based on the user_id. We use another lambda function 'profile-view' which allows the user to simply view the profile details.

E. Homepage

To enhance the retrieval efficiency of activities registered or created by a user on the homepage, we leverage OpenSearch. Each record is uniquely identified by the user_id and contains fields corresponding to the meetup ids, event ids, and study group ids that the user has registered for (only the ids, while the actual details are stored in DynamoDB, as explained in the next section). Additionally, there is a dedicated field to store the ids of activities created by the user. This design choice capitalizes on the strengths of OpenSearch, eliminating the need for a DynamoDB scan and ensuring rapid retrieval of

a user's homepage details. OpenSearch consolidates all user-specific information in one location, simplifying the retrieval process and significantly improving the overall performance of the homepage.

F. Activity Creation and Registration

For storing activity details, DynamoDB serves as a robust solution, efficiently capturing essential information within its schema. Vital fields, including the unique activity identifier (activity_id), timestamp, an initially empty list for attendees (dynamically populated), the activity's category (e.g., Meetup), creator's ID, date and time, description, location, tags, and title, are seamlessly accommodated. Simultaneously, OpenSearch is dynamically updated to include the ID of each new activity the user registers for (as mentioned in the previous section). This integration ensures the smooth retrieval and display of the user registrations and creations on the homepage. The combined utilization of DynamoDB and OpenSearch reflects a strategic design choice, effectively optimizing data organization and retrieval efficiency for both activity creation and user registration facets of the platform.

G. Email Notification

Upon successful registration, user details, including user_id and other relevant attributes are added to a SQS message queue. An EventBridge rule triggers a Lambda function every minute. The Lambda function then reads the message from the SQS queue, and fetches the user_email from the profile table. Fianlly, using SES, personalized confirmation email is swiftly dispatched to the user, confirming successful registration for the activity.

H. Activity Search Functionality

The implemented Activity Search Functionality utilizes OpenSearch's fast text search capabilities. The OpenSearch index is structured to contain crucial details such as title, description, location, and category, which is added during an activity creation.

In the Lambda function, a combination of must and should conditions is applied to perform a multi-match query. The fields used in the search query includes title, description, and location. This design ensures that even if only one word from the search text corresponds with any of these fields, the activity is considered a potential match. The must condition is implemented for the activity category, ensuring relevance to the specified category the user is interested in. Furthermore, a compelling aspect of our search logic is the incorporation of fuzzy search. This feature enhances the search experience by returning documents that contain terms similar to the search term, as measured by a Levenshtein edit distance.

I. Recommendation using AWS Personalize

1) Dataset Collection and Preprocessing: For training the AWS Personalize recommendation system we used the Nashville Meetup Network Dataset from Kaggle (Dataset Link). The dataset comprises 24591 users, 19307 events, and 126813 user event interactions. Each event is associated with tags. The tags considered includes the following: Community & Environment, Pets & Animals, Support, Career & Business, Fashion & Beauty, Tech, Health & Wellbeing, New Age & Spirituality, Singles, Arts & Culture, Book Clubs, Movements & Politics, Fitness, Games, Outdoors & Adventure, Socializing, Music, Education & Learning, Hobbies & Crafts, Sports & Recreation, Parents & Family, Food & Drink, Religion & Beliefs, LGBT, Sci-Fi & Fantasy, Language & Ethnic Identity, Writing, Photography, Dancing, Movies & Film, and Cars & Motorcycles.

Data preprocessing techniques were used to format the data according to the application requirements. For this purpose, we assigned activity categories (Meetup, Event, and Study Group) based on the tag values. Records with tag as Socializing were considered as Events. Records with tag as Education & Learning or Book Clubs were considered as Study Groups. All other records were considered as Meetups. Every activity was randomly assigned a timestamp and location.

2) Dataset Group, Solution, Campaign, Filtering: From the preprocessed dataset, three dataset files were compiled for users dataset, items dataset, and user item interactions dataset. The fields considered in the final dataset include item_id, description, tags/category, user_id, timestamp, and user's name. The dataset group consisted of three datasets and the corresponding data was imported from the dataset files upload to an S3 bucket. After the dataset group was created, we created an aws-user-personalization solution and a corresponding campaign. We also created three filters for the three activity categories, in order to give personalized recommendations for the three categories separately. Three separate API endpoints and lambda functions are used for obtaining the recommendations from AWS Personalize with appropriate filters.

