## horizontal line



Happening Place

Design Document - Team 20

Viswajeeet Balaji

Seerat Dheer

Sanjay Babu Krishna

Devansh Dharmeshkumar Panirwala

Sreekara Yachamaneni

CS 30700

Software Engineering I

Purdue University

# Index

1. Purpose
   1. Overview
   2. Goals
   3. Functional Requirements
   4. Non-functional Requirements
2. Design Outline
   1. High Level Overview
   2. Components
   3. Sequence of Event Overview
3. Design Issues
   1. Functional Issues
   2. Non-functional Issues
4. Design Details
   1. Class & Model Description
   2. Sequence diagram for viewing list of events
   3. Sequence diagram for joining an event
   4. Sequence diagram for event host to create event
   5. Sequence diagram for event host to add contributors to the event
   6. State Diagram
   7. UI Mockups

**Overview**

****

**Happening Place**

Hosting an event or party requires a lot of effort, thinking and coordination and when a group of people organize an event there is confusion about who’s in charge of what and when. Also during massive events like BGR, students come across various events around campus and since there is no central platform which organizes and lists all the event dates and details, students tend to miss events due to forgetfulness or time conflict.

Happening Place is an app which allows you to effectively host events by allowing multiple members to simultaneously plan for a single event without any hassle. It creates a unique dashboard interface for each event and let’s several manage effectively organize a single event. It also allows users to find relevant events around them based on their interests.

Although there exists apps which help users find events around them these apps do not have the functionality of event management. Our goal is to make a one integrated web app which allows users to find events as well as enables users to host an event as a group. This app can be extended to be used in software development projects where a project team leader can add tasks and assign it to specific contributors to finish the task. The core idea behind the project was to implement a event planning methodology so that organizing an event can take place in an efficient manner.

Happening Place also offers users the ability to view around them by using zip code. There is also the facility of filtering events based on tags. Happening Place also uses the guest list of a event and gives the event host a partial audience analysis, so that the host can tailor the event according to the guest list.

**Functional Requirements:**

1. As a user I would like to signup for an Happening Place account.
2. As a user I would like to login and manage my Happening Place account.
3. As a user I would like to reset my login credentials for my account.
4. As a user, I would like to get recommendations on the events nearby, so that I can get the most relevant events nearby me.
5. As a guest, I would like to view list of events in my zip code.
6. As a guest, I would like to view list of events around me in a map. (if time allows)
7. As a guest, I would like to have a recommended list of events based on my interests
8. As a guest, I would like to have the feature of sorting through available events based on certain filters, so that I can view events which are aligned to my interests
9. As a guest, I would like to be able to view the event information such as location, time and event description
10. As a guest, I would like to be able to join an event
11. As a guest, I would like to be able to remove myself from the event
12. As a guest, I would like to be reminded about any upcoming events I have enrolled in, so that I do not forget to attend the event
13. As a guest, I would like to have a timeline feature which lists all events I have attended in the past.
14. As a guest, I would like to have the ability to make my profile public or private.
15. As a guest, I would like to view profiles of other users and connect with them.
16. As a guest, I would like to able to view events visited by my friends.
17. As a guest, I would like to able to share my events with friends.
18. As a guest, I would like to have direct messaging feature with the event host, so that I can communicate directly with him or her.
19. As a guest, I would like to be able to rate an event I attended, so that other users will know the quality of the events organized by this particular host
20. As a host, I would like to create events so that I can communicate and collaborate with like minded people.
21. As a host, I would like to delete or modify the event details in case there is change of location or time due to unforeseen circumstances
22. As a host, I would like to add contributors to this event by searching for their username
23. As a host, I would like to be able to create a list of “cards” .i.e a list of tasks for to be completed for the specific event.
24. As a host, I would like to be able to manage all contributors for a specific event by assigning specific responsibilities(“cards”) to them.
25. As a host, I would like to have a separate channel of communication for with the contributors based on their assigned cards.
26. As a host, I would like to see a list of people who have signed up for this event.
27. As a host, I would like to see the demographics of the people who have signed up for a particular event based on the user’s interest tags, so that the event can be tailored to a specific audience
28. As a host, I would like to be able to communicate new information to the guests so that they can stay updated about the event
29. As a host, I would like to manage questions and responses from participants and forward them to the assigned contributors for appropriate response.
30. As a host, I would like to view past events in a timeline format for historical records purposes
31. As a host, I would like to view ratings and reviews of the event so that I can improve the event quality in the future.
32. As a host, I would like to use tags to describe the event so that the event can pop up in the recommended list for users who have similar tags.
33. As a host, I would like to be able to select the view mode of the event .i.e make an event either a closed private event, open to a select few or make it public.

