Event Management App Platform

Introduction

The Event Management App Platform is designed to facilitate the creation, management, and tracking of events within an organization or community. This platform provides a user-friendly interface for managing events, including CRUD (Create, Read, Update, Delete) operations. The application is built using modern technologies to ensure scalability, reliability, and ease of use.

Functional Requirements

Users

- **Create Events**: Users can create new events.
- **Update Events**: Users can update existing events.
- **Delete Events**: Users can delete events.
- **View Events**: Users can view all events.

Technology Stacks

Front End

- Angular

Back End

- Spring Boot
- MySQL Server Database

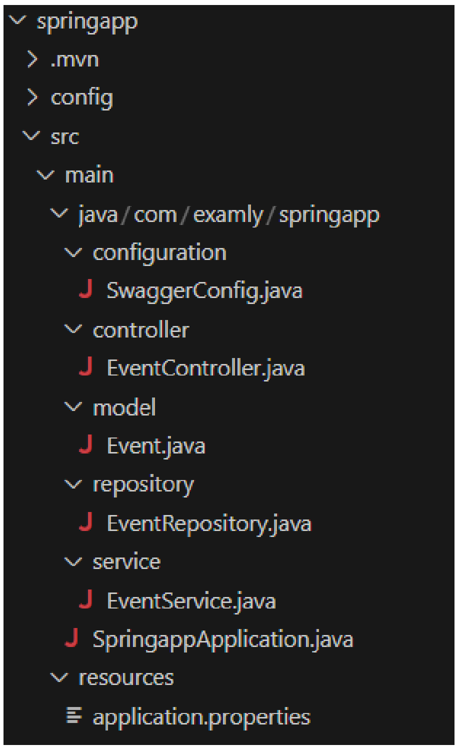
Application Assumptions

- The homepage should be the first page rendered when the application loads.
- Users can navigate directly to the homepage without login and registration.

Backend Requirements

Folder Structure

Create folders named as model, repository, controller, service, and config inside the springapp.



```
springapp
  .mvn
  config
  src
    main
      java / com / examly / springapp
        configuration
          SwaggerConfig.java
        controller
          EventController.java
        model
          Event.java
        repository
          EventRepository.java
        service
          EventService.java
        SpringappApplication.java
      resources
        application.properties
```

1. Model Class

Event.java: Represents an event in the system.

- **Properties:**
  - id: Long (Primary Key, auto-generated)
  - name: String
  - description: String
  - location: String
  - date: String

2. Repository Interface

EventRepository.java: Extends JpaRepository<Event, Long> to handle CRUD operations for events.

3. Service Class

EventService.java: Interface with methods for event management.

```
public interface EventService {
        List<Event> getAllEvents();
        Event getEventById(Long id);
        Event createEvent(Event event);
        Event updateEvent(Long id, Event event);
        void deleteEvent(Long id);
    }
```

EventServiceImpl.java: Implements the EventService interface.

4. Controller Class

EventController.java: Implements REST APIs for managing events.

5. Config Folder

CorsConfig.java: Create a class CorsConfig with @Configuration annotation that implements WebMvcConfigurer to enable CORS functionality.

API Endpoints

1. GET /api/events

- **Action**: Retrieve all event items.
- **Method**: GET
- **Responses**:
  - 200 OK: Returns a list of event items.

2. GET /api/events/{id}

- **Action**: Retrieve a specific event item by ID.
- **Method**: GET
- **Responses**:
  - 200 OK: Returns the event item with the specified ID.
  - 404 Not Found: If the item with the specified ID does not exist.

3. POST /api/events

- **Action**: Create a new event item.
- **Method**: POST
- **Responses**:
  - 201 Created: Returns the created event item.
  - 400 Bad Request: If the request data is invalid.

4. PUT /api/events/{id}

- **Action**: Update an existing event item by ID.

- **Method**: PUT
- **Responses**:
  - 200 OK: Returns the updated event item.
  - 404 Not Found: If the item with the specified ID does not exist.

5. DELETE /api/events/{id}

- **Action**: Delete an event item by ID.
- **Method**: DELETE
- **Responses**:
  - 200 OK: If the item is successfully deleted.
  - 404 Not Found: If the item with the specified ID does not exist.

Application Flow

1. Home Page

- The application starts with the home page, where users can view options to create an event and view the event list.

2. Event Management

- **Create Event**:
  - Users can navigate to a form to add a new event through the **"Create Event"** button present on the home page. After creating the event, it redirects to the Event List page.
- **View Events**:
  - Users can view a list of all events through the **"Event List"** button present on the home page.
- **Update Event**:
  - Users can select an event from the list to update its details by clicking on the **"Edit"** button. After updating the event, it redirects to the Event List page.
- **Delete Event**:
  - Users can delete an event from the list through the **"Delete"** button.

Frontend (Angular):

You will build an Angular application that interacts with a backend API to manage events. The frontend will include the following features:

1. Event Model (event.model.ts):

The Event interface will have the following properties:

```
Event {
  id?: number;
  name: string;
  description: string;
  location: string;
  date: string;
}
```

2. EventService (services/event.service.ts):

The EventService will be used to interact with the backend for performing CRUD operations on events. The service will contain methods as follows:

getEvents(): Retrieves all events (HTTP GET).

getEventById(id: number): Fetches a specific event by its ID (HTTP GET).

createEvent(event: Event): Creates a new event (HTTP POST).

updateEvent(id: number, event: Event): Updates an existing event (HTTP PUT).

deleteEvent(id: number): Deletes an event by its ID (HTTP DELETE).

3. EventListComponent (event-list.component.ts):

This component will display the list of events, allow users to delete events, and navigate to the event form for creating or editing events.

Properties:

events: Array of Event[], which stores the list of events.

successMessage: Stores messages like **"Event deleted successfully."**

Methods:

ngOnInit(): Loads the events using getEvents() from EventService.

deleteEvent(id: number): Deletes an event, updates the list, and sets the successMessage.

refreshEvents(): Reloads the events using getEvents() from EventService.

editEvent(id: number): Navigates to the event form for editing an event.

addEvent(): Navigates to the event form for adding a new event.

HTML Template:

A dynamic table will list all events. Each row displays the event's name, description, location, and date.

A "**Delete**" button will trigger the **deleteEvent**() method to remove an event.

Buttons for adding and editing events will navigate the user to the respective forms.

4. EventFormComponent (event-form.component.ts):

This component is used for both creating and editing events.

Properties:

eventForm: A reactive form containing fields for name, description, location, and date.

isEditMode: A boolean that toggles between create and edit modes.

minDate: The current date is used to enforce future date selection for the event.

successMessage: Stores messages like **"Event successfully created" or "Event successfully updated."**

successMessage: string;

Methods:

ngOnInit(): Initializes the form, checks if an event ID is provided (for edit mode), and pre-populates the form if needed.

onSubmit(): Handles form submission. In create mode, it calls createEvent(), and in edit mode, it calls updateEvent(). Sets the successMessage accordingly and navigates back to the event list after a short delay.

Form validation ensures that all required fields (name, description, location, and date) are filled and that the date is set in the future.

HTML Template:

The form includes input fields for the event's name, description, location, and date.

The submit button is disabled if the form is invalid.

Validation messages are displayed for required fields.

The date field enforces a minDate restriction to ensure events are set in the future.

5. Navigation:

The application supports navigation between the event list and the event form (for creating and editing events). The Router service is used to navigate to different routes within the application.

6. Form Validation:

Form validation ensures the following:

All fields are required: name, description, location, and date.

The date field must be set to today or a future date.