J. Tag based Recommendation

To facilitate tag-based recommendations, the user's interests are retrieved and stored in the website's local storage upon login. While a user may indicate multiple interests, we randomly select one to fetch relevant recommendations, preventing screen overload. This chosen interest is passed as a header to the API Gateway, subsequently transmitted to the Lambda function "tag-activity." This Lambda function utilizes two data sources: (1) OpenSearch and (2) DynamoDB (activity-details). Initially, OpenSearch is queried to retrieve activity IDs, filtered by the user's interest and activity category (e.g., meetup, event). Next, the Lambda function fetches detailed information about these activity IDs from the DynamoDB table "activity-details." The comprehensive activity details are then returned to the frontend user, ensuring a tailored and manageable set of recommendations

K. Polling

We use five lambda functions (poll-create, polls-view-all, poll-participate, poll-view-one, polls-view) and a DynamoDB

table (polls-db) to implement the polling feature. The DynamoDB 'polls-db' consists of the following fields: 'poll_id' (partition key), 'category', 'description', 'participants_count', 'tags', 'title' and 'user_id' (global secondary index). The 'user_id' field consists of the user_id of the creator of the poll. Polls are created with the help of 'poll-create'. This function generates a unique id for the poll_id and inserts all the respective data from the event to the 'polls-db' database. The poll_id is also stored in the respective user's profile-db entry (as per our assumption that the creator of a poll is by default interested in it). The function 'polls-view-all' returns all the poll_id's except the ones created by the user. We use this to display to the user polls in which they can participate in. The function 'polls-view' queries poll-db using 'user_id' as global secondary index, outputs list of polls created by a user. We use this function to display to the user the polls that they have created and the respective results or updates in participant count. 'Poll-view-one' gets a specific poll based on the poll_id. 'Poll-participate' is the function that we use to get the participant count or the result of the polls. It adds poll_id of the poll to profile-details table, in which the user participated in. On participation, the 'participants_count' field in the polls-db table is incremented by 1.

V. Code Structure

The entire codebase for our project is accessible at github.com/Smrithi23/Socialize-AWS. The repository is organized into two main folders: "Frontend" and "Backend." The Frontend folder stores the React-based frontend code deployed on AWS Amplify. On the other hand, the Backend folder comprises a subfolder named "LambdaFunctions" and an API-gateway Swagger file in YAML format. Within the "LambdaFunctions" subfolder, code is logically grouped based on functionality. For instance, the "poll" subdirectory contains Lambda functions related to the polling feature, and similar categorization is applied to other subdirectories. This structured organization enhances code management and understanding, facilitating collaboration and development on both the frontend and backend components of our "Socialize!" project.

VI. Results

A. Search Results

For the search result, we created events, meetup and study groups and tried to retrieve them based on title, location and description. The search capability has the same functionality across all activities, hence we demonstrate the result under the Events tab.

Search based on Location: We queried the location, "New York City". The system returned us with result from all the events in New York collectively from different boroughs such as Manhattan, Queens. The results are shown in the Appendix A, Fig. [2a]

Search based on Title : For this query, we used a single unique title of an event, "Techspo". The correct result with one unique event was returned. The results are shown in the Appendix A, Fig. [2b]

B. Recommendation based on Interest

For the recommendation we created a user with an interest tag "Tech". This is specified by the user during the creation of the profile and can be updated in the profile section. Subsequently, we provide suggestion of the activity based on the specified interest under the "Because you are interested in Tech" section for the particular user. This increases interpretability of the recommendation explaining why the activity was suggested in the first place. Results are shown in the Appendix. A, Fig. [3]

C. AWS Personalize Results

AWS personalize uses the past activities, interests of the user and various other information about the user to suggest activities. It suggests 10 activities to the user that are displayed under the "Suggested for you" section in the frontend. The results from AWS personalize for a particular user who had past activities under the Meetup activity is shown in Appendix A, Fig. [4]

VII. Individual Contribution

Following are the individual contributions of the team members:

- Smrithi: I collaborated with Varsha on AWS Personalize recommendation system implementation. I created the Dataset Group, solution, campaign, and filter on AWS Personalize. I developed the backend for Activity (Event, Study Group, Meetup) creation, fetching activity details given an activity_id, and obtaining recommendation results from AWS Personalize from Meetup, Study Group, and Events. I also integrated the API Gateway with Lambdas and created the Architecture Diagram after ideation with Varsha.
- Varsha: I collaborated with Smrithi on AWS Personalize recommendations implementation - Dataset Collection, Preprocessing,Testing for AWS Personalize Recommendations. Moreover, I implemented the entire Backend for Search Functionality using Opensearch, backend for Activity Registration (Meetup, Event, Study Group), User Homepage, Emailing service for Activity Registration, API design and Swagger documentation, Architecture Diagram Ideation.
- Soham: I worked on the UI design and implemented the frontend, authentication such as login and signup, recommendation based on interest tags and APIGateway integration of the frontend with backend.
- Madhura: In this project, I helped with UI designing, and implemented the polling and profile creation, viewing and updating feature. The polling feature includes viewing polls (user specific and general), participating in polls and creating polls.