**Non-­Functional Requirements:**

1. To make an intuitive, elegant and responsive user interface using the extensively used React JS framework.
2. To make a customized dashboard for the event host to manage the event contributors.
3. To make a robust server to handle all incoming requests, check their validity and to communicate with the database. For this we will be using the Amazon Web Service Lamba platform which can handle more than 1000 request simultaneously (response rate of ~10 sec) in the basic version. This limit can be increased by purchasing a premier web services plan.
4. We will be using AWS API Gateway to create a customized API, so that there is smooth and regulated communication between the frontend and backend.
5. To query and store the list of events and users along with other miscellaneous items in an appropriate database. We will be using AWS DynamoDB as our primary database.
6. We are using Amazon web services to ensure that our web app is scalable.
7. The web app will be hosted using Amazon S3 bucket hosting.
8. To give the list of recommended events around user within ~1 second based on an algorithm which parses the tags of events and users.
9. If time allows, we can make the response time of the website better, our code could be hosted at multiple servers across the globe with the help of AWS CloudFront. This would ensure that people across the world can access the website at a faster rate.

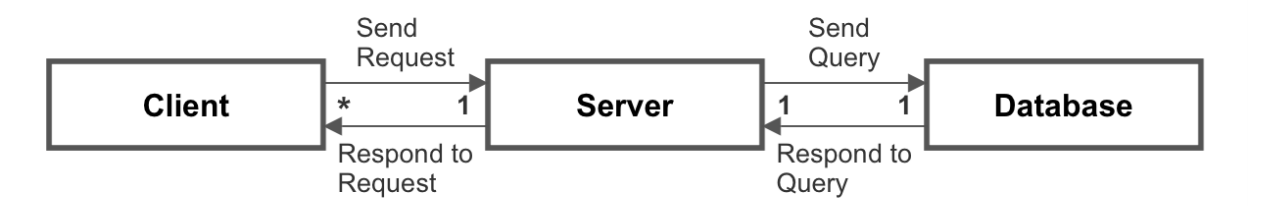
**Design Outline**

High Level Overview

Happening Place is a web application which allows users to search events around them based on their interests. It is also a event management portal which allows several users to organize and contribute to a single event.

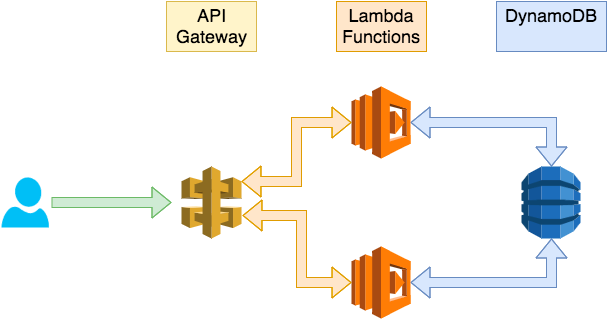
This web application will be implemented using the client server model, where several clients connect to a central server which in turn is connected to the database. For this project our frontend will be built using React JS, a popular javascript library. The client will then send request to the server which is validated through an API Gateway. We are using Amazon Web Services (AWS) API Gateway to catch the incoming requests, validate the request and forward it to the AWS Lambda which is our backend server.

AWS Lambda then processes the request and queries the appropriate database (AWS DynamoDB) and returns the result. In addition, our server will implement the Model-View-Controller pattern in order to modularize the development process.



Components

1. Client
   1. Web application can be run in a compatible browser such as Google Chrome or Mozilla Firefox.
   2. Client renders the user interface which is developed using HTML, CSS & React JS library.
   3. Client sends AJAX request to the API endpoints using HTTP requests such as GET and POST with data in JSON format.
   4. Client retrieves JSON data from server and parses the request and updates user interface.
   5. Client displays the list of events for users. It also displays the dashboard interface for event management portal.
2. API Gateway
   1. API Gateway serves as an abstraction layer between the client and the server.
   2. API Gateway acts a front-door to the server functionality and monitors incoming request from the client and outgoing requests from the server.
   3. API Gateway contains endpoints to which the clients directs the HTTP requests to.
   4. API Gateway validates the incoming request using REST URI conventions and then forwards the valid requests to the server
   5. API Gateway also assists in wrapping metadata around the content returned from the server before forwarding it to the client.
   6. API Gateway will also be handling concurrent calls to the server by aiding in traffic management.
3. Web Server
   1. Web server handles all traffic between the client and the database.
   2. Web server will query the database for a list of events and return it.
   3. Web server will assist in user authentication during login.
   4. Web server will also write and update the database.
   5. Web server will also be generating event lists of user depending on their interest tags.
   6. Web server will be implemented using the Model-View-Controller pattern
4. Database
   1. There will be two databases, one which stores the list of users and other to store the list of events.
   2. Database communicates with the server and responds to the queries appropriately
   3. All relationships between the users and events will be appropriately stored in a persistent database.
   4. AWS DynamoDB is a NoSQL database thus allows flexibility while storing the dynamic user data.



**Design Issues**

Functional Issues

Issue: What information do we need for signing up an account?

● Option 1: Username, email and password

● **Option 2:**, Email, password, interest tags

● Option 3: Username, email, password and tags

We decided to settle on email, password and interest tags since having username and email is superfluous as email id itself would be unique and it would be easier for users to remember. We also wanted to take input of some interest tags from users during their signup so that we can generate some recommended events on the landing page. The password field is required for a secure login and we decided to make it at least 8 characters long.

Issue: Which kind of rating system to use?

