1. Setup Guide for Local and Docker Environments

Local Environment Setup Guide

Prerequisites:

Before starting, ensure you have the following installed on your local machine:

- **Node.js**: Node.js is a runtime environment for executing JavaScript outside the browser. This project relies on Node.js for backend processing.

Installation:

  - Download the latest stable version of Node.js from the official website.
  - Follow the installation instructions provided on the website.

- **npm (Node Package Manager)**: npm is used to manage and install dependencies for your project.

Installation:

  - npm comes preinstalled with Node.js. You can verify npm installation by running npm -v in your terminal.

- **Database** (MySQL/PostgreSQL/MongoDB): Depending on the database used by the project, ensure you have the corresponding database installed and running. The project should specify which database it is using.
  - **MySQL**:
    - Install MySQL from the official website, then start the MySQL server.
    - Use tools like MySQL Workbench or command line for database interaction.
  - **PostgreSQL**:
    - Install PostgreSQL from the official website and follow installation instructions.
    - You can use pgAdmin or the command line for database management.

Steps:

1. **Clone the repository**:
   - First, open a terminal or command prompt.
   - Clone the project’s Git repository using the following command:

```
git clone https://github.com/your-repo/project-name.git
```

   - After cloning, navigate into the project folder:

```
cd project-name
```

2. **Install dependencies**:
   - The project’s dependencies are listed in the package.json file. To install all required dependencies, run:

```
npm install
```

   - This will download and install all dependencies required by the project, ensuring that everything is ready for use.

3. **Configure environment variables**:
   - Environment variables are critical for keeping sensitive data (e.g., database credentials) secure. Create a .env file at the root of the project directory. This file will hold all necessary environment-specific variables.

Example .env file content:

ini

```
DB_HOST=localhost

DB_PORT=5432

DB_USER=youruser

DB_PASSWORD=yourpassword

DB_NAME=yourdbname
```

   - Replace the placeholders with actual values. These variables will be used to configure the database connection or other services in your application.

4. **Run the application locally**:
   - Now, you can start the application locally by running:

```
npm start
```

   - This will run the application in your local environment, typically on port 3000. You can access the application by navigating to:

arduino

```
http://localhost:3000
```

This should open the application in your browser, and you can interact with it as if it were deployed on a server.

Docker Environment Setup Guide

Prerequisites:

Before proceeding, ensure Docker is installed and running on your system.

- **Docker**: Docker allows you to package your application along with its dependencies into a container, ensuring it runs consistently across different environments.

Installation:

  - Download Docker from the official Docker website.
  - Follow the installation instructions for your operating system (Windows, macOS, or Linux).
  - Once installed, open Docker Desktop and ensure it is running.

Steps:

1. **Clone the repository**:
   - Open a terminal and clone the repository using Git:

```
git clone https://github.com/your-repo/project-name.git
```

   - Navigate to the project directory:

```
cd project-name
```

2. **Build the Docker image**:
   - In order to run the project in a Docker container, you need to build a Docker image from the project's Dockerfile (which contains the setup instructions for the container).
   - To build the image, use the following command:

```
docker build -t project-name .
```

   - Here, -t project-name tags the image with the name project-name, which can be used to refer to the image later. The . refers to the current directory where the Dockerfile is located.

3. **Create and start the container**:
   - Once the Docker image is built, you can create and start a container based on that image. To do this, use:

```
docker run -d -p 3000:3000 --name project-container project-name
```

- **Explanation**:
    - -d runs the container in detached mode (in the background).
    - -p 3000:3000 binds port 3000 on your local machine to port 3000 in the container, ensuring the application is accessible.
    - --name project-container assigns the container the name project-container.
    - project-name is the name of the image you built earlier.

4. **Check the logs**:
    - To ensure the container is running correctly, you can check the logs of the container using:

```
docker logs -f project-container
```

- The -f flag will stream the logs, allowing you to monitor the container's output in real time.

5. **Access the application**:
    - Finally, open your browser and navigate to:

```
http://localhost:3000
```

- This will load the application running inside the Docker container, just like in the local environment setup.

Important Docker Commands for Development

- **Stop the container**:

```
docker stop project-container
```

- **Remove the container**:

```
docker rm project-container
```

- **List running containers**:

```
docker ps
```

- **Remove unused Docker images**:

```
docker image prune -a
```

API Documentation Using Postman

Postman is a powerful API testing and documentation tool that allows you to create, test, and document your API endpoints. It can generate interactive API documentation that can be shared with your team and users.

1. Setting Up Postman for API Documentation

Prerequisites:

- **Postman** installed on your local machine.
    - You can download Postman from here.

---

2. Create a Postman Collection

A **Postman Collection** is a group of related API requests, and it allows you to organize your API tests and document them in one place.

Steps to Create a Collection:

1. **Open Postman**: Start Postman after installation.
2. **Create a New Collection**:
    - Click on the **Collections** tab in the left sidebar.
    - Click the **New Collection** button to create a new collection.
    - Provide a name for the collection, e.g., "Project API Documentation".
    - You can also provide a description of the collection to explain its purpose.

