Event Management API Documentation

This API allows you to manage events, attendees, and tasks associated with those events. It includes the ability to create, view, edit, and delete events, attendees, and tasks.

Prerequisites

Before running the application, ensure that you have the following installed:

1. **Node.js** (Version 14.x or above)
2. **PostgreSQL** (Database for storing event, attendee, and task data)

If you don't have Node.js or PostgreSQL installed, follow these steps:

Install Node.js:

- Download and install Node.js if you don't have it installed.

Install PostgreSQL:

- Download and install PostgreSQL.

Steps to Set Up and Run the API

1. Clone the Repository (or create a new project)

bash
Copy code

```
git clone https:https://github.com/snnath/Event-manager
cd Event-manager
```

2. Install Dependencies

Run the following command in the project directory to install required dependencies:

```
npm install
```

3. Set Up the Database

Create the PostgreSQL Database:
In your PostgreSQL command line or pgAdmin, create a database (for example, `event_management`):

```
CREATE DATABASE event_management;
```

1. **Set Up the Database Schema:**
   Create the necessary tables for events, attendees, and tasks. You can execute the following SQL queries in the PostgreSQL command line or pgAdmin.
   sql

```
     CREATE TABLE events (

    id SERIAL PRIMARY KEY,
    name VARCHAR(255) NOT NULL,
    description TEXT,
    location VARCHAR(255),
    date DATE,
    createdAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updatedAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE TABLE attendees (
    id SERIAL PRIMARY KEY,
    name VARCHAR(255) NOT NULL,
    task TEXT,
    event_id INTEGER REFERENCES events(id),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

```
CREATE TABLE tasks (
    id SERIAL PRIMARY KEY,
    name VARCHAR(255) NOT NULL,
    description TEXT,
    deadline DATE,
    status VARCHAR(20) DEFAULT 'Pending',
    event_id INTEGER REFERENCES events(id),
    attendee_id INTEGER REFERENCES attendees(id),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

2.

4. Configure Database Connection

In your project, ensure that the database connection is set up correctly in the `db.js` file. Here's an example of how the database connection might look:

```
const { Pool } = require('pg');

// Create a pool of database connections
const pool = new Pool({
  user: 'your-username',           // Replace with your
PostgreSQL username
  host: 'localhost',               // Your PostgreSQL host
(usually localhost)
  database: 'event_management',    // Database name
  password: 'your-password',       // Replace with your
PostgreSQL password
  port: 5432,                      // Default PostgreSQL port
});

module.exports = pool;
```

5. Run the Server

Once you have everything set up, start the server using the following command:

```
nodemon index
```

By default, the server will run on port 4000. You should see the following message if the server starts successfully:

```
Server has started on port 4000
```

6. Run the Client

```
npm start
```

API Endpoints

1. Create an Event

- **Endpoint:** `POST /events`
- **Description:** Creates a new event.

Request Body:

```
{
  "name": "Event Name",
  "description": "Event Description",
  "location": "Event Location",
  "date": "2024-12-25"
}
```

Response:

```
{
  "id": 1,
  "name": "Event Name",
  "description": "Event Description",
  "location": "Event Location",
  "date": "2024-12-25",
  "createdAt": "2024-12-23T00:00:00Z",
  "updatedAt": "2024-12-23T00:00:00Z"
}
```

---

2. Get All Events

- **Endpoint:** `GET /events`
- **Description:** Fetches all events.

Response:

```
[
  {
    "id": 1,
    "name": "Event Name",
    "description": "Event Description",
    "location": "Event Location",
    "date": "2024-12-25",
    "createdAt": "2024-12-23T00:00:00Z",
    "updatedAt": "2024-12-23T00:00:00Z"
  }
]
```

---

3. Get a Specific Event

- **Endpoint:** `GET /events/:id`
- **Description:** Fetches a specific event by its ID.

Response:

```
{
  "id": 1,
  "name": "Event Name",
  "description": "Event Description",
  "location": "Event Location",
  "date": "2024-12-25",
  "createdAt": "2024-12-23T00:00:00Z",
  "updatedAt": "2024-12-23T00:00:00Z"
}
```

---

4. Update an Event

- **Endpoint:** `PUT /events/:id`
- **Description:** Updates the details of a specific event.