● Option 1: Calculate rating based on attendance

● **Option 2:** Provide the users with a 5-star rating system

● Option 3: Calculate rating based on interest tags of guests

We decided to provide the users who have attended an event to rate the event out of 5 stars. This approach is the best option since having attendance based system might not reflect the quality of well organized small scale events. Even the interest tags approach did not seem appropriate since an event host may had incorrect but popular tags to the event to increase rating.

Issue: How to allow the users to communicate with event host?

● Option 1: Real time messaging service

● Option 2**:** Provide the users with the contact info of event host

● **Option 3**: Message Dashboard

We decided to provide the users and event host a message dashboard for the event so that the users can send their questions to the event host in the same platform rather than having to message them or contact them through a mobile phone. We decided our goal was to not make a messaging service app and since real time messaging takes a lot of effort we decided to settle with message dashboard.

Issue: How should we create data structure for Users?

● Option 1: Create a different login for Hosts/Collaborators and Guests

● **Option 2:** Use a single login but different portal for Hosts/Collaborator and Guests

We decided to use a single login for both the admins and the guests as it would be much more memory efficient and easier to work with as a host for one event can be a guest for another event. We decided to have different portal for the guest and host, but with the same login, so that it is easier to work with and also the users will not be confused when they are trying to navigate through their event they are hosting and the events they are planning to attend.

Issue: How should the hosts collaborate to organize the event?

● Option 1: Allow the collaborators to message each other on a group chat

● **Option 2:** By creating a flash card for each event requirements and display it on the screen

● **Option 3:** Let the admin/main host assign roles to the collaborators

● Option 4: Let the guests join as a collaborator by requesting the host

We wanted a very interactive menu which would be simple and efficient for the user to navigate through. We were going to implement chat service for the users but we felt like it was not the core idea of our project. Also chat messaging felt like we were leaning towards instant messaging apps that are currently present in today's world and it didn't feel original. We agreed upon more practical style of our UI so that it is easier for the host and the contributors to distribute the tasks. Letting the guests join the contribution team meant that we would have very complicated code.

Non-functional Issues

Issue: Which cloud service should we use?

● **Option 1:** Amazon Web Services

● Option 2: Microsoft Azure

● Option 3: Google Cloud Platform

We decided to use Amazon Web services since we had some prior experience using it. Also AWS promises higher scalability than Microsoft Azure thus we decided to go with option 1. SInce we are using AWS API Gateway for API endpoints we planned to use Amazon web services for the entire backend infrastructure. We did not have previous experience in Google Cloud Platform or Microsoft Azure.

Issue: What backend language/framework should we use?

● Option 1: Python (Django)

● **Option 2:** NodeJS

● Option 3: PHP

We were considering Node JS or Django for backend framework, but we finalized it to Node JS as it is easier to code it in AWS Lambda. Moreover, we did not have experience in PHP or Django. Additionally, since we are using React JS for the frontend and will be exposed to that framework, we decided that NodeJs will be a great choice for backend since it has lot of documentation.

Issue: What database software should we use?

● Option 1: Firebase

● **Option 2:** AWS DynamoDB

● Option 3: MongoDB

After doing some research on the above three database software we agreed upon using the AWS DynamoDB for storing our data. As DynamoDB is a NoSQL it will allow us to be more flexible as we are going to work and communicate with more flexibility. Unlike firebase and MongoDB comes with many features and web services like AWS Lambda, AWS S3, API gateway etc. Also, on performance bases, DynamoDB provides us with multi-region and multi-master databases with consistent single-digit millisecond latency, and offers built-in security, backup and restore, and in-memory caching.

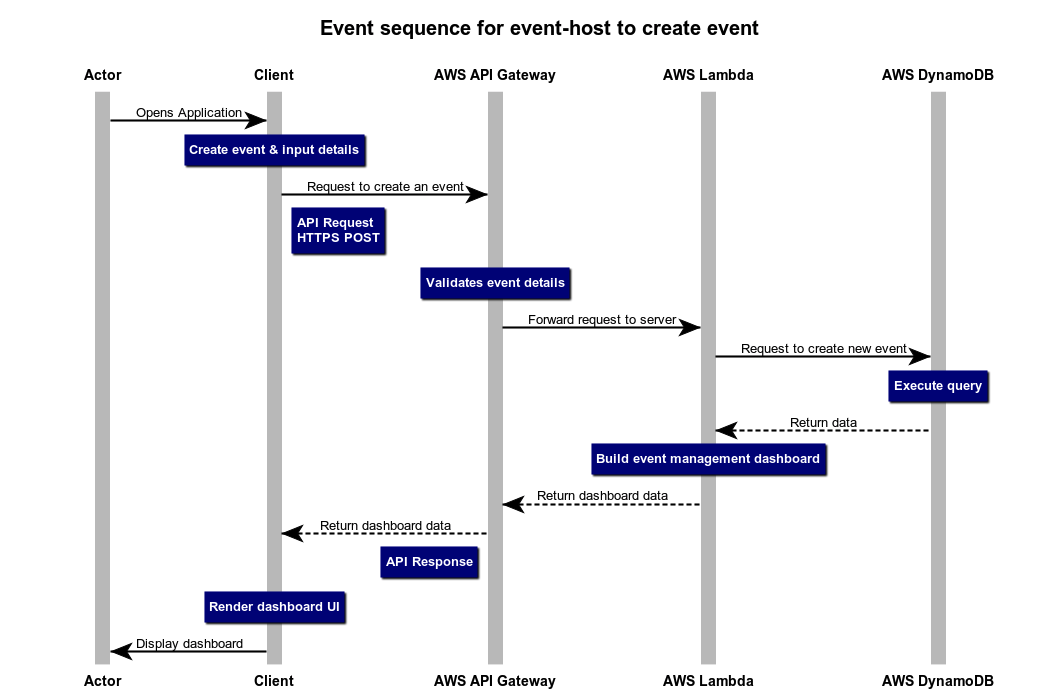
Issue: What front end framework should we use?