Example:

- **Collection Name**: Project API
- **Description**: This collection includes all the API endpoints for the Project, including authentication, user management, and event handling.

3. **Add Requests to the Collection**:
    - Inside your newly created collection, click **Add Request** to create a new API request.
    - Fill in the following details:
        - **Request Type**: GET, POST, PUT, DELETE, etc.
        - **URL**: The endpoint URL of the API (e.g., https://api.example.com/users).
        - **Headers**: Specify headers such as Content-Type, Authorization, etc.
        - **Body** (for POST/PUT requests): Provide JSON data, form data, etc.
4. **Save the Request**:
    - Once you've configured the request, click the **Save** button to add it to the collection.

5. **Test the API Requests**:
    - After saving the requests, you can click on the **Send** button in Postman to test them.
    - Check the **Status Code**, **Response Body**, and **Headers** to verify that the API is functioning as expected.

---

3. Documenting API Requests

To document your API, Postman allows you to add descriptions and comments for each API request. This makes it easy to explain what each endpoint does and how to use it.

Steps to Add Descriptions:

1. **Adding a Description to a Request**:
    - After creating a request in your collection, click on the request to open it.
    - In the request's window, you will see an option to add a description at the top.
    - Describe the request's purpose, any required parameters, sample inputs, and expected outputs.

Example:

- **Request**: POST /users
- **Description**:

pgsql

Copy code

This endpoint creates a new user in the system.

- **Body**: JSON object containing user details.

- **Example**:

```
 {
   "username": "john_doe",
   "email": "john@example.com",
   "password": "password123"
 }
```

2. **Adding Request Examples**:
    - You can add example requests and responses within the **Example** tab.
    - This is useful for demonstrating the expected format of the request body and the response.

Example:

- **Request Example**:

json

Copy code

```
{
  "username": "john_doe",
  "email": "john@example.com",
  "password": "password123"
}
```

- **Response Example**:

json

Copy code

```
{
  "id": 1,
  "username": "john_doe",
  "email": "john@example.com"
}
```

---

4. Organizing API Documentation

You can organize your Postman requests into folders within a collection to group similar API endpoints.

Steps to Organize Requests:

1. **Create Folders**:
   - Within your collection, click on the **New Folder** button.
   - Name the folder based on the logical grouping of endpoints (e.g., User Management, Event API, Authentication).
2. **Move Requests to Folders**:
   - Once a folder is created, you can drag and drop related API requests into the appropriate folder.
3. **Use Tags for Better Organization**:
   - When describing each request, you can also add **tags** to group them further. Tags can be used to identify API categories, such as **auth**, **users**, **events**, etc.

---

5. Generating API Documentation from Postman

Once you've added all the API requests and descriptions, you can generate interactive API documentation directly from Postman.

Steps to Generate Documentation:

1. **Click on the Collection**:
   - In the **Collections** tab, select the collection you want to document.
2. **Generate Documentation**:
   - Click on the **three dots** next to the collection name.
   - Select **Generate Documentation** from the dropdown.
3. **Configure Documentation Settings**:
   - You can configure the title, description, and other settings of your API documentation.
   - Postman will automatically generate and format the API documentation, including all your requests, descriptions, and example responses.
4. **Publish the Documentation**:
   - After generating the documentation, you can publish it to Postman's cloud, making it publicly available or sharing it with your team.
   - Click on the **Publish** button to get a shareable link or to make the documentation accessible to others.

---

6. Exporting Postman Collection

If you want to share your Postman collection with others, you can export it.

Steps to Export:

1. **Click on the Collection**:
   - In the **Collections** tab, click the three dots next to the collection name.
2. **Select Export**:
   - Choose **Export** to download the collection as a **JSON** file.
   - You can then share this file with others or import it into another instance of Postman.