7. Success Messages:

Upon successful creation or updating of an event, a success message is displayed, and the user is redirected to the event list after a short delay (e.g., 2 seconds).

Upon deletion of an event, the list is updated, and a success message is shown.

8. Date Validation:

The date field must be set to a future date (tomorrow or later).

9. Routing:

The application will use Angular's routing module to navigate between components. The following routes should be defined:

- '/events': EventListComponent (default route)

- '/add-event': EventFormComponent in create mode

- '/edit/:id': EventFormComponent in edit mode

Implement a routing module (app-routing.module.ts) with these routes.

In the EventListComponent:

- The 'Add Event' button should navigate to '/add-event'

- The 'Edit' button for each event should navigate to '/edit/:id' where :id is the event's ID

In the EventFormComponent:

- After successful creation or update of an event, navigate back to '/events' after a short delay

- Implement logic to determine whether the component is in create or edit mode based on the presence of an 'id' parameter in the route

<u>**Frontend Requirements**</u>

1. Event List Component:

- Displays a list of events with options to edit, delete, or view details.

2. Event Form Component:

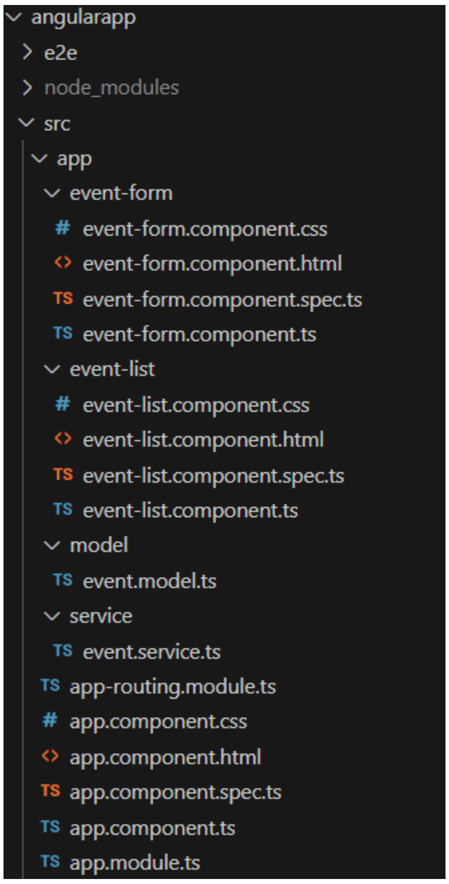
- Form for adding or editing event details.
- Includes validation for required fields such as Name, Description, Location, and Date.

3. Route Configuration:

- **"/events":** Displays the EventListComponent.
- **"/events/add":** Displays the EventFormComponent for adding a new event.
- **"/events/edit/{id}":** Displays the EventFormComponent for editing an existing event by ID.

Folder Structure

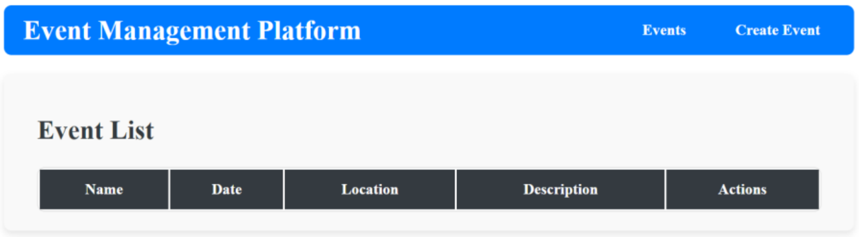
- Maintain a clear and consistent folder structure within the frontend to manage components, services, and routing.



```
angularapp
  e2e
  node_modules
  src
    app
      event-form
        event-form.component.css
        event-form.component.html
        event-form.component.spec.ts
        event-form.component.ts
      event-list
        event-list.component.css
        event-list.component.html
        event-list.component.spec.ts
        event-list.component.ts
      model
        event.model.ts
      service
        event.service.ts
      app-routing.module.ts
      app.component.css
      app.component.html
      app.component.spec.ts
      app.component.ts
      app.module.ts
```

Frontend Sample Screenshots

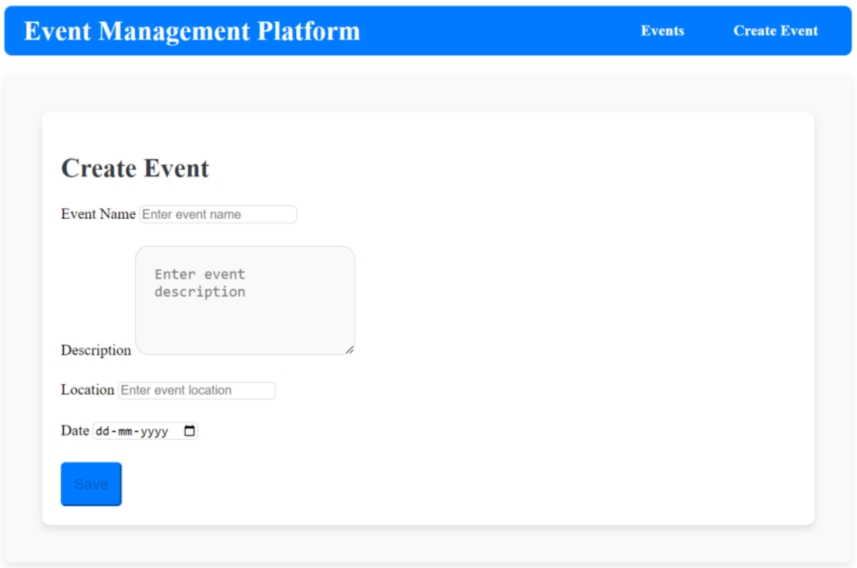
Home Page



Event List

Name	Date	Location	Description	Actions

Create Event Form



Event Management Platform

Events

Create Event

Create Event

Event Name Enter event name

Enter event description

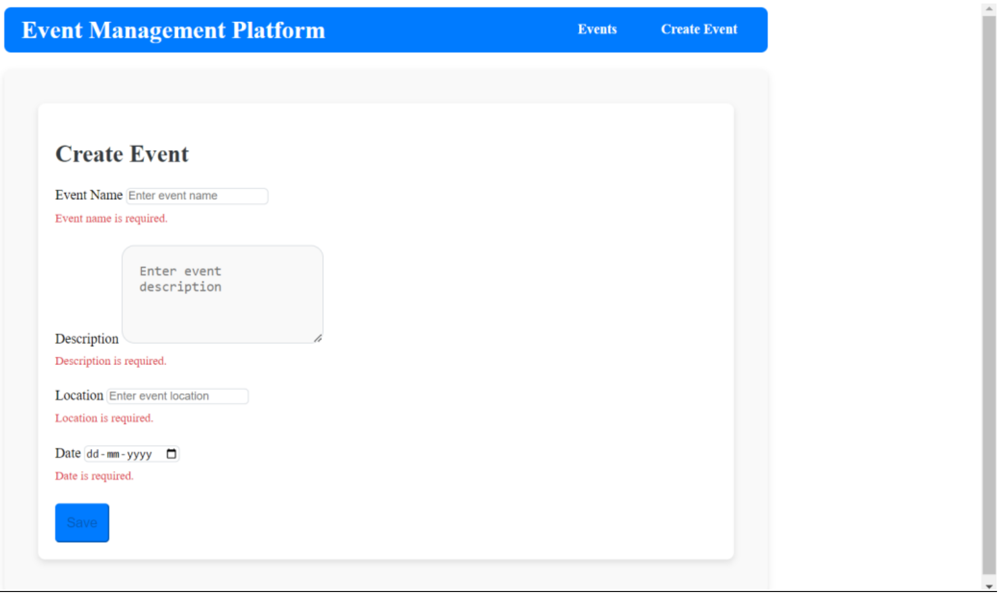
Description

Location Enter event location

Date dd-mm-yyyy

Save

Incase of fields are empty



Event Management Platform

Events

Create Event

Create Event

Event Name Enter event name

Event name is required.