● Option 1: Angular

● **Option 2:** React

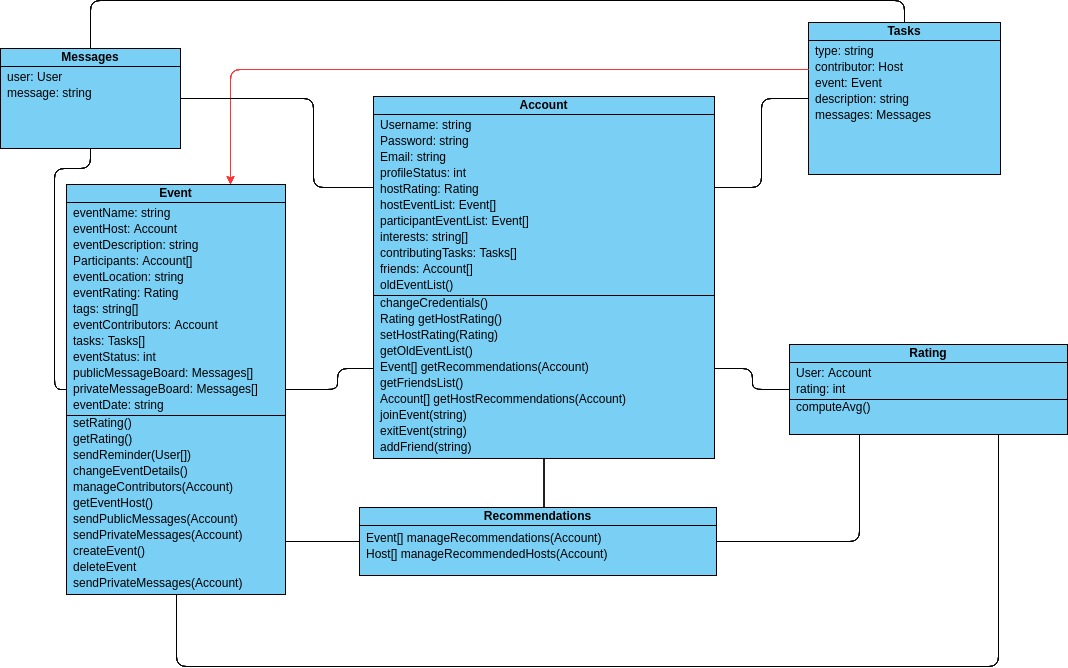
● Option 3: Vue

We decided to use react as it is much flexible than the other two and it meets our software requirements quite well. We chose React.js because it provides us with much better VDOM representation. React js also allows us to define separate components and since our UI is going to be updated frequently, React Js DOM update feature will ensure that our UI elements are updated in a quick and timely manner.