---

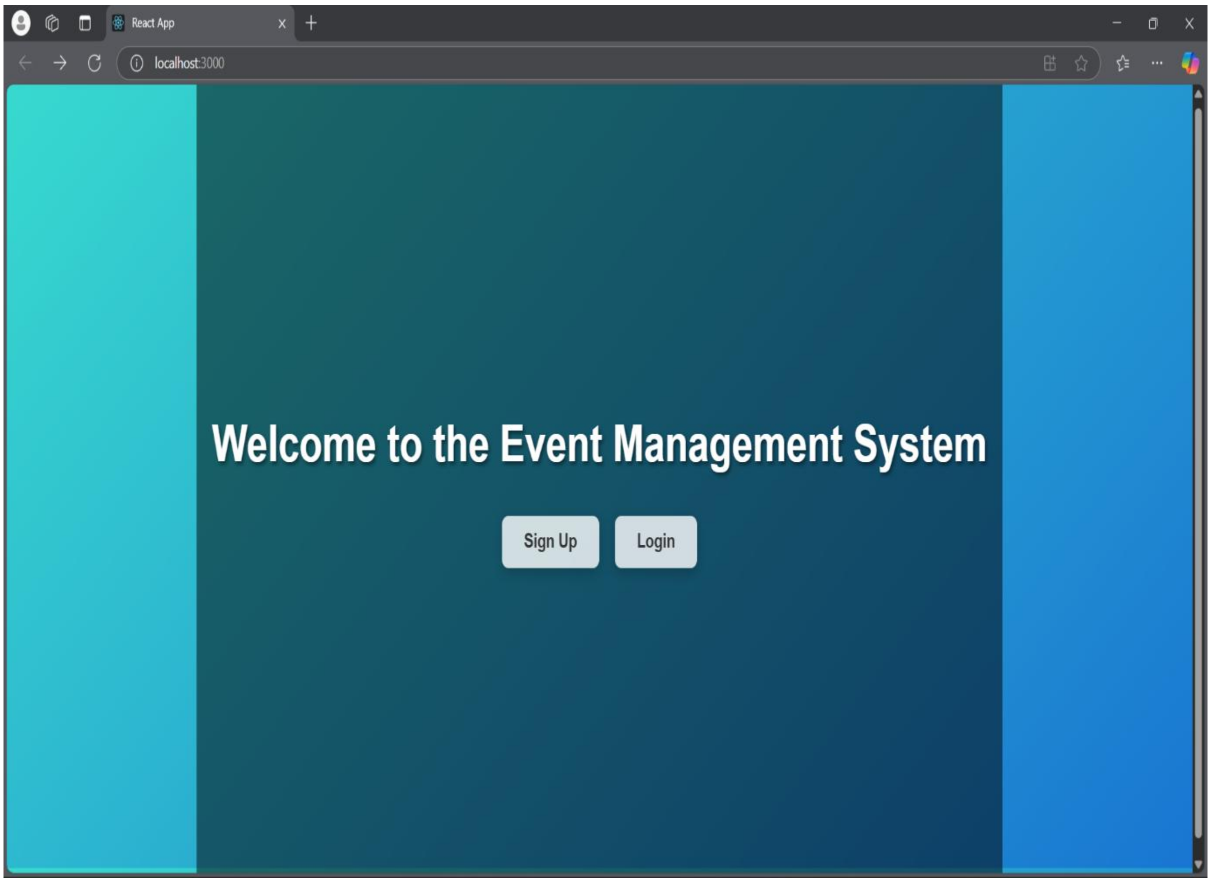
7. Example Postman API Documentation

GET /users

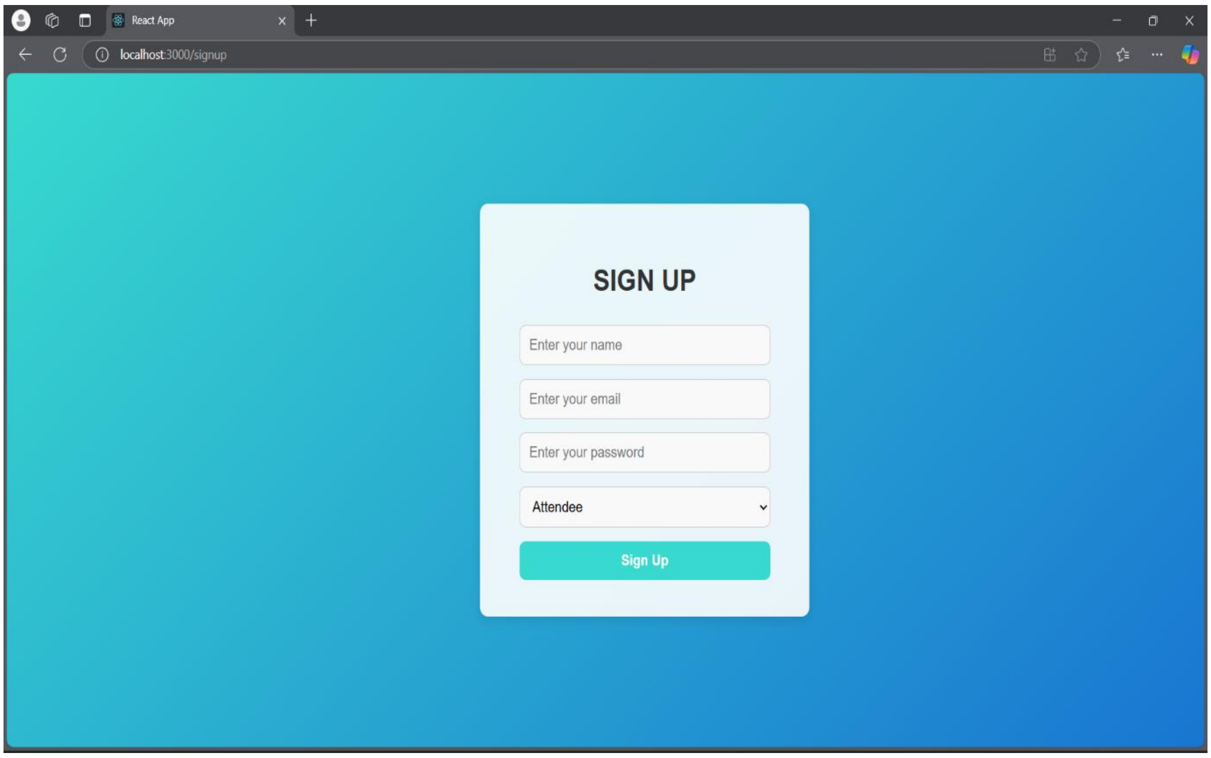
Description: Fetch all users.