Request Body:

```
{
  "name": "Updated Event Name",
  "description": "Updated Event Description",
  "location": "Updated Event Location",
  "date": "2024-12-30"
}
```

Response:

```json
{
  "message": "Event updated successfully",
  "updatedEvent": {
    "id": 1,
    "name": "Updated Event Name",
    "description": "Updated Event Description",
    "location": "Updated Event Location",
    "date": "2024-12-30",
    "createdAt": "2024-12-23T00:00:00Z",
    "updatedAt": "2024-12-23T00:00:00Z"
  }
}
```

---

5. Delete an Event

- **Endpoint:** `DELETE /events/:id`
- **Description:** Deletes a specific event by ID.

Response:

```json
{
  "message": "Event deleted successfully",
  "deletedEvent": {
    "id": 1,
    "name": "Event Name",
    "description": "Event Description",
    "location": "Event Location",
    "date": "2024-12-25",
    "createdAt": "2024-12-23T00:00:00Z",
    "updatedAt": "2024-12-23T00:00:00Z"
  }
}
```

---

6. Create an Attendee

- **Endpoint:** `POST /attendees`
- **Description:** Creates a new attendee.

Request Body:

```
{
  "name": "Attendee Name",
  "task": "Assigned Task",
  "event_id": 1
}
```

Response:

```
{
  "id": 1,
  "name": "Attendee Name",
  "task": "Assigned Task",
  "event_id": 1,
  "created_at": "2024-12-23T00:00:00Z",
  "updated_at": "2024-12-23T00:00:00Z"
}
```

---

7. Get All Attendees

- **Endpoint:** `GET /attendees`
- **Description:** Fetches all attendees with their associated event.

Response:

```
[
  {
    "id": 1,
    "task": "Assigned Task",
    "event_name": "Event Name"
  }
]
```

---

8. Delete an Attendee

- **Endpoint:** `DELETE /attendees/:id`
- **Description:** Deletes a specific attendee by ID.

Response:

```
{
  "message": "Attendee deleted successfully",
  "deletedAttendee": {
    "id": 1,
    "task": "Assigned Task",
    "event_name": "Event Name"
  }
}
```

9. Create a Task

Endpoint: `POST /tasks`
Description: `Creates a new task.`
Request Body:

```json
{
  "name": "Task Name",
  "description": "Task Description",
  "deadline": "2024-12-25",
  "event_id": 1,
  "attendee_id": 1
}
```

Response:

```json
{
  "id": 1,
  "name": "Task Name",
  "description": "Task Description",
  "deadline": "2024-12-25",
  "status": "Pending",
  "event_id": 1,
  "attendee_id": 1,
  "created_at": "2024-12-23T00:00:00Z",
  "updated_at": "2024-12-23T00:00:00Z"
}
```

10. Get All Tasks

Endpoint: `GET /tasks`
Description: Fetches all tasks.
Response:

```json
[
  {
    "id": 1,
    "name": "Task Name",
    "description": "Task Description",
    "deadline": "2024-12-25",
    "status": "Pending",
    "event_id": 1,
    "attendee_id": 1,
    "created_at": "2024-12-23T00:00:00Z",
    "updated_at": "2024-12-23T00:00:00Z"
  }
]
```

11. Update Task Status

Endpoint: `PUT /tasks/:id`
Description: Update the status of a task (Pending/Completed).
Request Body:

```
{
  "status": "Completed"
}
```

Response:

```
{
  "message": "Task updated successfully",
  "updatedTask": {
    "id": 1,
    "name": "Task Name",
    "description": "Task Description",
    "deadline": "2024-12-25",
    "status": "Completed",
    "event_id": 1,
    "attendee_id": 1,
    "created_at": "2024-12-23T00:00:00Z",
    "updated_at": "2024-12-23T00:00:00Z"
  }
}
```

12. Delete a Task

Endpoint: `DELETE /tasks/:id`
Description: Deletes a specific task by ID.
Response:

```
{
  "message": "Task deleted successfully",
  "deletedTask": {
    "id": 1,
    "name": "Task Name",
    "description": "Task Description",
    "deadline": "2024-12-25",
    "status": "Pending",
    "event_id": 1,
    "attendee_id": 1,
    "created_at": "2024-12-23T00:00:00Z",
    "updated_at": "2024-12-23T00:00:00Z"
  }
}
```

Notes

- Replace `localhost:4000` with your server's address if deploying remotely.
- You can test the API endpoints using tools like **Postman**