**Design Details**

Class Design

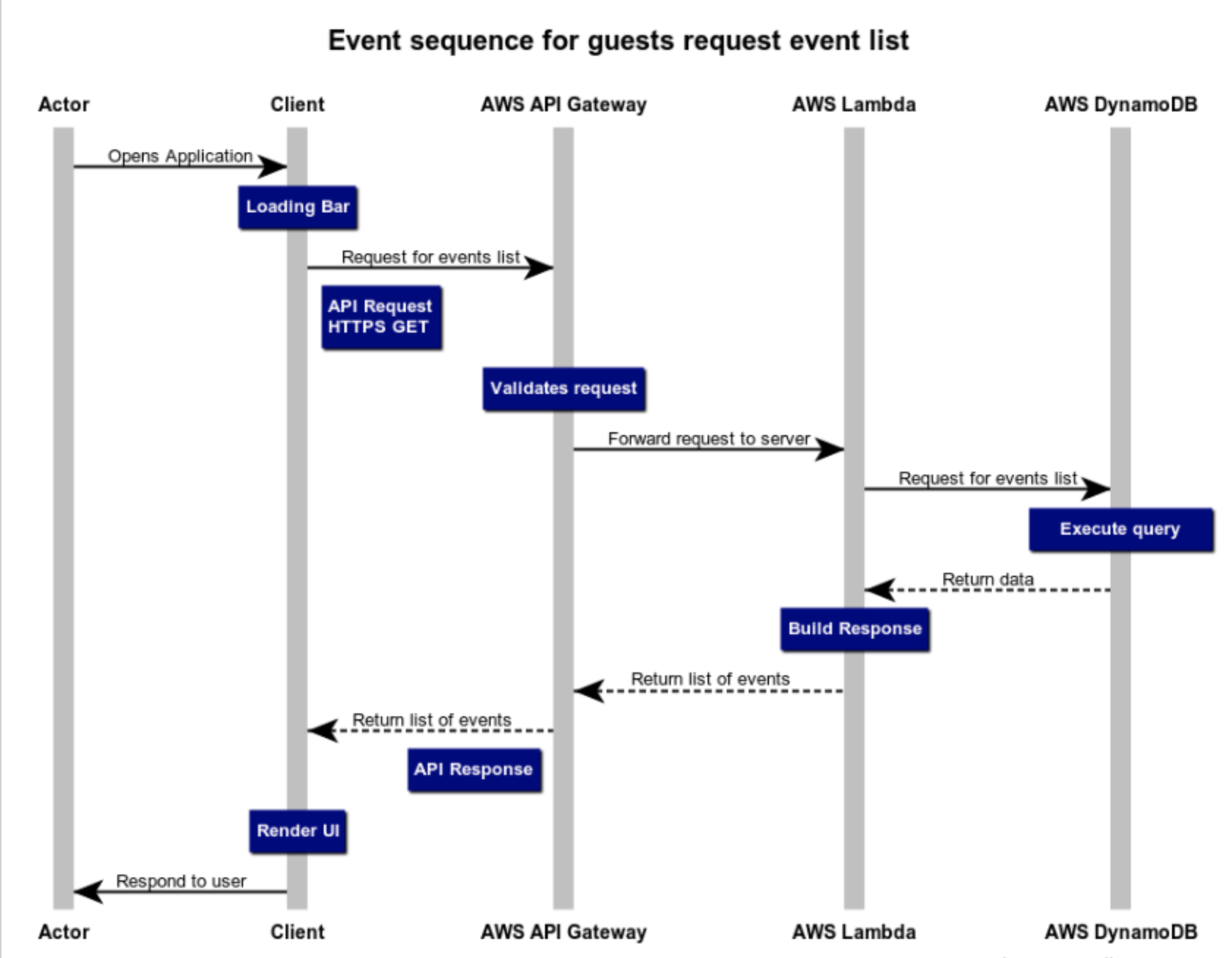


Description of Classes & Interaction among classes

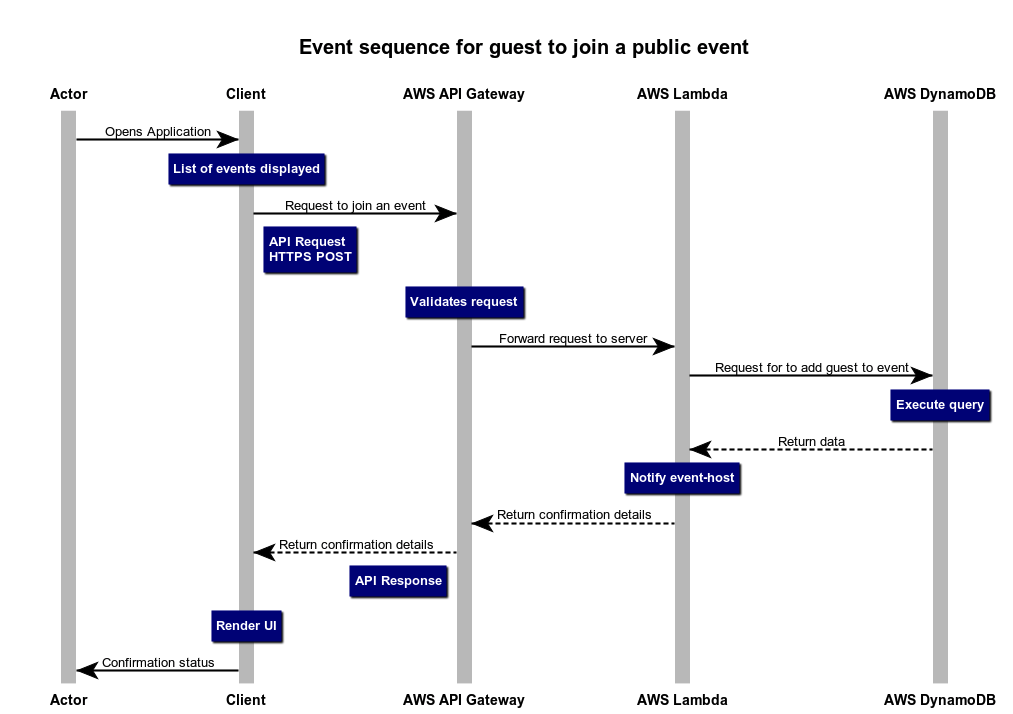
* **Account**
  + Account object is created when someone signs up in our application
  + Each account will have a username, password and a unique email ID, which will be used for login purposes
  + User will set their account status to private(0) or public(1)
  + Every Account object can act as both an event Host and/or event Participant
  + Every Host(Account) will have a rating which will be provided by other Accounts.
  + Every Host(Account) will have an event list which will show the list of events they have created
  + A host will be able to view a list of events they have made in the past
  + Account will have a list of events, which shows the events he has decided to participate in
  + Account will have a list of interests which allows the recommendations class to suggest events
  + Account will also have a list of friends whom they can add using email ID
* **Event**
  + Event will have a Name
  + Event will have a Host and the Host will provide all the details for the event
  + Event will have a list of participants which will be an array of users who have registered for the event
  + Each event will have a list of tasks associated with it
  + Event will have a rating which depends on the inputs of Account
  + Event will have interests (tags) which can be used to find the event using the search bar or users personal interests
  + Event will help manage messages. Private messages among participant and host and Public messages from host to users
* **Tasks**
  + Each tasks will have an individual contributor of type host.
  + Each task will be for a single, specific event
  + The task will have a type(name) and and a description of what it is
  + Task will also have a list of messages which is basically how a host of the event and the contributor for the specific task will communicate
* **Recommendations**
  + Will provide event list based on common event and user tags and the rating of the event
  + Will provide a host list based on tags of events managed by the host and host rating
* **Rating**
  + Will allow account class to rate events and/or hosts
  + Will have a user object which will be used to provide details of the person who rated
  + Will follow a like-dislike system. More like means higher in recommendations. Dislike is only for personal reference
  + Also contains rating type. I.e. Host(0) and Event(1).
* **Messages**
  + Helper class which helps in communication in different classes
  + Stores User object and messages
  + Messages can be sent from host to all the participants.
  + Messages can be sent from a single participant to host.
  + Messages can be sent to contributors as mentioned in **Task**

