Mini Project Report: Event Management Website

Date: 26-06-2025

Abstract

This mini project presents a **full stack Event Management Website** developed using modern web technologies.

- The application allows users to **create, view, and RSVP to events** through an intuitive interface.
- It demonstrates integration of **frontend and backend frameworks**, secure user authentication, and REST API development.
- The project reflects essential skills in **database management, UI/UX design, and deployment** of a live web application.

Objective

To design and implement a dynamic event management platform that simplifies event creation and participation.

- Strengthen skills in **building RESTful APIs** and connecting them with a responsive frontend.
- Develop **secure user authentication** and authorization using JWT.
- Gain experience in **deploying full stack applications** on cloud platforms.

Software Requirements

- **Frontend:** React.js / Next.js for building interactive, component-based UIs.
- **Backend:** Node.js + Express for handling API routes and server logic.
- **Database:** MongoDB (NoSQL) / PostgreSQL (SQL) for storing event and user data.
- **Styling:** Tailwind CSS / Bootstrap for responsive and clean design.

- **Other:** JWT for secure auth, Socket.io for optional real-time updates, deployment via Vercel + Render.

🚀 Key Featuress

💻 User Interface

- **Event Listings:** Home page displays upcoming events with title, date, and location in card format.
- **Event Details Page:** Clicking an event shows full description, organizer info, and RSVP option.
- **Create/Edit Events:** Authenticated users can create, update, or delete their events.

🔗 API Integration

- **RESTful APIs:** CRUD operations on events and RSVPs using JSON data exchange.
- **Modular Routes:** Separate routes for events, users, and RSVPs to ensure clean codebase.
- **Pagination & Sorting:** Backend supports paginated event lists and sort by date or popularity.

🔒 Authentication

- **User Registration/Login:** New users can sign up and securely log in.
- **JWT Tokens:** Protect routes for creating or editing events — only authorized users allowed.
- **Password Hashing:** User passwords are hashed before storing in the database for security.

⚠ Error Handling

- **Input Validation:** Frontend + backend checks (e.g., no past dates for events).
- **Graceful Failures:** Handles API, database, and auth errors with user-friendly messages.
- **Unauthorized Access:** Returns clear error for invalid or expired tokens.

💻 Sample Code (Simplified Backend Route)

```
// eventRoutes.js
const express = require('express');
const router = express.Router();
const { verifyToken } = require('./authMiddleware');
const Event = require('./models/Event');

// Create a new event (protected route)
router.post('/create', verifyToken, async (req, res) => {
 try {
  const { title, description, date, location } = req.body;

  // Basic validation
  if (!title || !description || !date || !location) {
   return res.status(400).json({ error: 'All fields are required' });
  }

  if (new Date(date) < new Date()) {
   return res.status(400).json({ error: 'Event date must be in the future' });
  }

  // Create and save event
  const event = new Event({
   title,
   description,
   date,
   location,
   organizer: req.user.id // From JWT payload
  });
```

```
    await event.save();
    res.status(201).json({ message: 'Event created successfully', event });

  } catch (error) {
    console.error('Create Event Error:', error);
    res.status(500).json({ error: 'Server error while creating event' });
  }
});

router.get('/', async (req, res) => {
  try {
    const events = await Event.find().sort({ date: 1 });
    res.json(events);
  } catch (error) {
    console.error('Fetch Events Error:', error);
    res.status(500).json({ error: 'Server error while fetching events' });
  }
});


router.post('/:eventId/rsvp', verifyToken, async (req, res) => {
  try {
    const event = await Event.findById(req.params.eventId);
    if (!event) {
      return res.status(404).json({ error: 'Event not found' });
    }

    if (event.attendees.includes(req.user.id)) {
      return res.status(400).json({ error: 'Already RSVP’d to this event' });
    }

    event.attendees.push(req.user.id);
    await event.save();

    res.json({ message: 'RSVP successful', event });
  } catch (error) {
    console.error('RSVP Error:', error);
    res.status(500).json({ error: 'Server error while RSVPing' });
  }
});
```

📂 Supporting Files

✅ `authMiddleware.js` → Verifies JWT and attaches user to `req.user`.
✅ `Event.js` (Mongoose Model) → Schema includes:

```
{
  title: String,
  description: String,
  date: Date,
  location: String,
  organizer: { type: ObjectId, ref: 'User' },
  attendees: [{ type: ObjectId, ref: 'User' }]
}
```

☑ Conclusion

The **Event Management Website** demonstrates how full stack development can address real-world problems like managing events and attendees.

- It showcases **API creation, frontend-backend communication**, and **secure user management**.
- The project strengthens skills in **designing user-centric web apps** with modern technologies.