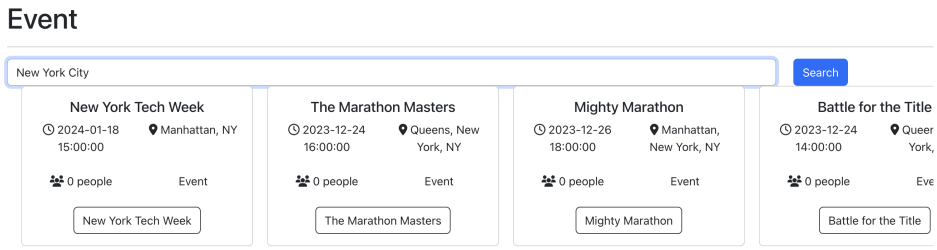
VIII. Conclusion

In conclusion, our project "Socialize!" addresses the challenges faced by Columbia University freshmen in establishing connections by providing a dedicated platform for event exploration, meetups, and study group formation. Unlike existing solutions, our app emphasizes a holistic approach, incorporating polls, AI-based recommendations, and exclusive accessibility to Columbia students. By streamlining event coordination and study group creation, we aim to enhance social interactions and foster a sense of community. "Socialize!" represents a novel and valuable addition to campus life, promoting inclusivity, trust, and a more integrated social network for Columbia students.

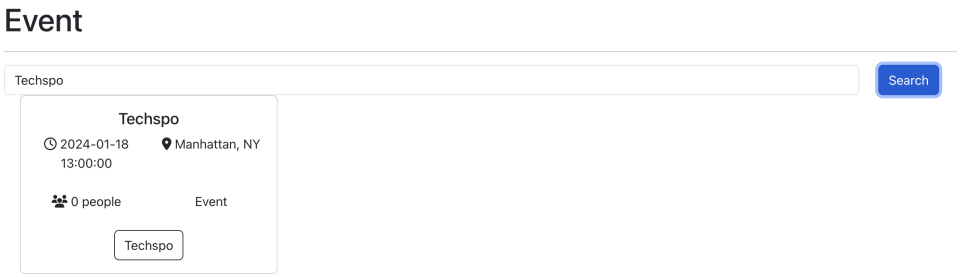
Socialize currently caters to Columbia University students, with a focus on freshmen and international students new to the city. While our initial target is Columbia, the product holds the potential for future expansion, making it a scalable venture that could encompass other universities.

Beyond its university focus, Socialize envisions potential applications in corporate settings, offering a versatile platform for team-building, event coordination, and fostering connections within corporate companies.

APPENDIX A
RESULTS



(a) Search based on location



(b) Search based on title

Fig. 2: Results based on different search such as location and title

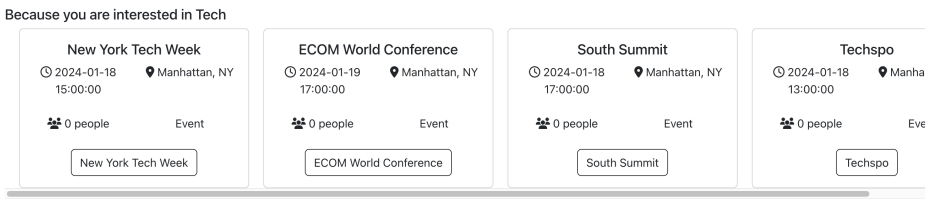


Fig. 3: Results as displayed in the frontend for interest based tag

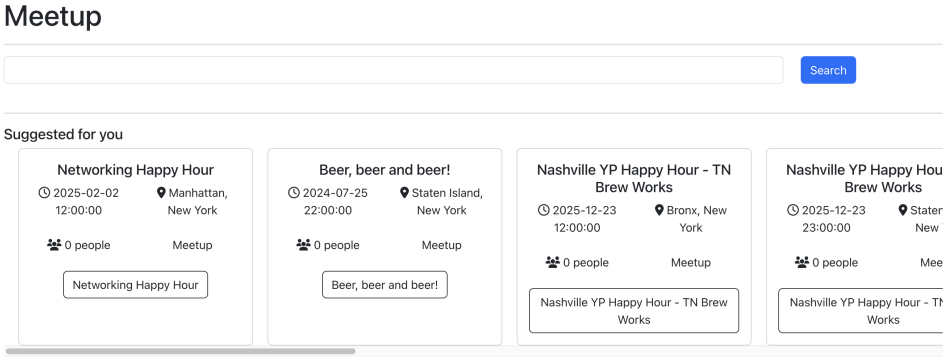


Fig. 4: Results as displayed in the frontend from AWS personalize