Virtual Event Scheduler with Attendance Insights

Edgerunners

April 2025

A web application for managing events and tracking attendance

Source code available at `https://github.com/saumadeepsardar/Event-Scheduler`

Contents

1 Introduction

The Virtual Event Scheduler with Attendance Insights is a web application for creating, managing, and analyzing events. It includes a frontend (HTML, CSS, JavaScript) served at `http://localhost:5000` and a backend (Node.js with MySQL). This document guides users to set up the project locally, details the database structure, and explains key functionality.

2 System Overview

The system allows users to register, log in, create events, RSVP, check in, join waitlists, and submit feedback. The frontend communicates with a Node.js backend via REST APIs, using MySQL for data storage.

3 Database Structure

The `EventSchedulerPublic` database comprises six tables: `Users`, `Events`, `Event_RSVPs`, `Attendance`, `Waitlist`, and `Feedback`. Below are their attributes, relationships, and constraints.

3.1 Tables and Attributes

- **Users**: Stores user information.
  - `user_id`: INT, PRIMARY KEY, AUTO_INCREMENT.
  - `name`: VARCHAR(255), NOT NULL.
  - `email`: VARCHAR(255), UNIQUE, NOT NULL.
  - `password_hash`: VARCHAR(255), NOT NULL.
  - `role`: ENUM('attendee', 'organizer', 'admin'), DEFAULT 'attendee'.
  - `major`: VARCHAR(100), NULL.
  - `year`: VARCHAR(50), NULL.
- **Events**: Stores event details.
  - `event_id`: INT, PRIMARY KEY, AUTO_INCREMENT.
  - `title`: VARCHAR(255), NOT NULL.
  - `description`: TEXT, NULL.
  - `date_time`: DATETIME, NOT NULL.
  - `location`: VARCHAR(255), NULL.
  - `category`: VARCHAR(100), NULL.
  - `max_capacity`: INT, NULL.
  - `organizer_id`: INT, FOREIGN KEY (references `Users(user_id)`), NOT NULL.
  - `recurrence`: VARCHAR(50), NULL.

  - `check_in_code`: VARCHAR(50), NULL.

- **Event_RSVPs**: Tracks RSVPs.
  - `rsvp_id`: INT, PRIMARY KEY, AUTO_INCREMENT.
  - `event_id`: INT, FOREIGN KEY (references `Events(event_id)`), NOT NULL.
  - `user_id`: INT, FOREIGN KEY (references `Users(user_id)`), NOT NULL.
- **Attendance**: Records check-ins.
  - `attendance_id`: INT, PRIMARY KEY, AUTO_INCREMENT.
  - `event_id`: INT, FOREIGN KEY (references `Events(event_id)`), NOT NULL.
  - `user_id`: INT, FOREIGN KEY (references `Users(user_id)`), NOT NULL.
- **Waitlist**: Manages waitlisted users.
  - `waitlist_id`: INT, PRIMARY KEY, AUTO_INCREMENT.
  - `event_id`: INT, FOREIGN KEY (references `Events(event_id)`), NOT NULL.
  - `user_id`: INT, FOREIGN KEY (references `Users(user_id)`), NOT NULL.
  - `position`: INT, NOT NULL.
- **Feedback**: Stores event feedback.
  - `feedback_id`: INT, PRIMARY KEY, AUTO_INCREMENT.
  - `event_id`: INT, FOREIGN KEY (references `Events(event_id)`), NOT NULL.
  - `user_id`: INT, FOREIGN KEY (references `Users(user_id)`), NOT NULL.
  - `rating`: INT, NOT NULL.
  - `comments`: TEXT, NULL.