- **Request**:
    - **Method**: GET
    - **URL**: https://api.example.com/users
    - **Headers**:
        - Authorization: Bearer your-token
- **POST /users**
    - **Description**: Create a new user.
    - **Request**:
    - **Method**: POST

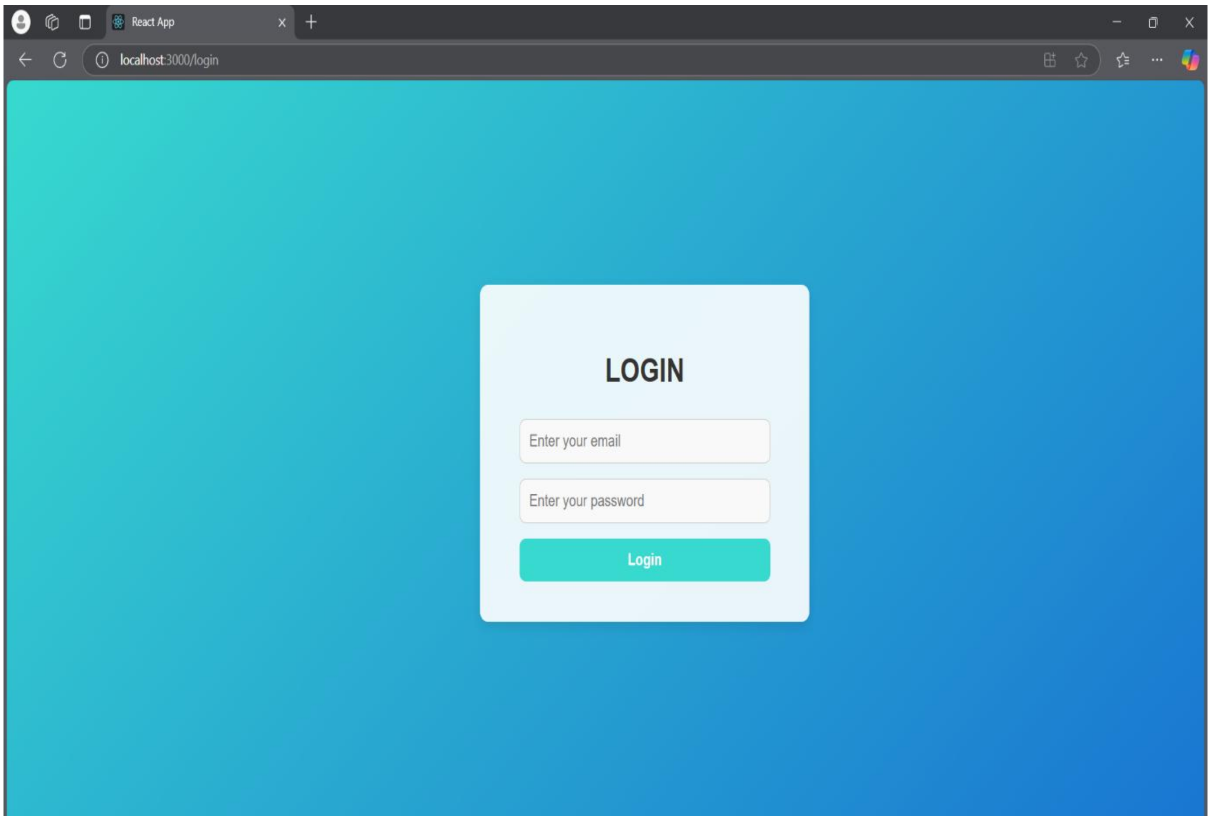
HOME PAGE



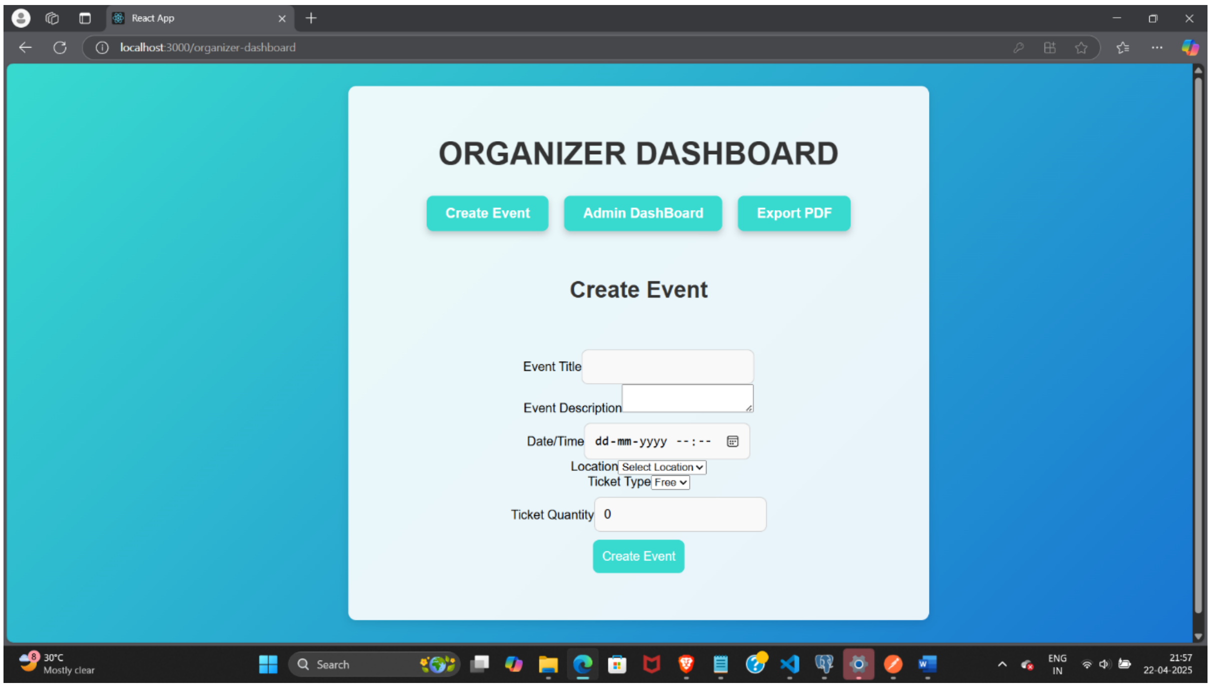