**Sequence Diagrams**

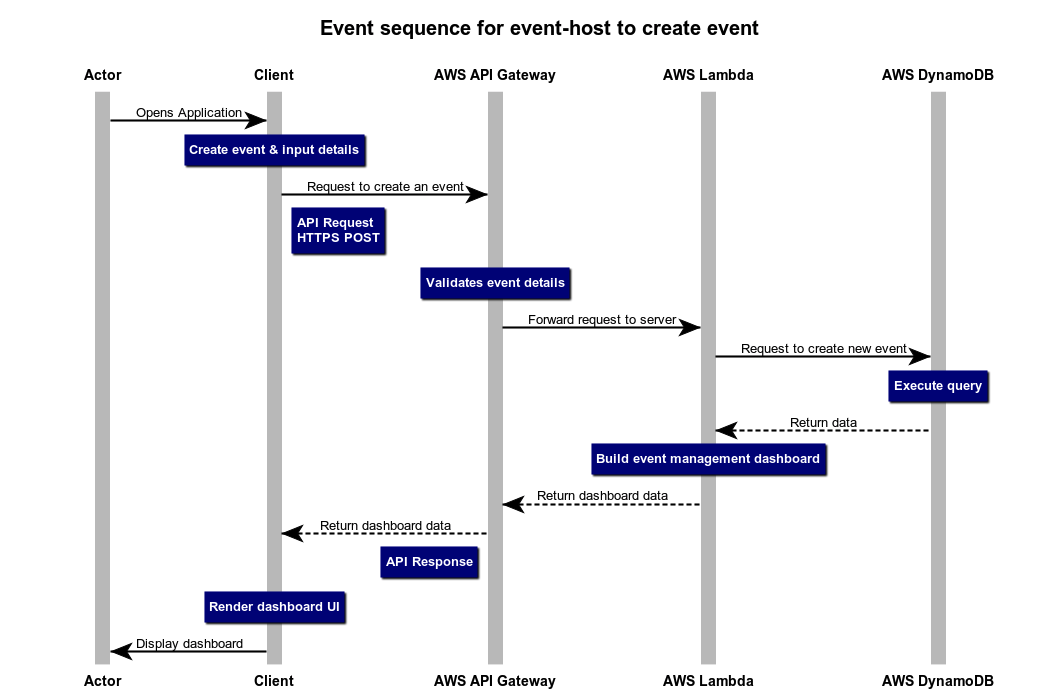
The sequence diagrams show the typical interaction among the actor/user, client, API Gateway, server and database. The user initiates the sequence and continues interacting with the client. As the client receives an input, it sends the request to the API gateway using HTTP GET and POST requests. The API gateway validates the request and forwards the request to the server. The server then connects with the database, queries it and returns the requested data or updates the database with the appropriate information.