3.2 Relationships and Participation

- **Users - Events (organizes)**:
  - *Type*: One-to-Many (1:N). One user (organizer) can organize multiple events, but each event has one organizer.
  - *Participation*: Partial (not every user organizes events).
  - *Constraint*: `Events.organizer_id` references `Users.user_id`.
- **Users - Events (rsvps)**:
  - *Type*: Many-to-Many (M:N) via `Event_RSVPs`.
  - *Participation*: Partial (users can RSVP to zero or more events).
  - *Constraint*: `Event_RSVPs(event_id, user_id)` links `Events` and `Users`.
- **Users - Events (attends)**:
  - *Type*: Many-to-Many (M:N) via `Attendance`.

  - *Participation*: Partial.
  - *Constraint*: `Attendance(event_id, user_id)` links tables.

- **Users - Events (waitlists)**:
  - *Type*: Many-to-Many (M:N) via `Waitlist`.
  - *Participation*: Partial.
  - *Constraint*: `Waitlist(event_id, user_id)`.

- **Users - Events (provides feedback)**:
  - *Type*: Many-to-Many (M:N) via `Feedback`.
  - *Participation*: Partial.
  - *Constraint*: `Feedback(event_id, user_id)`.

3.3 ER Diagram

The ER diagram visualizes the schema.

4 Setup and Installation from GitHub

This section guides users to download and run the project locally on `http://localhost:5000`.

4.1 Prerequisites

Install:

- **Git**: `https://git-scm.com/downloads`.
- **Node.js**: Version 18+, with npm (`https://nodejs.org`).
- **MySQL**: Via:
  - Windows: XAMPP (`https://www.apachefriends.org`).
  - macOS: `brew install mysql`.
  - Linux: `sudo apt-get install mysql-server`.
- **Text Editor**: E.g., VS Code (`https://code.visualstudio.com`).
- A browser.

4.2 Step-by-Step Guide

1. **Clone the Repository**
   - Open a terminal.
   - Run:

     ```
     git clone <repository-url>
     ```

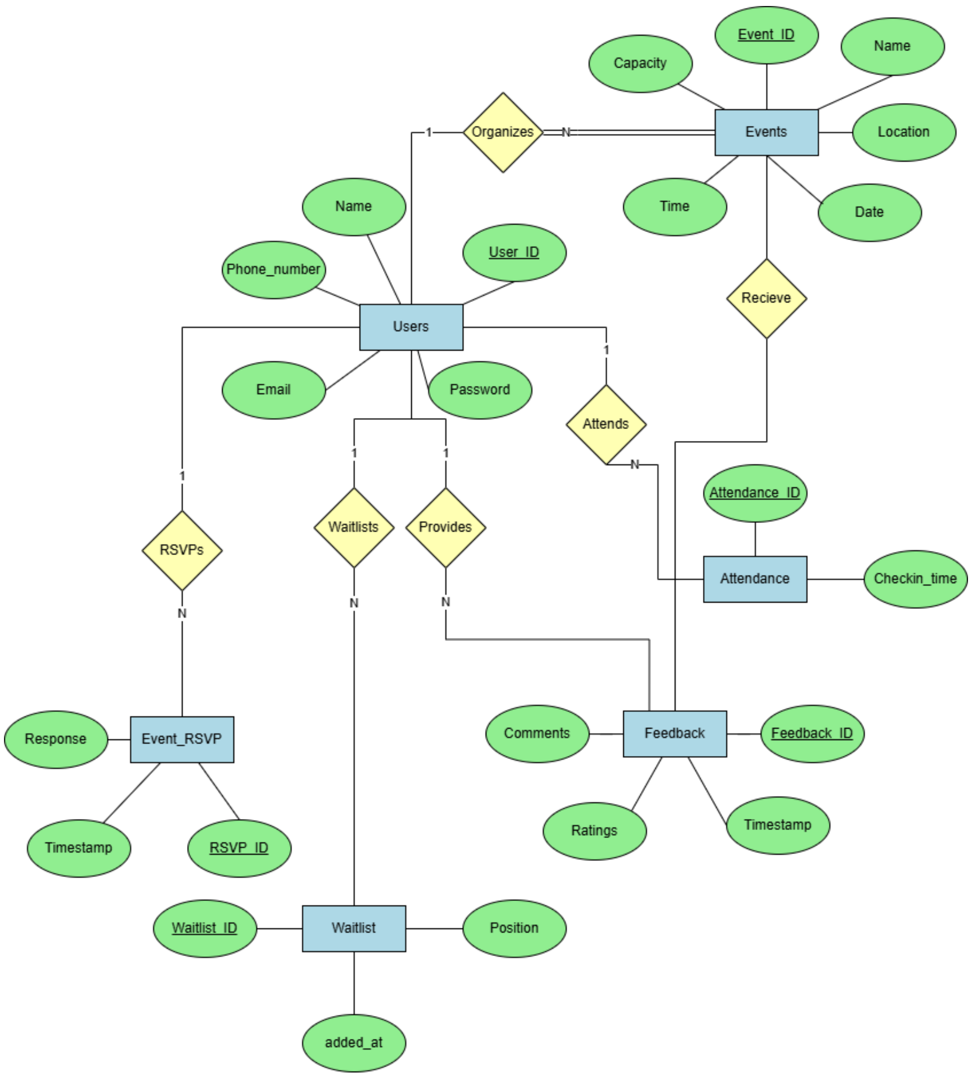

Figure 1: ER Diagram of EventSchedulerPublic Database

Replace `<repository-url>` with the GitHub URL.

- Navigate:

```
cd event-scheduler
```

Contains `frontend/` and `backend/`.

2. **Install Backend Dependencies**

- Go to:

```
cd backend
```

- Install:

```
npm install express mysql2 bcrypt jsonwebtoken cors
```

3. **Set Up MySQL**

- Start MySQL:
  - macOS:
    ```
    mysql.server start
    ```
  - Windows: Start MySQL in XAMPP.
  - Linux:
    ```
    sudo systemctl start mysql
    ```
- Log in:
  ```
  mysql -u root -p
  ```
  Enter password (or none).
- Create database:
  ```
  CREATE DATABASE EventSchedulerPublic;
  ```
- Create tables:
  ```
  USE EventSchedulerPublic;
  CREATE TABLE Users (
      user_id INT AUTO_INCREMENT PRIMARY KEY,
      name VARCHAR(255),
      email VARCHAR(255) UNIQUE,
      password_hash VARCHAR(255),
      role ENUM('attendee', 'organizer', 'admin') DEFAULT 'attendee',
      major VARCHAR(100),
      year VARCHAR(50)
  );
  CREATE TABLE Events (
      event_id INT AUTO_INCREMENT PRIMARY KEY,
      title VARCHAR(255) NOT NULL,
      description TEXT,
      date_time DATETIME NOT NULL,
      location VARCHAR(255),
      category VARCHAR(100),
      max_capacity INT,
      organizer_id INT,
      recurrence VARCHAR(50),
      check_in_code VARCHAR(50)
  );
  CREATE TABLE Event_RSVPs (
      rsvp_id INT AUTO_INCREMENT PRIMARY KEY,
      event_id INT,
      user_id INT
  );
  CREATE TABLE Attendance (
  ```

```
    attendance_id INT AUTO_INCREMENT PRIMARY KEY,
    event_id INT,
    user_id INT
);
CREATE TABLE Waitlist (
    waitlist_id INT AUTO_INCREMENT PRIMARY KEY,
    event_id INT,
    user_id INT,
    position INT
);
CREATE TABLE Feedback (
    feedback_id INT AUTO_INCREMENT PRIMARY KEY,
    event_id INT,
    user_id INT,
    rating INT,
    comments TEXT
);
```

- Add test data:

```
INSERT INTO Users (name, email, password_hash, role)
VALUES ('Admin', 'admin@example.com', '$2b$10$examplehash', 'admin');
INSERT INTO Events (title, date_time, location, category, max_capacity,
                    organizer_id, check_in_code)
VALUES ('Test Event', '2025-04-20 10:00:00', 'Online', 'Workshop', 50, 1, 't
```