SIGN UP PAGE



LOGIN PAGE:



ORGANIZER – DASHBOARD


ORGANIZER DASHBOARD

Create Event | Admin DashBoard | Export PDF

Create Event

Event Title

Event Description

Date/Time dd-mm-yyyy --:--

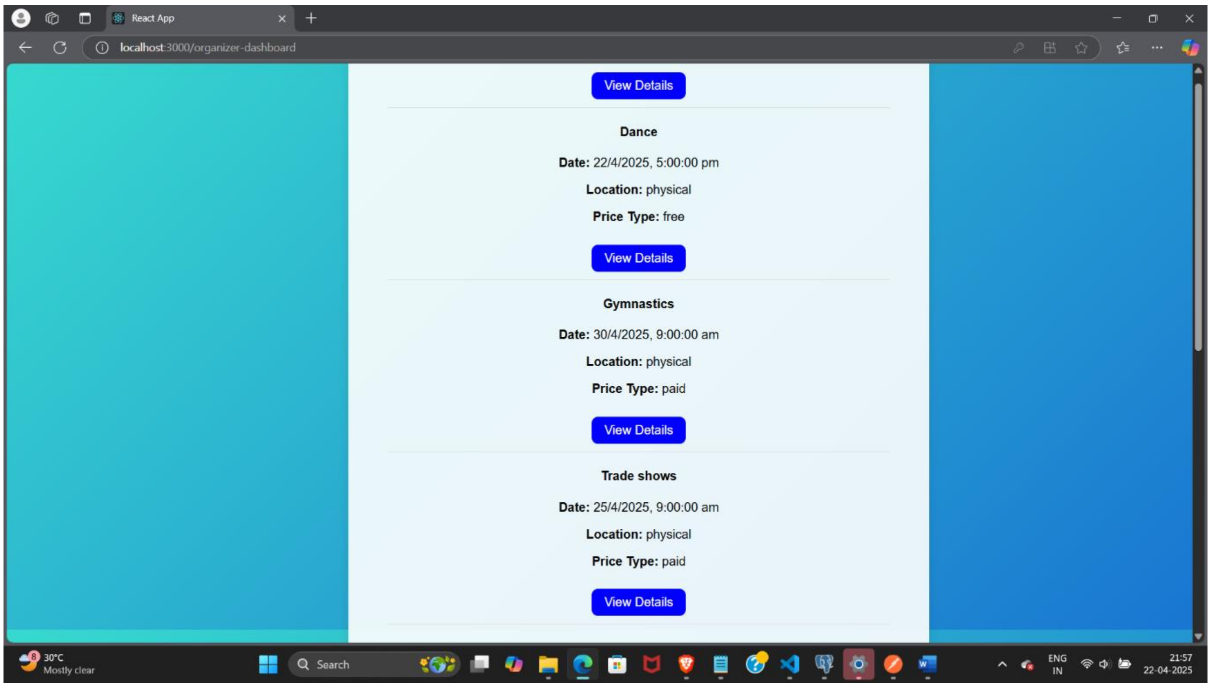
Location Select Location

Ticket Type Free

Ticket Quantity 0

Create Event

LIST OF EVENTS:


View Details

Dance

Date: 22/4/2025, 5:00:00 pm

Location: physical

Price Type: free

View Details

Gymnastics

Date: 30/4/2025, 9:00:00 am

Location: physical

Price Type: paid

View Details

Trade shows

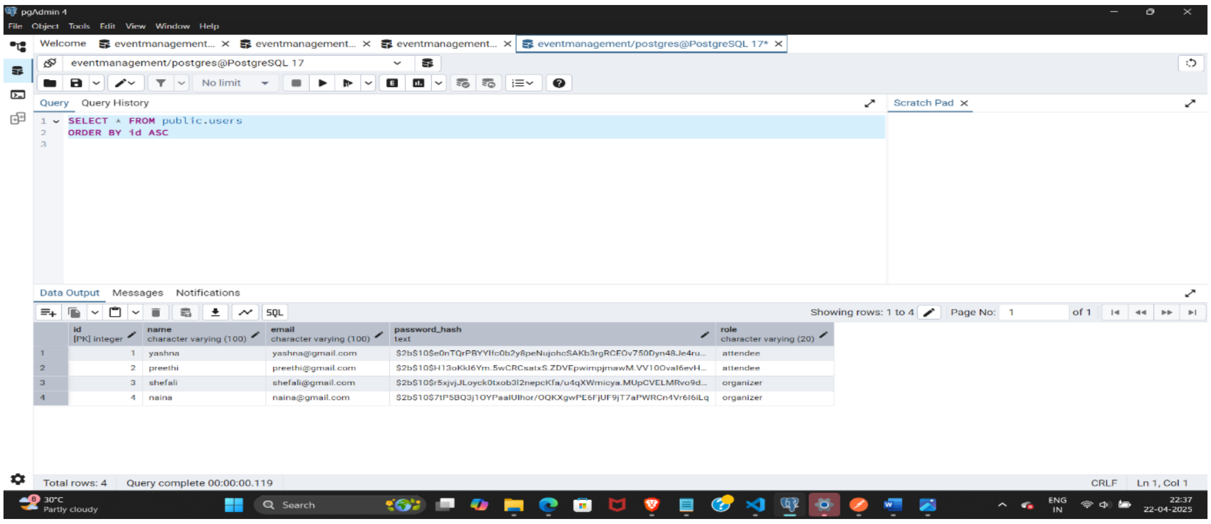
Date: 25/4/2025, 9:00:00 am

Location: physical

Price Type: paid

View Details

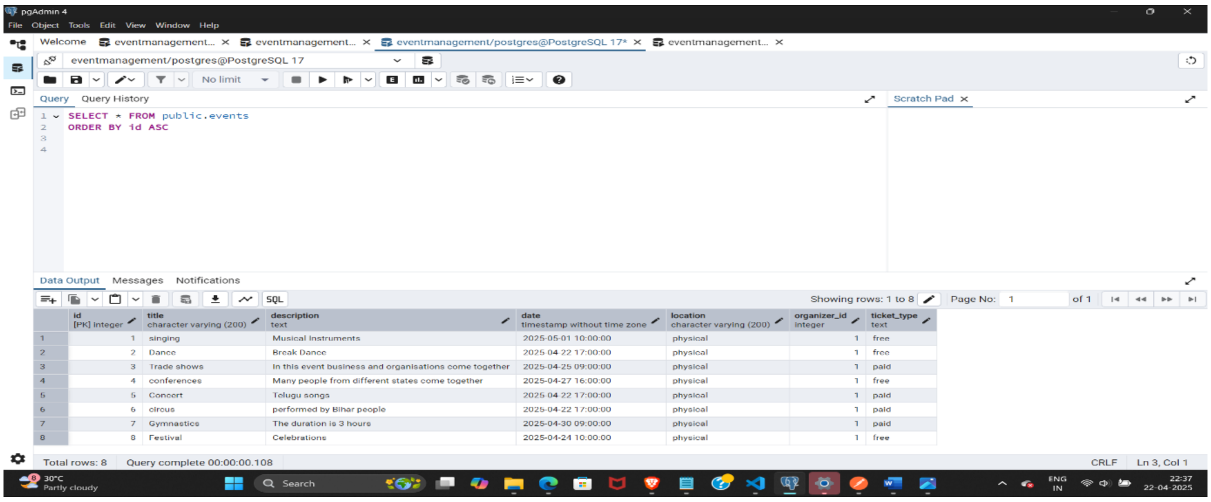
DATA BASE TABLES:



```sql
SELECT * FROM public.users
ORDER BY id ASC
```

	id [PK] integer	name character varying (100)	email character varying (100)	password_hash text	role character varying (20)
1	1	yashna	yashna@gmail.com	$2b$10$e0nTQrPBYYlfc0b2y8peNujohcSAKb3rgRCEOv750Dyn48.Je4ru...	attendee
2	2	preethi	preethi@gmail.com	$2b$10$H13oKkI6Ym.5wCRCsatxS.ZDVEpwimpjmawM.VV10Oval6evH...	attendee
3	3	shefali	shefali@gmail.com	$2b$10$r5xjvjJLoyck0txob3l2nepcKfa/u4qXWmicya.MUpCVELMRvo9d...	organizer
4	4	naina	naina@gmail.com	$2b$10$7tP5BQ3j1OYPaalUlhor/OQKXgwPE6FjUF9jT7aPWRCn4Vr6I6iLq	organizer