Enter event description

Description

Description is required.

Location Enter event location

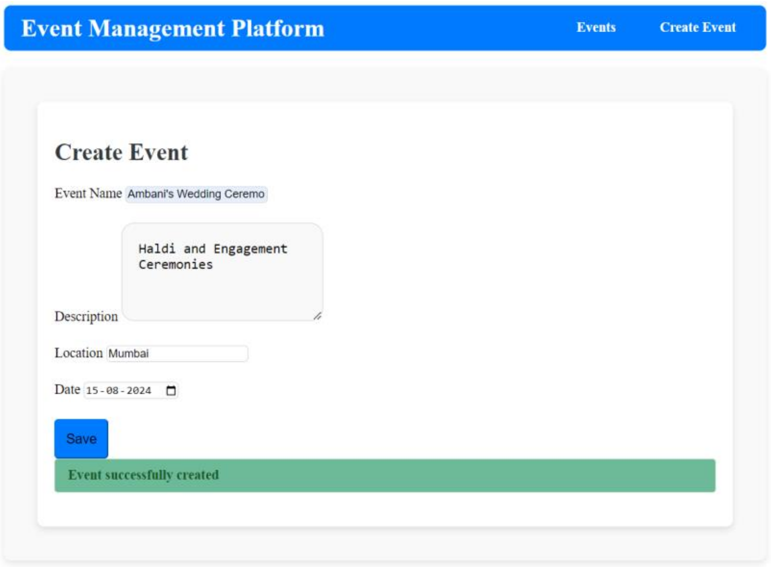
Location is required.

Date dd-mm-yyyy

Date is required.

Save

Adding events to Event List


Event Management Platform

Events Create Event

Create Event

Event Name Ambani's Wedding Ceremo

Description
```
Haldi and Engagement
Ceremonies
```

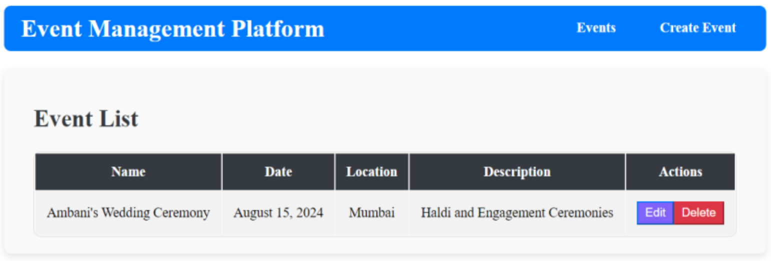
Location Mumbai

Date 15-08-2024

Save

Event successfully created

Event List


Event Management Platform

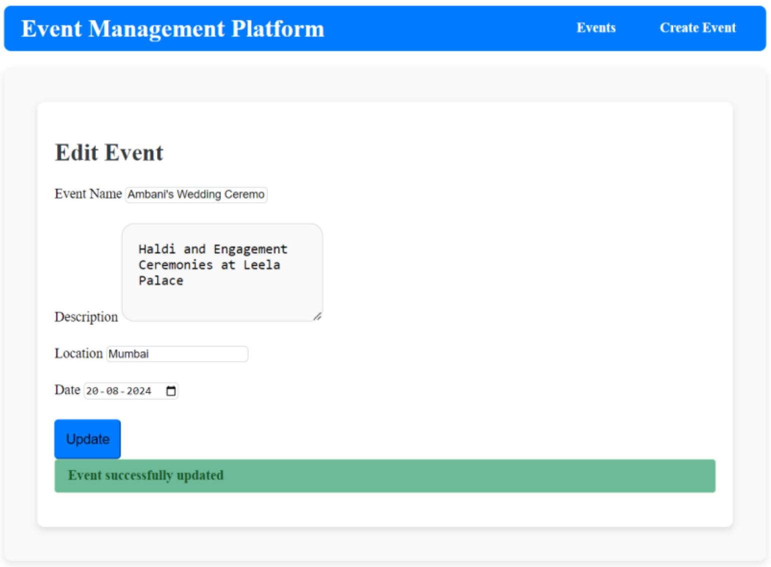
Events Create Event

Event List

Name	Date	Location	Description	Actions
Ambani's Wedding Ceremony	August 15, 2024	Mumbai	Haldi and Engagement Ceremonies	Edit Delete

---

Update Event Form

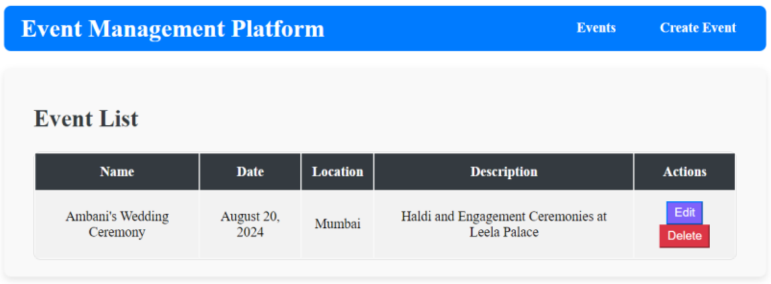


Edit Event

Event Name	Ambani's Wedding Ceremo
Description	Haldi and Engagement Ceremonies at Leela Palace
Location	Mumbai
Date	20-08-2024

Event successfully updated

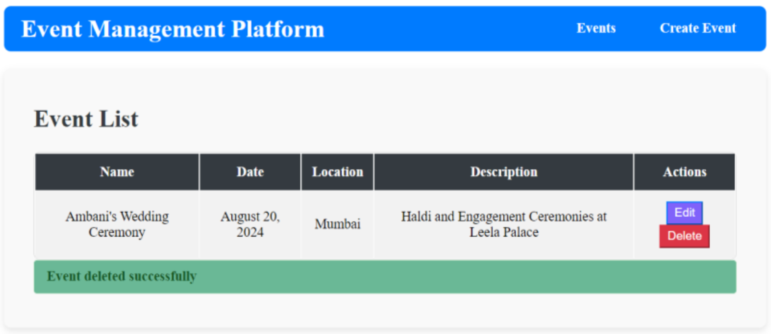
Updated Event List



Event List

Name	Date	Location	Description	Actions
Ambani's Wedding Ceremony	August 20, 2024	Mumbai	Haldi and Engagement Ceremonies at Leela Palace	Edit Delete

Deleted Event from Event List



Event List

Name	Date	Location	Description	Actions
Ambani's Wedding Ceremony	August 20, 2024	Mumbai	Haldi and Engagement Ceremonies at Leela Palace	Edit Delete