- Exit:

```
EXIT;
```

4. **Configure Backend**

- Ensure backend/db.js:

```
const mysql = require('mysql2/promise');
module.exports = mysql.createPool({
    host: 'localhost',
    user: 'root',
    password: '', // Update with MySQL password
    database: 'EventSchedulerPublic'
});
```

- Verify server.js includes:

```
const db = require('./db');
app.use(cors());
app.listen(5000, () => console.log(`Server running on
   port 5000`));
```

5. **Start Backend**

- Run:

  ```
  node server.js
  ```

- Expect:

  ```
  Server running on port 5000
  ```

- Test:

  ```
  http://localhost:5000/api/v1/events
  ```

  Should return `[]` or events.

6. **Set Up Frontend**

   - Navigate:

     ```
     cd ../frontend
     ```

   - Install:

     ```
     npm install -g http-server
     ```

     Use `sudo` if needed.

   - Start:

     ```
     http-server -p 5000
     ```

   - Open `http://localhost:5000`.

7. **Verify**

   - Check events at `http://localhost:5000`.
   - Developer Tools (F12):
     - `Network`: `api/v1/events` should be `200`.
     - `Console`: No errors.

8. **Troubleshooting**

   - **Backend Fails**: Reinstall dependencies or fix `db.js`.
   - **Database**: Verify MySQL and tables.
   - **CORS**: Use `http-server`.
   - **Events Error**: Check `Events` table.

5 Key Code and User Flow

5.1 Important Code Snippets

- **Frontend: Fetching Events** (`script.js`)

```
async function loadDashboard() {
    const eventGrid = document.getElementById('eventGrid');
    const response = await fetch('http://localhost:5000/api/v1/events');
    const events = await response.json();
    // Render events to eventGrid
}
```

  Fetches events and displays them.

- **Backend: Events Endpoint** (`server.js`)

```
app.get('/api/v1/events', async (req, res) => {
    const [rows] = await db.execute(`
        SELECT e.*, COUNT(r.rsvp_id) as rsvp_count, u.name as organizer
        FROM Events e
        LEFT JOIN Event_RSVPs r ON e.event_id = r.event_id
        LEFT JOIN Users u ON e.organizer_id = u.user_id
        GROUP BY e.event_id
    `);
    res.json(rows);
});
```

  Queries events with RSVP counts.

- **Database Connection** (`db.js`)

```
const mysql = require('mysql2/promise');
module.exports = mysql.createPool({
    host: 'localhost',
    user: 'root',
    password: '',
    database: 'EventSchedulerPublic'
});
```

5.2 User Flow

1. **Registration/Login**: Users access `http://localhost:5000`, register (`POST /api/v1/register`
   or log in (`POST /api/v1/login`).

2. **View Events**: The dashboard calls `loadDashboard`, fetching `/api/v1/events` to display events.

3. **RSVP**: Users click RSVP, sending `POST /api/v1/rsvp`.

4. **Create Event**: Organizers use a form to send `POST /api/v1/events`.

5. **Check-In**: Attendees submit `check`$_{in_c}odevia$`POST /api/v1/checkin`.`Feedback` : $Userssubmitratings$

6 Adding ER Diagram to the Website

To display er$_{d}iagram.pngonthewebsite$:

6. Copy Image:

- Place er$_{d}iagram.pngin$frontend/($e.g.,$ frontend/images/).

  Update HTML:

  - In index.html or a new page (e.g., about.html):

```html
<div class="container">
    <h2>Database Schema</h2>
    <img src="images/er_diagram.png" alt="ER Diagram"
       style="max-width: 100%;">
</div>
```

  Style (Optional):

  - In styles.css:

```css
.container {
    text-align: center;
    margin: 20px;
}
img {
    border: 1px solid #ccc;
    padding: 10px;
}
```

  Serve:

  - Run http-server -p 5000 in frontend/.
  - Verify at http://localhost:5000/about.html.

  Test:

  - Ensure the image loads without 404 errors (check DevTools).

- Adjust src path if needed (e.g., ./er$_{d}iagram.png$).