GetOutNow

Final Project Report
COMS E6998 008 Spring 2022 – Cloud Computing & Big Data

Mentor
Parth Jawale

Team
Sairam Haribabu (sh4188)
Roshan Koottiyaniyel (rbk2145)
Aishwarya Sivakumar (as6418)
Kaylee Treviño (kt2846)

Table of Contents

Abstract **3**

Introduction **3**

Problem Statement 3

Architecture **3**

Architecture Diagram 3

Architecture Flow 4

API Design **5**

Project Details **7**

Conclusion **7**

Future Work 7

Link to Demo **7**

Abstract

We will maintain an online web service that allows users to search for events, receive recommended events, find events they're interested in, and find events their friends are interested in. Through this service we will also provide a social media-like experience that allows users to browse through a feed of their friends' events.

Introduction

Sometimes it can be difficult to decide what to do in our free time. There are so many events happening around us, but we don't always know which ones we'll enjoy and which ones our friends will enjoy. It can additionally be even more difficult to coordinate attending those events with our friends.

Problem Statement

We want an online system that allows users to select types of events they're interested in and what location they're in so that our service can better provide the user with event recommendations. We additionally want our system to maintain unique and secure profiles for each user; each user can have friends for whom they will be able to see what events they are interested in and planning to attend. Additionally, for each event, we will also provide suitable information for that event such that the user would be able to learn more about the event, book a ticket for the event, and view information about the event that is sufficient enough for the user to know when and where to attend the event. Finally, for each event which a user has decided to attend, a user will be sent an email, confirming their decision to attend this event.

Architecture

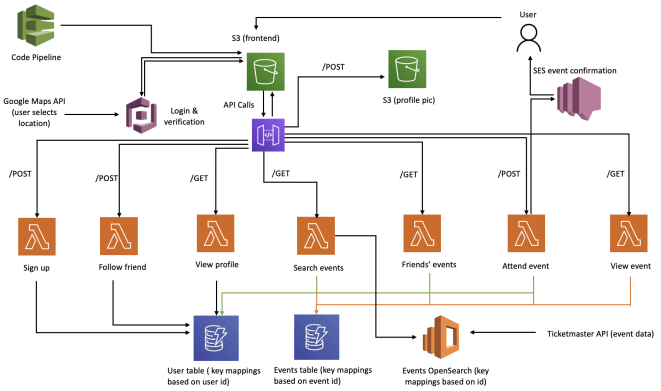
We used a multitude of AWS services such as

- CodePipeline to better maintain the development process and ensure secure and consistent deployment of our code
- Cognito to allow secure user sign-up and verify user sign-in
- S3 bucket to host our frontend
- S3 bucket to maintain user's profile pictures
- SES to email users, reminding them of and confirming their decision to attend an event
- Lambda functions to perform the main functionality of our web application and allow user interaction with our site
- DynamoDB to maintain a user table
- DynamoDB to maintain an events table
- OpenSearch to effectively search for events which will be shown to users based on their preferences or searches

We also used the following external APIs:

- Ticketmaster API to provide event data
- Google Maps API to allow users to select their location

Architecture Diagram



Architecture Flow

The flow for our service is as follows:

1. Data in the Events OpenSearch and Events DynamoDB will be populated in advance using data from the Ticketmaster API
2. User accesses the frontend hosted on S3
3. User must login with a valid username and password, which is verified using AWS Cognito
    a. If user does not have a valid username and password, user must create a profile where they will provide their name, username, bio, preferences, and profile picture
    b. Entries in the Users DynamoDB table will be populated for each user upon sign-up
    c. Profile pictures for each user will be placed in a S3 bucket upon sign-up
    d. Users will select a valid location with the help of the Google Maps API
4. After login, user will originally be shown events based on their preferences provided during sign-up
5. User can search for events based on name, aliases of the event name, location, categories, and dates
6. Users can view a feed of all of the events which their friends have decided to attend
7. S3 makes API calls via API Gateway
8. API Gateway has four /GET calls

    Get user profile information

    a. Uses the View Profile lambda to retrieve information about the user such as their name, username, bio, profile picture (which is stored in the S3 bucket), and more
    b. When the user clicks on any other user or on their profile page, the View Profile lambda is triggered which uses the user's username to query information about the user from the user DynamoDB table

    Get information about events

    a. Fetches basic information about upcoming events based on the user's preferences (their location and favorite categories) or based on the user's search for events
    b. When the user clicks on the homepage, the Search Events lambda is triggered, which will use the user's location and favorite categories to suggest events for the user
    c. When the user searches for events in the search bar, this same lambda is triggered, and it will provide the user with events that match the user's search based on name, aliases of the name, location, categories, or date
    d. In both cases, we will query the OpenSearch for events that match the criteria mentioned above

    Get information on friends' events

    e. Uses the Friends Events lambda to search the user DynamoDB table to find the user's list of friends, events each friend is attending, and the events DynamoDB table to find basic information about those events