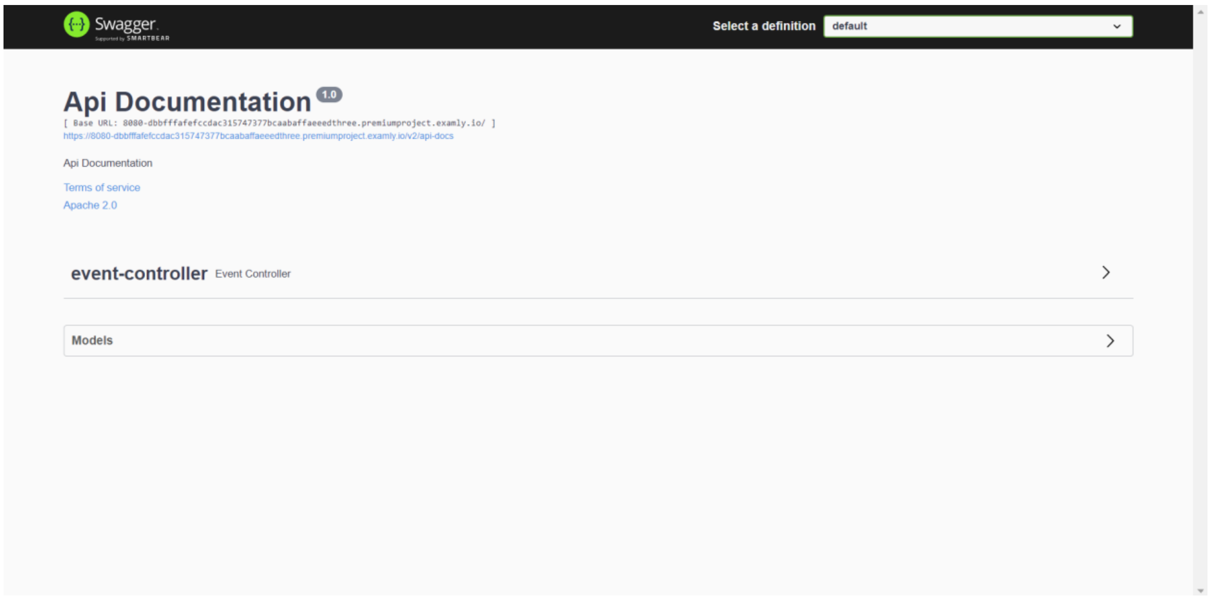
Event deleted successfully

Swagger Implementation

Ensure you are creating a folder and file under the src where you can configure Swagger and implement the same. Below is the expected output post proper configuration and implementation.

Swagger Page

- URL: **backend URL/swagger-ui/index.html**



Api Documentation 1.0

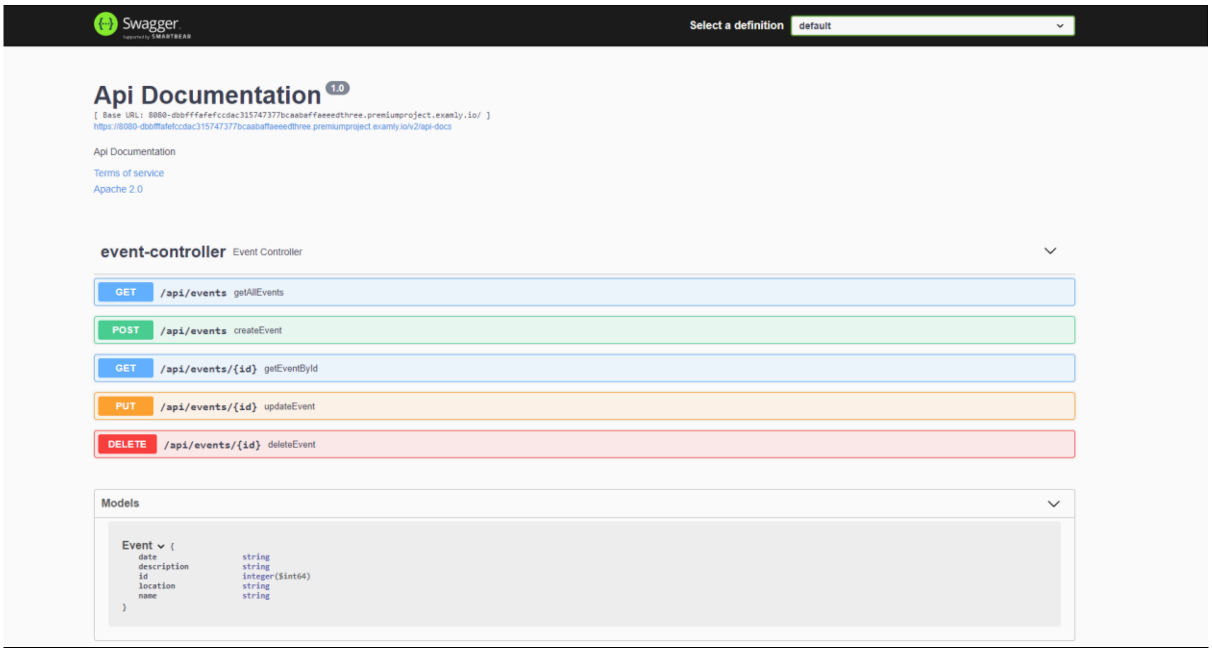
[Base URL: 8080-dbbfffafefccdac315747377bcaabaffaeeedthree.premiumproject.examly.io/]
https://8080-dbbfffafefccdac315747377bcaabaffaeeedthree.premiumproject.examly.io/v2/api-docs

Api Documentation

Terms of service
Apache 2.0

event-controller Event Controller

Controllers and API Endpoints



Api Documentation 1.0

```
[ Base URL: 8080-dbbfffafefccdac315747377bcaabaffaeeedthree.premiumproject.examly.io/ ]
```

https://8080-dbbfffafefccdac315747377bcaabaffaeeedthree.premiumproject.examly.io/v2/api-docs

Api Documentation

Terms of service

Apache 2.0

event-controller Event Controller

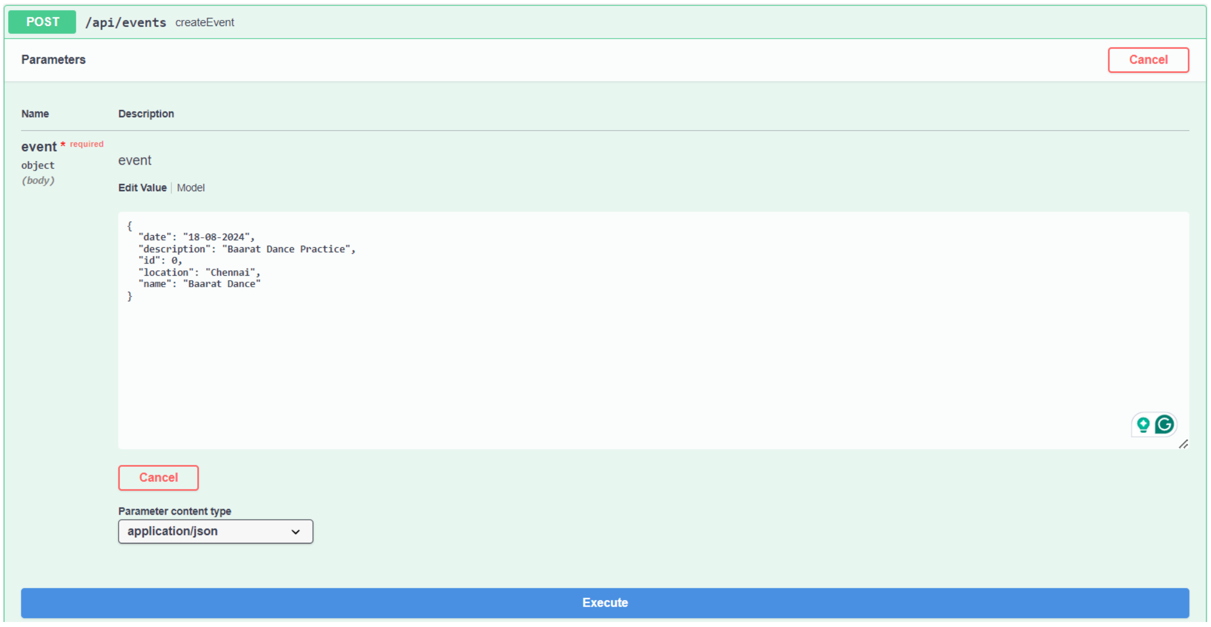
```
GET     /api/events        getAllEvents
POST    /api/events        createEvent
GET     /api/events/{id}   getEventById
PUT     /api/events/{id}   updateEvent
DELETE  /api/events/{id}   deleteEvent
```