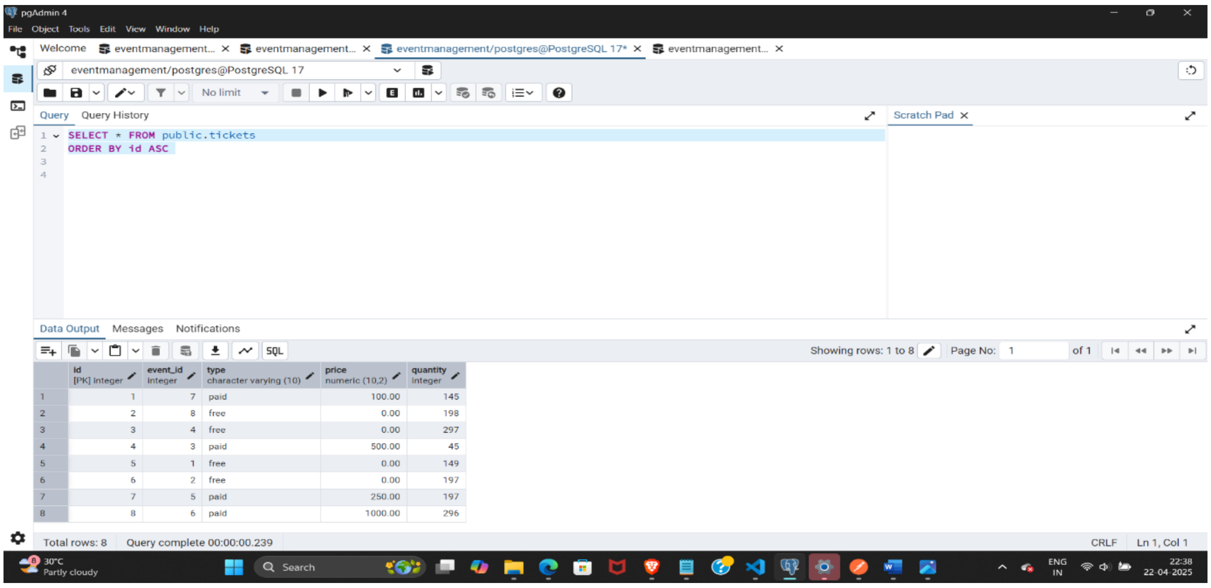
Total rows: 4 Query complete 00:00:00.119



```sql
SELECT * FROM public.events
ORDER BY id ASC
```

	id [PK] integer	title character varying (200)	description text	date timestamp without time zone	location character varying (200)	organizer_id integer	ticket_type text
1	1	singing	Musical Instruments	2025-05-01 10:00:00	physical	1	free
2	2	Dance	Break Dance	2025-04-22 17:00:00	physical	1	free
3	3	Trade shows	In this event business and organisations come together	2025-04-25 09:00:00	physical	1	paid
4	4	conferences	Many people from different states come together	2025-04-27 16:00:00	physical	1	free
5	5	Concert	Telugu songs	2025-04-22 17:00:00	physical	1	paid
6	6	circus	performed by Bihar people	2025-04-22 17:00:00	physical	1	paid
7	7	Gymnastics	The duration is 3 hours	2025-04-30 09:00:00	physical	1	paid
8	8	Festival	Celebrations	2025-04-24 10:00:00	physical	1	free

Total rows: 8 Query complete 00:00:00.108

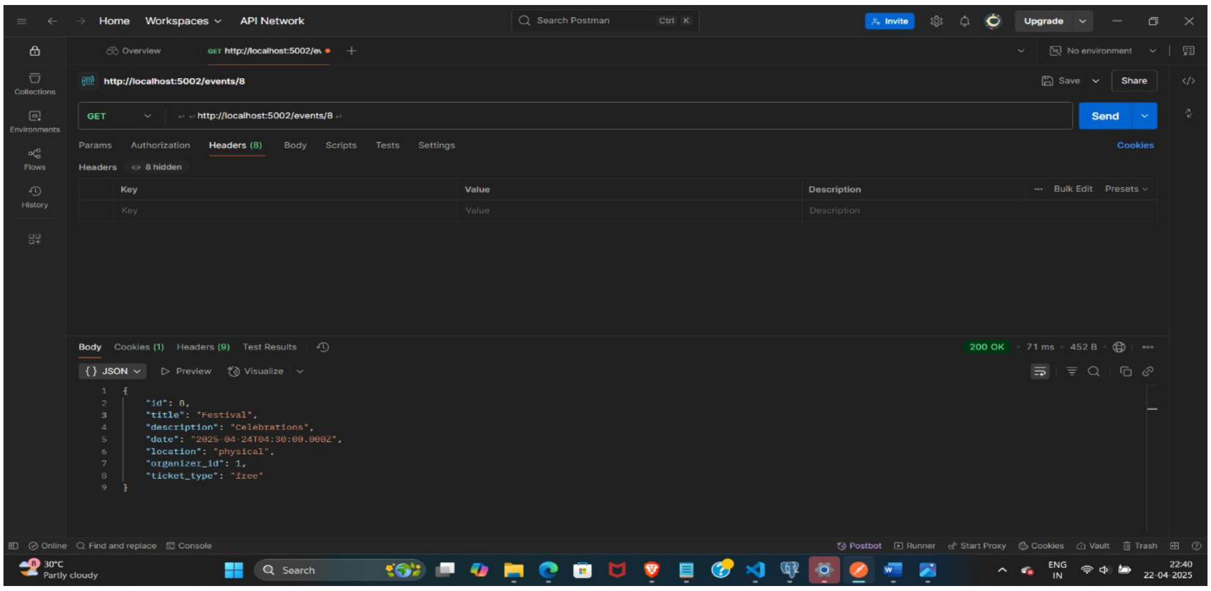


```sql
SELECT * FROM public.tickets
ORDER BY id ASC
```