    Get information about an event

    a. When a user clicks on an event, the View Event lambda is triggered to query the events DynamoDB table based on the event ID and find more

information about that event which can be shown to the user

9. API Gateway has three /POST calls
   Sign up as a user
   a. Uses the Sign Up lambda to allow a user to create a unique and secure profile
   Friend another user
   a. Uses the Follow Friend lambda to update the user's list of friends in the DynamoDB table
   State intent to attend an event
   a. Uses the Attend Event lambda to assign this user to the list of attendees for the event in the event DynamoDB table
   b. Uses the same lambda to update the user DynamoDB table, indicating the user will attend this event
10. Query results and updates to a user's profile (such as adding a friend or an event) will be displayed in the frontend for the users to see

API Design

/GET (Get user profile information)
Description: With the JSON response returned from the View Profile lambda function, we will display the user's profile information; this includes a link to the user's profile picture which is being stored in an S3 bucket.
Request Body: {
    userID: string
}
Response Body: User

/GET (Get information about events)
Description: With the JSON response returned from the Search Events lambda function, we will display the events, each with their image and name, on the frontend homepage. These events will originally be based on a user's favorite categories and saved location. However, these events can also be updated based on a search performed by the user, requesting users with a certain name, alias, location, category, or date.
Request Body: {
    keyword: string
}
Response Body: {
    event: [Event]
}
Each Event object is in the following format:

```
Event = {
    name: string,
    aliases: [string],
    date: string,
    image: string,
    note: string,
    info: string,
    categories: [string],
    url: string,
    seatmap: string,
    attractions: {
        name: string,
        externalLinks: {
            homepage: string,
            twitter: string,
            instagram: string,
            facebook: string,
            youtube: string
        }
    }
    venue: {
        name: string,
        address: string,
        location: {
            city: string,
            state: string,
            country: string
        }
    }
    attendees: [User]
}
```

Each User object is in the following format

```
User = {
```

name: string,
bio: string,
email: string,
categories: string,
photo: string,
city: string,
events: [{eventid, name, photo}],
friends: [{username, name, photo}]
}

/GET (Get information about friends' events)
Description: With the JSON response returned from the Friends' Events lambda, we will display the events in the same manner as before, but only display events which the user's friends will be attending.
Request Body: {
userID: string
}
Response Body: {[Event]}

/GET (Get information about an event)
Description: With the JSON response returned from the View Event lambda, we will display more detailed information about the event, such as the event's information, location, image, dates, and more that is provided in the Event object.
Request Body: {
eventID: string
}
Response Body: Event

/POST (Upload profile picture to S3 bucket)
Description: When filling in their profile information, the user will upload their profile picture which will be stored in an S3 bucket.
Request Body: {
photo: Object
}
Response Body: {
success: boolean
}

/POST (Sign up as a user)
Description: After the user signs up to use the web service and submits the correct verification code, the user will provide more information about themselves such as their name, username, email, bio, preferred categories, preferred locations, and profile picture.
Request Body: {
username: string,
name: string,
bio: string,
email: string,
city: string,
categories: string,
photo: string
}
Response Body: {
success: boolean
}

/POST (Friend another user)
Description: Upon visiting another user's profile page, the user can click a button indicating they are friending that user.
Request Body: {
userID: string,
friendUserID: string
}
Response Body: {
success: boolean
}

/POST (State intent to attend an event)
Description: Upon visiting an event's page, the user can click a button indicating they would like to attend this event.
Request Body: {
userID: string,
eventID: string
}
Response Body: {
success: boolean
}

Project Details

Overall, we allow users to have a secure sign in, provide information for their profile including a profile picture, provide their preferred categories and location of events for which they'd like to be recommended, friend other users, and view other user profiles. Users can also search for events based on their name, alias, category, location, and/or date. By allowing searching for aliases, we allow users to see events that are commonly referred to as a different name but are not an exact match with the event's actual name. We also allow users to search based on users' usernames and names. We additionally allow users to view a feed of all of their friends' events. For each event shown, in both the search results and the friends' events feed, the user can learn more information about that event. This includes standard information such as the event name, information, date, location, and categories; this also includes information such as a link to tickets for the event and links to social media handles for the group or people which the event pertains to. Finally, we allow users to indicate they will be attending an event and we then send the user an email confirming their decision to attend the event.

Conclusion

Overall, we were able to provide a social-media like service that allows users to maintain individual profiles and interact with both users and events. With this service, users can easily see which events they have decided to attend as well as which events they're friends will be attending in the future. Through this service, users also have the ability to find events based on a robust search engine.

Future Work

In the future, we would like to implement additional features such as allowing users to invite other users to events and unfriend or block other users. We would also like to allow users to create groups within their set of friends; these groups can collectively be invited to an event and will also be able to see posts about group members' plans to meet up at the event, bring items to the event, and more.

Link to Demo

https://www.youtube.com/watch?v=MpTRx82dT00