Models

```
Event {
    date            string
    description     string
    id              integer($int64)
    location        string
    name            string
}
```

POST API using Swagger

- Click on “Try it out” and enter the event name, description, location, and date. Then click on “Execute” to add the event to the list.



event * required
object
(body)

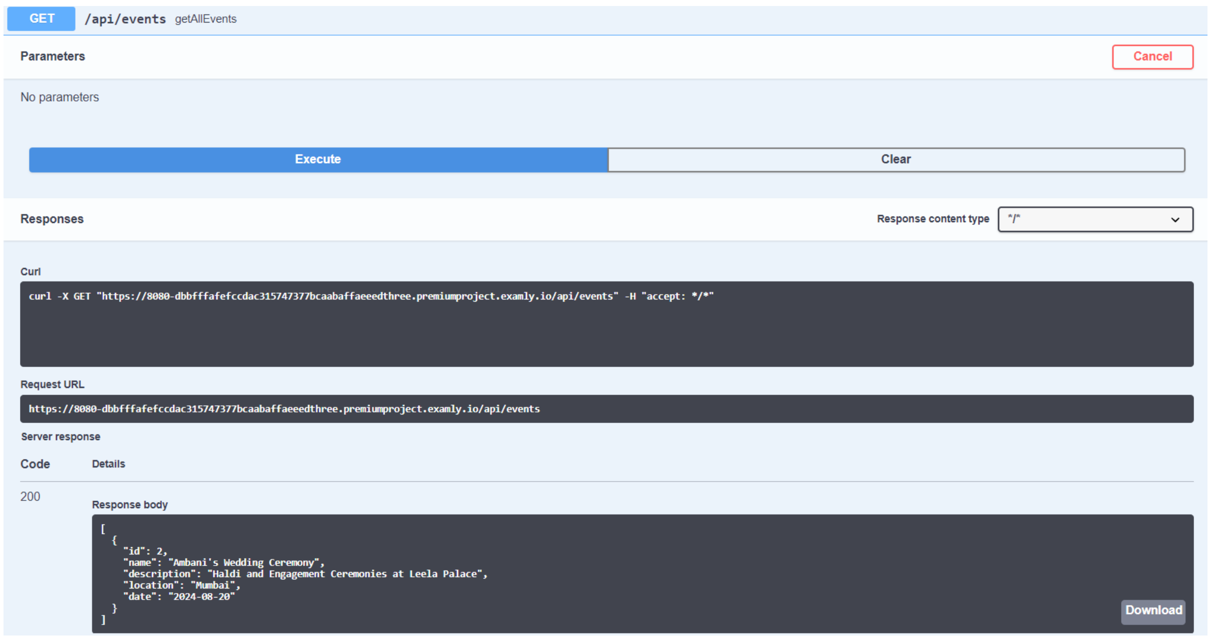
event

```
{
  "date": "18-08-2024",
  "description": "Baarat Dance Practice",
  "id": 0,
  "location": "Chennai",
  "name": "Baarat Dance"
}
```

Parameter content type: application/json

GET API using Swagger

- You can see the event list that has been added recently using POST API.



Curl

```
curl -X GET "https://8080-dbbfffafefccdac315747377bcaabaffaeeedthree.premiumproject.examly.io/api/events" -H "accept: */*"
```

Request URL

```
https://8080-dbbfffafefccdac315747377bcaabaffaeeedthree.premiumproject.examly.io/api/events
```

Server response

Code Details

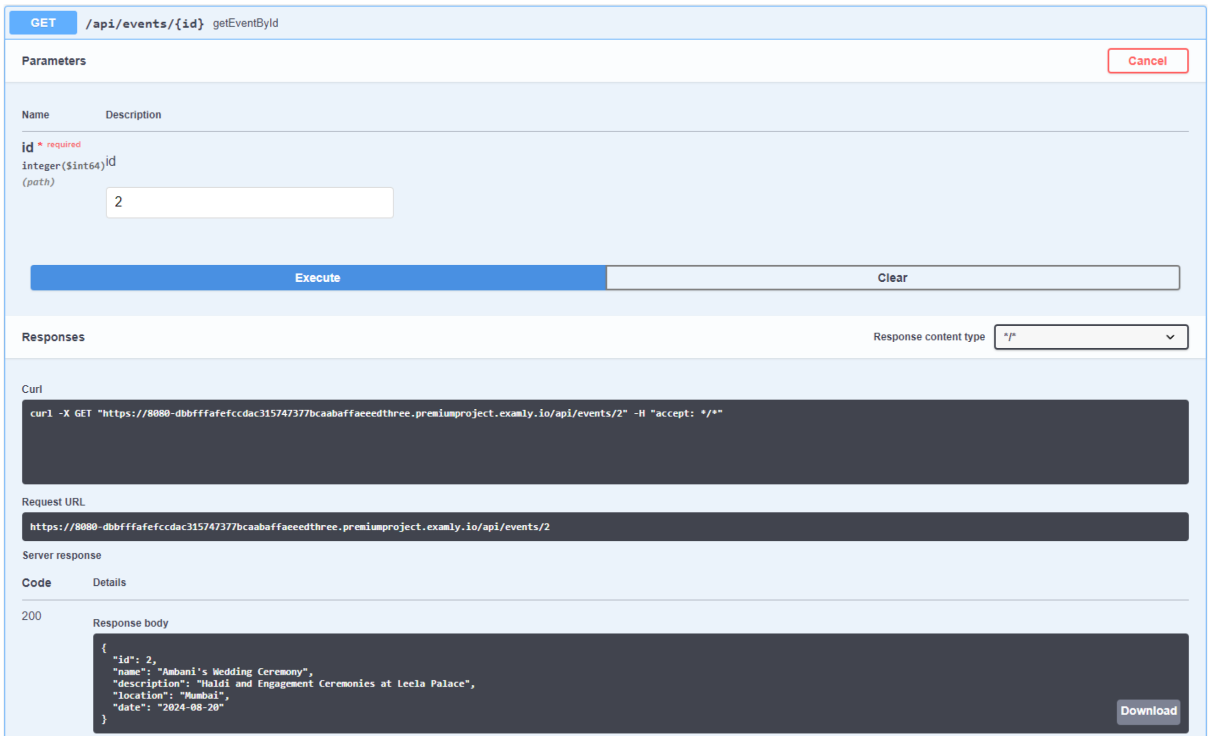
200

Response body

```
[
  {
    "id": 2,
    "name": "Ambani's Wedding Ceremony",
    "description": "Haldi and Engagement Ceremonies at Leela Palace",
    "location": "Mumbai",
    "date": "2024-08-20"
  }
]
```

GET API by Id using Swagger

- Enter the ID for which you want to fetch the details from the event list and click on "Execute" to get all the necessary details.



id * required
integer($int64) id
(path)
2

Curl

```
curl -X GET "https://8080-dbbfffafefccdac315747377bcaabaffaeeedthree.premiumproject.examly.io/api/events/2" -H "accept: */*"
```

Request URL

```
https://8080-dbbfffafefccdac315747377bcaabaffaeeedthree.premiumproject.examly.io/api/events/2
```