****

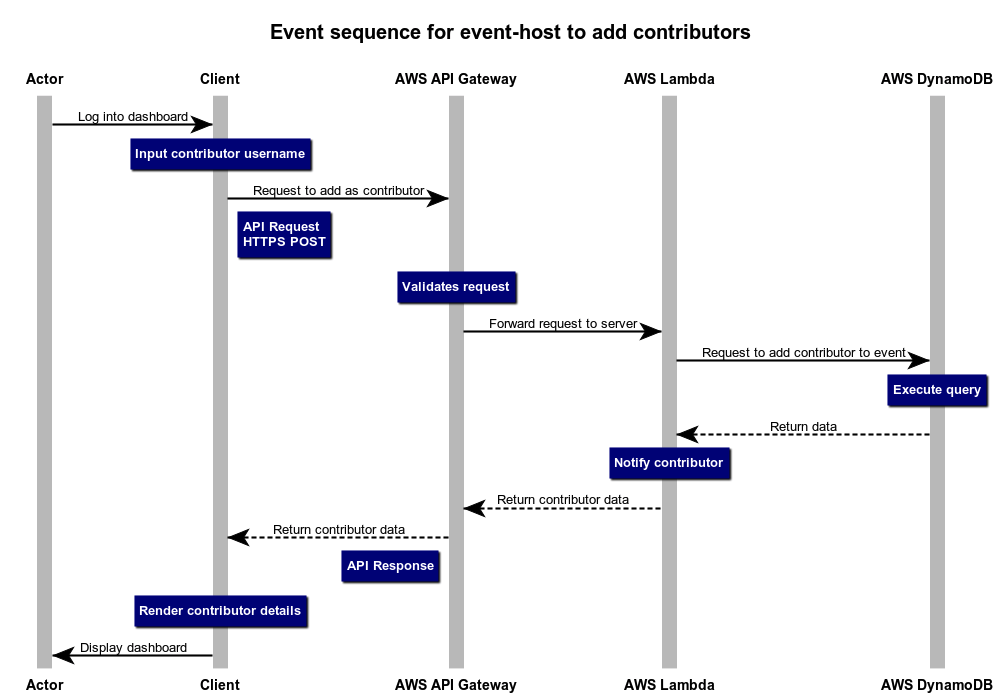
The above sequence diagram shows the typical sequence of steps which a guest takes when he or she wants to view a list of events. The user enters input in a search bar and clicks on the find button. The client then forwards a HTTP request to the API gateway, which in turn validates the request and sends it to the server code at AWS Lambda. This server functionality code then queries the database and retrieves a list of events based on search parameters and the client renders the UI.



The above diagram shows the sequence of events which occurs when the user wants to join an event. The users clicks on the join link button and this sends a HTTP POST request to the API gateway. The API gateway validates the request and forwards it to the server which in turn queries the database and adds the username to the particular event in the database. The server then returns more data about the event to the client and the UI is rendered to the user. This sequence of events successfully adds the user to the guest list of a public event.

****

The above diagram shows the sequence of events which occurs when the event host wants to create and manage an event. The event host clicks on the create event button and this sends a HTTP POST request to the API gateway. The API gateway validates the request and forwards it to the server which in turn queries the database and creates a new event in the database. The server then builds and initializes event management dashboard for that event and the UI is rendered to the user. This sequence of events successfully enables the event host to create a new event.



The above diagram shows the sequence of events which occurs when the event host wants to add more contributors to the event in so that several people can organize a single event. The event host enters a unique user ID of the person and adds the user as contributor. This sends a HTTP POST request to the API gateway. The API gateway validates the request and forwards it to the server which in turn queries the database and adds the specified user as a contributor to the event. The server then builds and initializes event management dashboard for that event and the UI is rendered to the user. This sequence of events successfully enables the event host to add contributors to the event.

**UI Mockups**

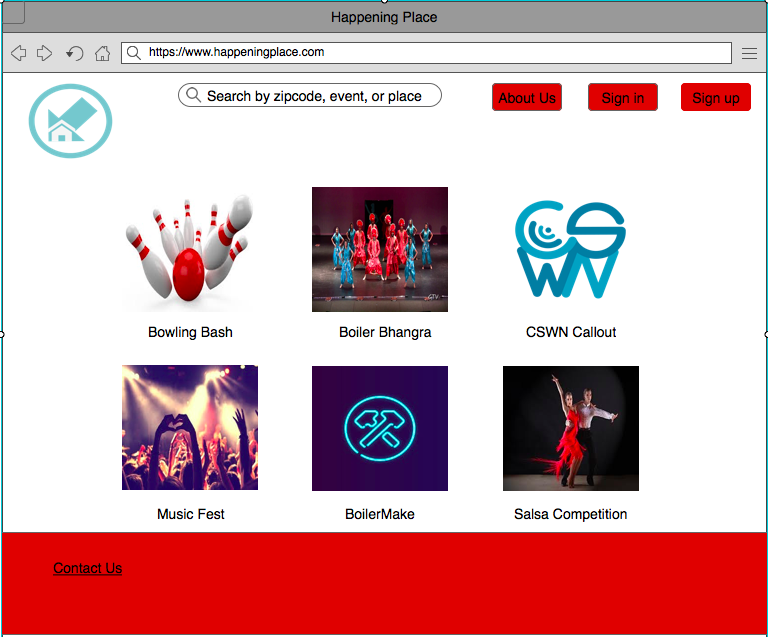


Figure 1: The user interface initially: when a user enters the page initially, he/she will get some of the most popular events nearby. On the top right of the page, the user can learn more about the app by clicking “About Us”. The user can also sign in or sign up from here. On the bottom of the page, the user can contact the website owners with any queries.