	id [PK] integer	event_id integer	type character varying (10)	price numeric (10,2)	quantity integer
1	1	7	paid	100.00	145
2	2	8	free	0.00	198
3	3	4	free	0.00	297
4	4	3	paid	500.00	45
5	5	1	free	0.00	149
6	6	2	free	0.00	197
7	7	5	paid	250.00	197
8	8	6	paid	1000.00	296

Total rows: 8 Query complete 00:00:00.239

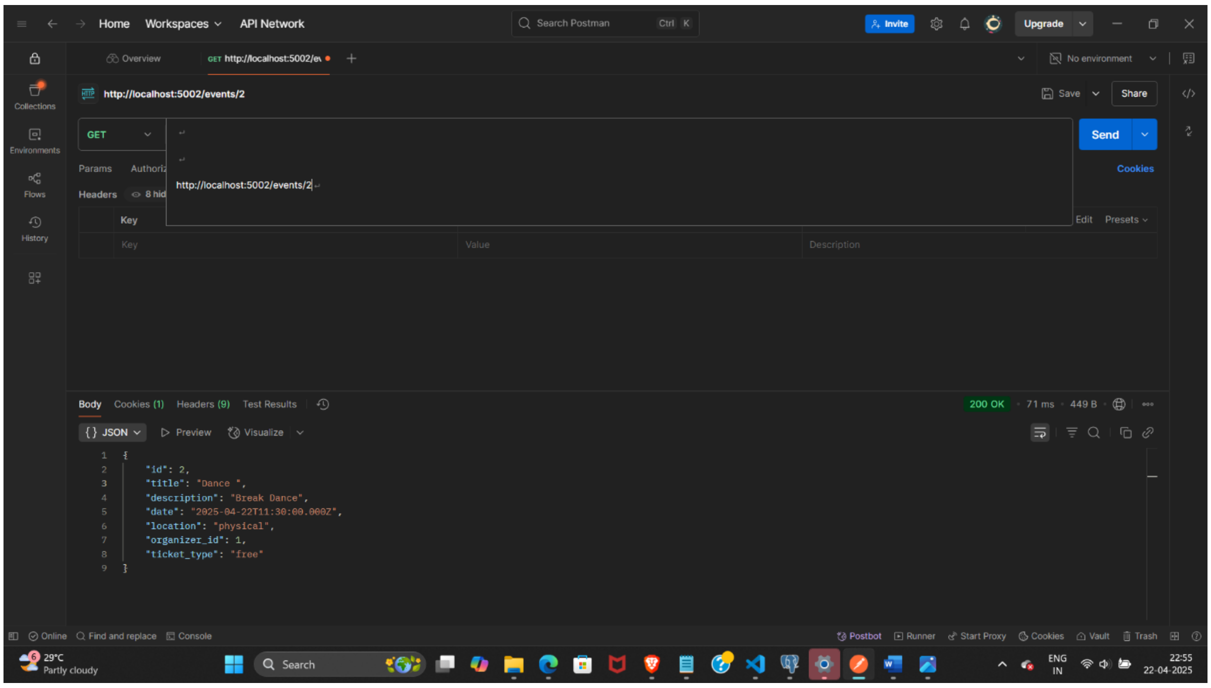
API CHECKING USING POSTMAN :



GET http://localhost:5002/events/8

200 OK · 71 ms · 452 B

```
{
    "id": 8,
    "title": "Festival",
    "description": "Celebrations",
    "date": "2025-04-24T04:30:00.000Z",
    "location": "physical",
    "organizer_id": 1,
    "ticket_type": "free"
}
```



GET http://localhost:5002/events/2

200 OK · 71 ms · 449 B

```
{
    "id": 2,
    "title": "Dance ",
    "description": "Break Dance",
    "date": "2025-04-22T11:30:00.000Z",
    "location": "physical",
    "organizer_id": 1,
    "ticket_type": "free"
}
```