Server response

Code Details

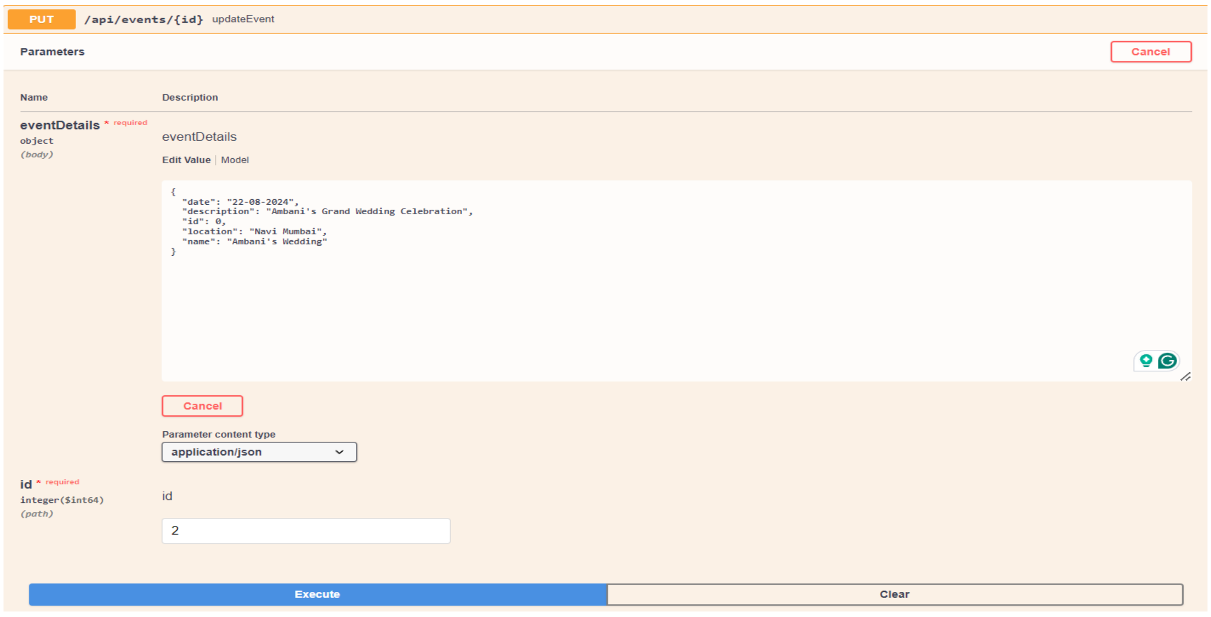
200

Response body

```
{
  "id": 2,
  "name": "Ambani's Wedding Ceremony",
  "description": "Haldi and Engagement Ceremonies at Leela Palace",
  "location": "Mumbai",
  "date": "2024-08-20"
}
```

PUT API using Swagger

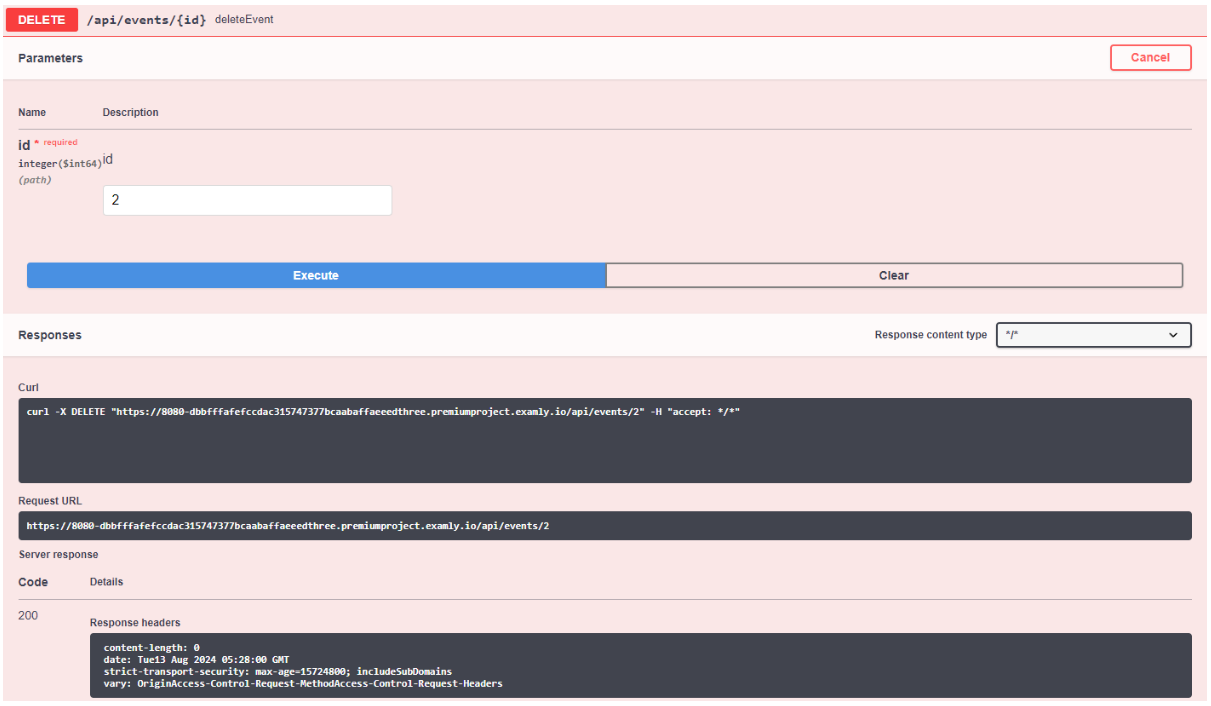
- Provide the Id of the event which you want to update and then enter the updated field details. Then click on the “Execute” button to update the event from the event list.



```
{
  "date": "22-08-2024",
  "description": "Ambani's Grand Wedding Celebration",
  "id": 0,
  "location": "Navi Mumbai",
  "name": "Ambani's Wedding"
}
```

DELETE API using Swagger

- Provide the Id of the event you want to delete and then click on the “Execute” button to delete the event.



```
curl -X DELETE "https://8080-dbbfffafefccdac315747377bcaabaffaeeedthree.premiumproject.examly.io/api/events/2" -H "accept: */*"
```

```
https://8080-dbbfffafefccdac315747377bcaabaffaeeedthree.premiumproject.examly.io/api/events/2
```

```
content-length: 0
date: Tue13 Aug 2024 05:28:00 GMT
strict-transport-security: max-age=15724800; includeSubDomains
vary: OriginAccess-Control-Request-MethodAccess-Control-Request-Headers
```

Key Points to Remember

1. The IDs and attributes mentioned in the SRS should not be modified at any cost.
2. Strictly adhere to the proper project scaffolding (Folder structure), coding conventions, method definitions, and return types.
3. Adhere strictly to the endpoints mentioned in the API endpoints section.
4. Do not delete any files in the project environment.

Steps to Execute the Project

Frontend

To run Angular app:

1. Use **cd angularapp** command to go inside the angularapp folder.
2. Install Node Modules using the command **npm i**
3. Command to start the Angular app: **npm start**
4. Write the code inside the src folder by creating the necessary components and services folders with files.
5. Command to run the project: **npm test**

Note: Click **PORT 8081** to view the result/output. If any error persists while running the app, delete the node modules and reinstall them.