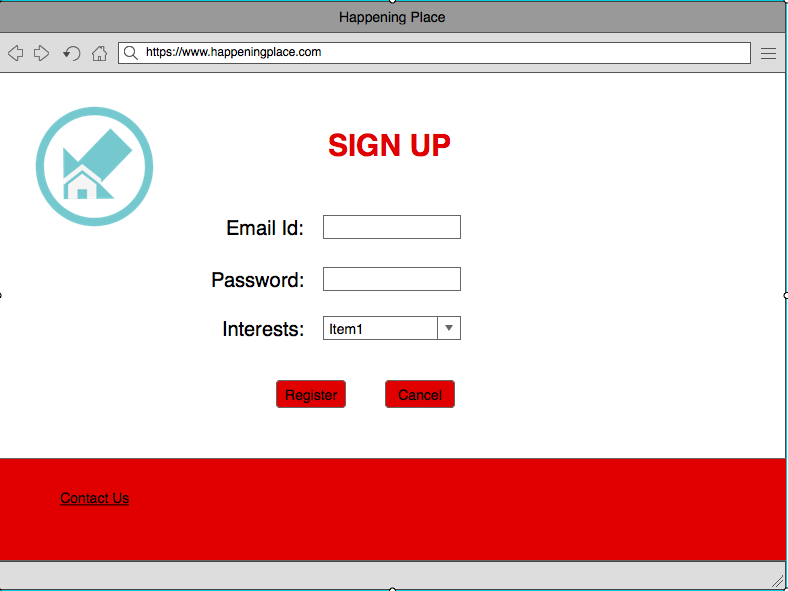


Figure 2: The user interface for sign-up: The user can register to the website. Here the user has to provide his/her email id and password. The user can also put multiple interests so that he/she gets recommendations for the most relevant events nearby. The user can also choose to skip the interests part.

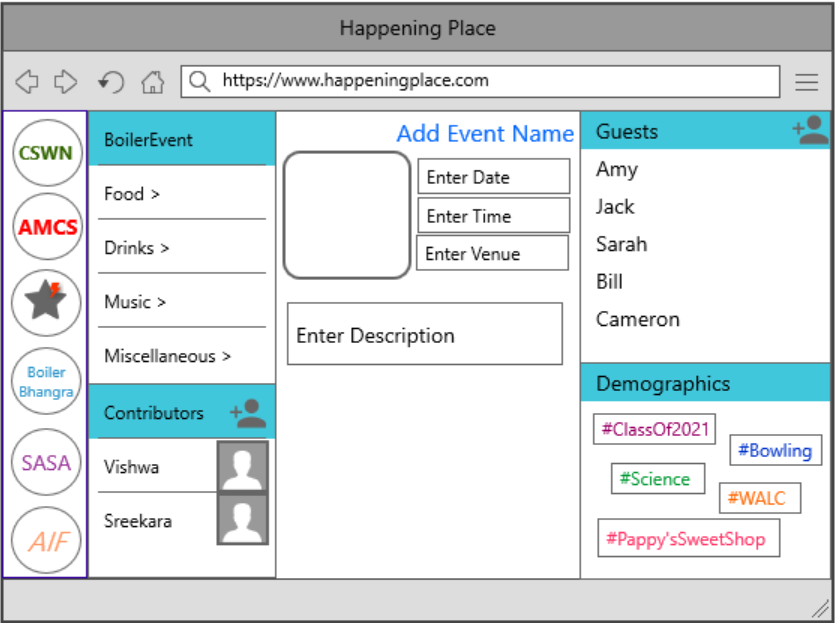


Figure 3 : The User Interface for the Host - This page will be displayed to the user who creates an event. He can add event name, date, time, and venue. He can also upload an image for the event. On the extreme left, there’s a panel containing logos for the other events the host likes. The host can update information about different attributes related to the event, such as Food, Drink, Music, Misc. etc. He can also add other contributors and host using the button provided. On the extreme right, the host can add and view the guests attending the events. The host can also view various tags that the guests have in common under the demographics section.

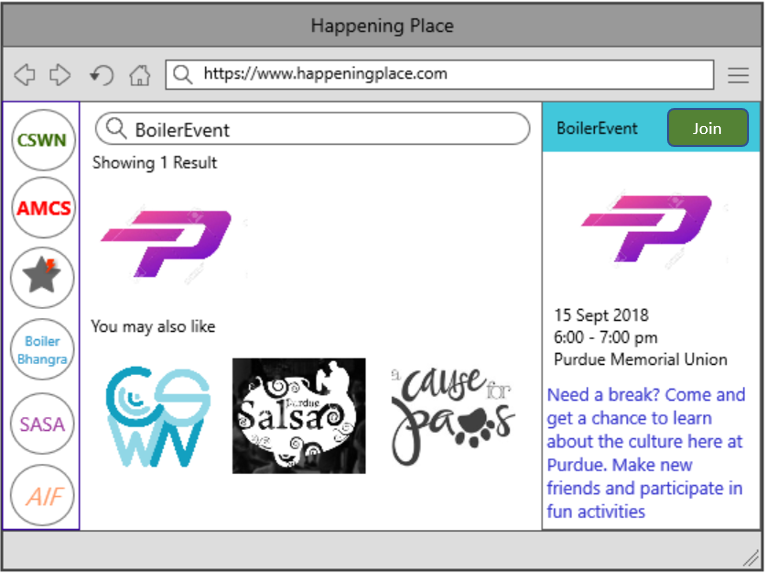


Figure 4 : The user Interface for the guest looking for an event - The guest can search for different events. The results will be displayed along with other recommended events. On the extreme right, the guest can view more information about the event and can join the event using the button provided. On the extreme left, there’s a panel containing the logos of different joined/followed events of the guest.