Backend

To run Spring Boot:

1. Use **cd springapp** command to go inside the springapp folder.
2. Command to run/start the application: **mvn spring-boot:run**
3. Command to run the project: **mvn clean test**

- **API endpoint**: 8080

CLOUD DEPLOYMENT:

Event Management Application Deployment on Google Cloud Platform (GCP)

1. Introduction

1.1 Scope

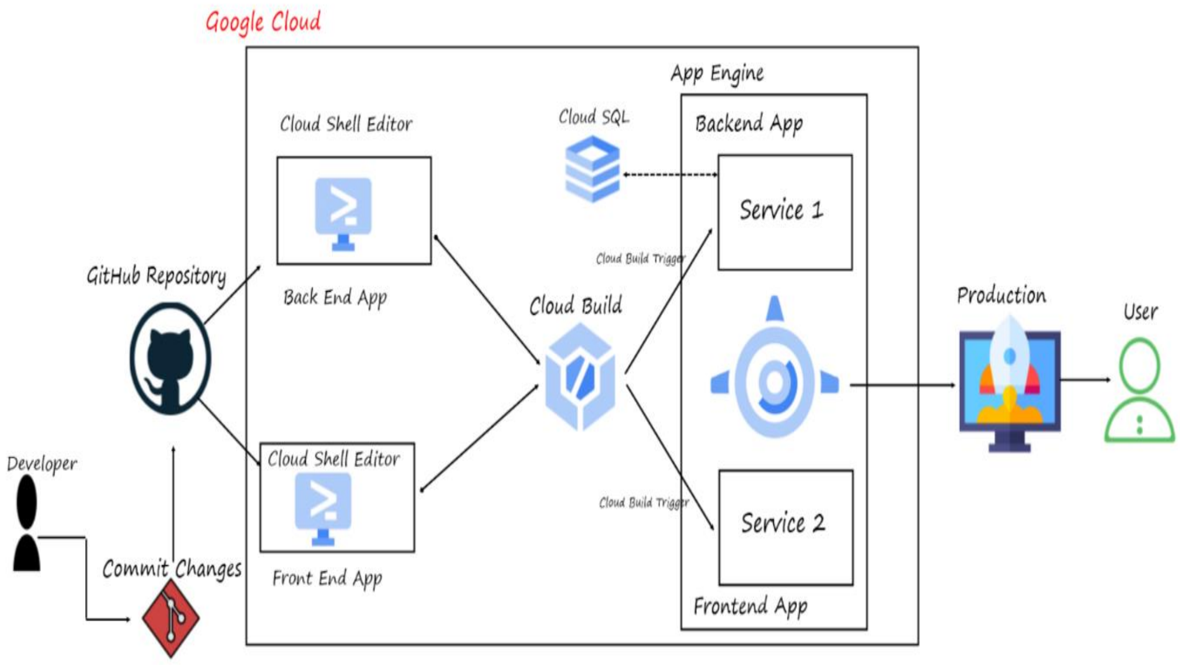
This project involves deploying a cloud-native Event Management Application on Google Cloud Platform (GCP). The application is designed to provide a scalable solution for event planning and organization, with secure access, real-time updates, robust logging, and monitoring capabilities. The deployment leverages GCP services such as App Engine, Cloud SQL, and Stackdriver, integrating security measures and ensuring compliance with industry regulations.

2. System Architecture

2.1 Overview

The system architecture consists of two main components: the frontend and backend, each deployed using GCP services to ensure scalability, security, and high availability. The backend is implemented in Java with Spring Boot, while the frontend is developed as a Single-Page Application (SPA) using Angular.

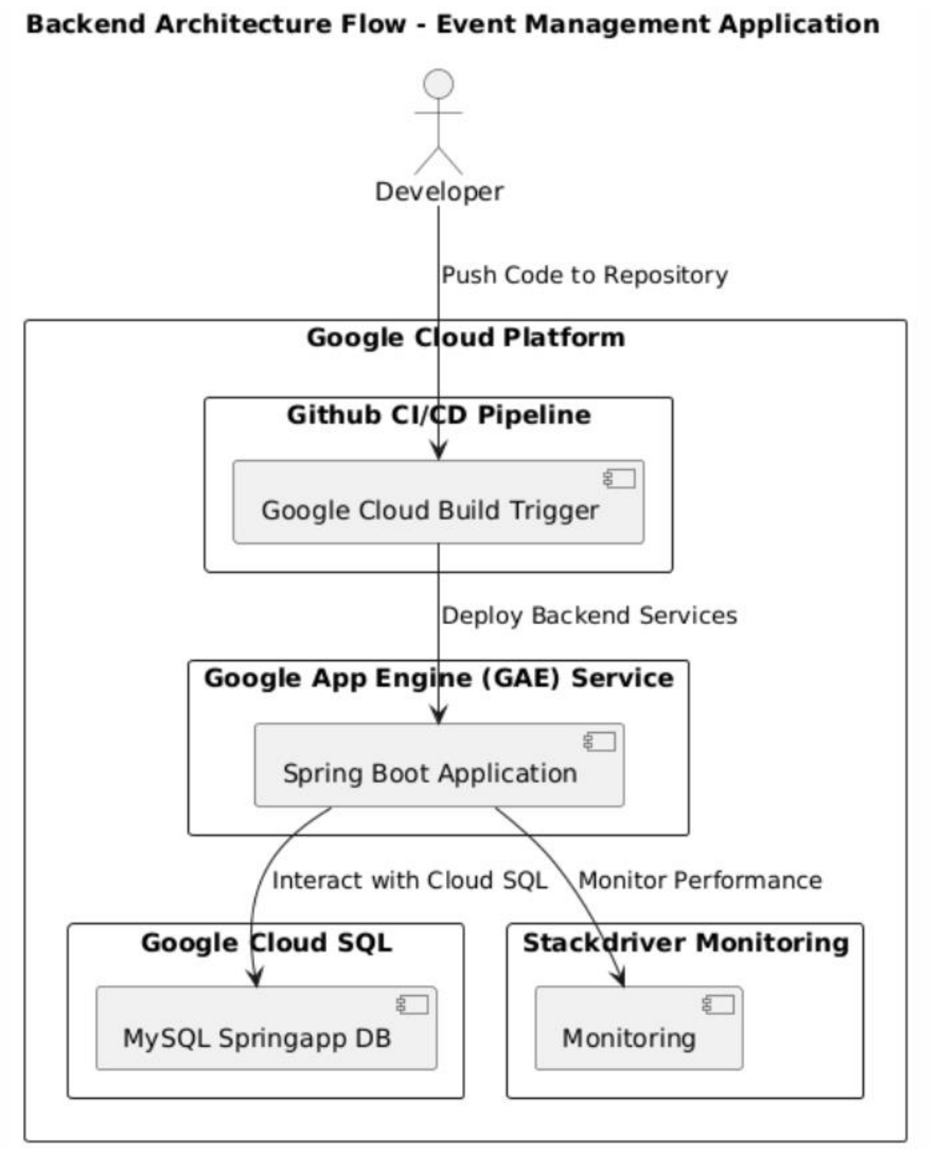
Architecture Diagram:



2.2 Backend Architecture

Google App Engine for Backend Services

- **Service Name**: springapp
- **Configuration**:
    - Deploy the backend application using Google App Engine for a fully managed, scalable environment.
    - Integrate with Cloud SQL for storing and retrieving event data.
    - Implement robust logging and monitoring with Stackdriver Logging and Monitoring.



Database Name: springdb

- **Configuration**:
  - Set up Cloud SQL with MySQL for managing relational event data.
  - Apply IAM roles to secure access and integrate with App Engine.
  - Use firewall rules for secure and efficient database access.

Stackdriver Logging and Monitoring

- **Configuration**:
  - Capture application logs for troubleshooting and monitoring performance.
  - Set up monitoring for application metrics to optimize performance and ensure system reliability.

2.3 Frontend Architecture

Google App Engine for Frontend Services

- **Service Name**: angularapp
- **Configuration**:
  - Deploy the frontend SPA using Angular on Google App Engine.
  - Retrieve configuration settings securely using Google Cloud Secret Manager.

Stackdriver Logging and Monitoring

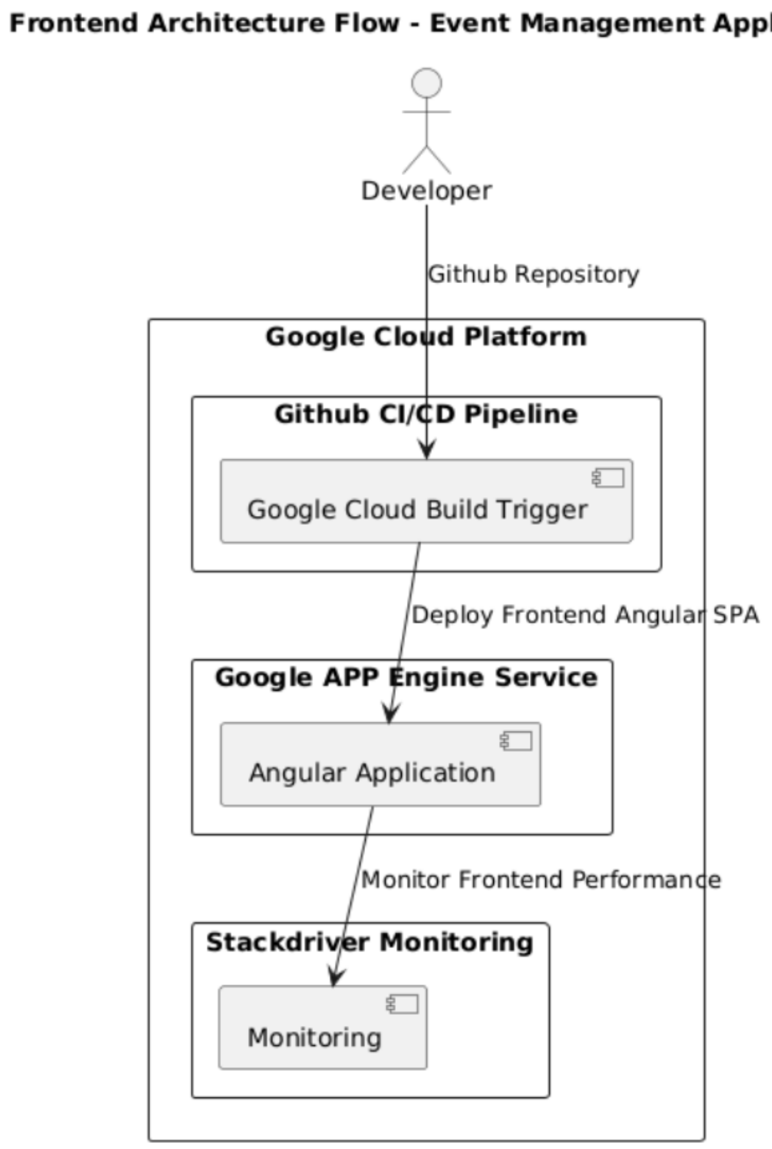
- **Configuration**:
  - Set up logging and monitoring for frontend performance and user activity tracking.

3. Specific Requirements

3.1 Backend Services

- **Deployment Steps**:
  - Configure Google Cloud Build to automate the deployment of backend services to App Engine.
  - Secure secrets using Google Cloud KMS.
  - Establish secure connections to Cloud SQL using VPC and IAM roles.
  - Integrate Stackdriver for logging and monitoring.

3.2 Frontend Services
Flow Diagram:


Frontend Architecture Flow - Event Management Application
Developer
Github Repository
Google Cloud Platform
Github CI/CD Pipeline
Google Cloud Build Trigger
Deploy Frontend Angular SPA
Google APP Engine Service
Angular Application
Monitor Frontend Performance
Stackdriver Monitoring
Monitoring

- **Deployment Steps**:
  - Configure Google Cloud Build to deploy the frontend SPA on App Engine.
  - Implement Firebase Authentication for secure user login.
  - Use Google Cloud Secret Manager for securely storing and retrieving API keys and configurations.

3.3 Logging and Monitoring

- **Configuration Steps**:
  - Use Stackdriver Logging to capture detailed application logs.
  - Set up Stackdriver Monitoring to track application performance and system health.

4. Testing and Deployment

- **Testing**:
    - Write unit and integration tests for backend services to ensure correct functionality.
    - Verify interaction between frontend and backend services.
    - Use Cloud Build for continuous integration and deployment to manage updates.

5. Conclusion

The Event Management Application on GCP provides a scalable, secure, and efficient solution for managing events. By leveraging GCP's managed services such as App Engine, Cloud SQL, Firebase Authentication, and Stackdriver, the application ensures high performance, robust security, and seamless user experience while adhering to industry standards and regulations.

Detailed Deployment Steps:

1. Set Up MySQL Database on Google Cloud SQL

1. **Create a MySQL Database Instance:**
    - Go to the Google Cloud Console.
    - Navigate to **SQL** > **Create Instance**.
    - Choose **MySQL** as the database type.
    - Set the **Instance ID** to springdb.
    - Select the **Region** as us-central1.
    - Configure other settings (e.g., machine type, storage) as needed.
    - Click **Create Instance**.
2. **Configure Database Schemas and Tables:**
    - Connect to the newly created MySQL instance via the **Cloud SQL** console or a MySQL client.

Configuration details:

- **App Engine Service 1 (Backend)**:
- **Name**: springapp
- **Region**: us-central1
- **Runtime**: Java 21

- **Environment**: standard
- **Instance Class**: F2
- **Version**: v1
- **App Engine Service 2 (Frontend)**:
- **Name**: angularapp
- **Region**: us-central1
- **Runtime**: Node.js 18
- **Environment**: standard
- **Instance Class**: F2
- **Version**: v1
- **Cloud SQL Instance Details**:
- **Name**: eventdb
- **Region**: us-central1
- **Cloud SQL Edition**: Enterprise
- **Machine Shape**: Shared core (1 vCPU, 0.614 GB)
- **Database Version**: MySQL 8.0.31
- **VM Instance**:
- **Name**: evnetvm
- **Region**: us-central1-a
- **Machine Type**: db-f1-micro
- **Boot Disk**: ubuntu-2004-focal-v20240731
- **Alert Policy**:
- **Name**: Policy_on_Memory_Usage
- **Metric Type**: GAE Application - Memory usage
- **Cloud Build Triggers**:
- **Trigger Name 1**: springtrigger (for Spring Boot backend)
- **Trigger Name 2**: angulartrigger (for Angular frontend)

- **Region**: us-central1
- **Runtime**: Node.js 18
- **Environment**: standard
- **Instance Class**: F2
- **Version**: v1
- **Cloud SQL Instance Details**:
- **Name**: eventdb
- **Region**: us-central1
- **Cloud SQL Edition**: Enterprise
- **Machine Shape**: Shared core (1 vCPU, 0.614 GB)
- **Database Version**: MySQL 8.0.31
- **VM Instance**:
- **Name**: eventvm
- **Region**: us-central1-a
- **Machine Type**: db-f1-micro
- **Boot Disk**: ubuntu-2004-focal-v20240731
- **Alert Policy**:
- **Name**: Policy_on_Memory_Usage
- **Metric Type**: GAE Application - Memory usage
- **Cloud Build Triggers**:
- **Trigger Name 1**: springtrigger (for Spring Boot backend)
- **Trigger Name 2**: angulartrigger (for Angular frontend)