Charter Document

Campus Events Application – MITAOE

# Purpose

* To create a centralized platform for all MITAOE student clubs and departments to announce and manage their events.
* Allows students to browse, register, and get notified about upcoming events in one place.
* Supports event creation, updates, and filtering by category, club, or date.
* Enhances student engagement, simplifies event management, and promotes a digital campus experience.

# Objective

To streamline the event management and registration process for MITAOE by:

* Providing a unified portal for all clubs to post upcoming events
* Allowing students to discover and register for club events or large-scale college events.
* Supporting category-wise filtering (e.g., cultural, sports, tech)
* Sending notifications and reminders
* Tracking user registrations and participation history
* Offering a user-friendly interface

# Demand / Opportunity

**Institutional Demand**

Currently, event announcements in MITAOE are decentralized (e.g., posters, social media, WhatsApp groups). There is a strong need for a unified platform where:

* Clubs can manage and promote their events
* Students can easily view and register for events
* College admins can track engagement and participation

This project supports digital transformation and streamlines event operations campus-wide.

# Business Requirement:

* **User Authentication:** Secure login/registration using email/password.
* **Event Management (CRUD):** Organizers can create, read, update, and delete events.
* **Categorization & Tagging:** Events are organized under categories and searchable tags.
* **Event Registration:** Users can register for an event and receive confirmation.
* **Real-Time Notifications:** Users get notifications when registered for the event.
* **Search & Filters:** Efficient search and filter options past, registered, upcoming events.
* **Responsive Interface:** Fully functional and optimized UI for all device types.

## **Technical Requirement**

### ● User Authentication

○ UI for secure email/password login and registration  
○ API routes for student and club admin sign-up, login, logout, and session management  
○ Passwords securely hashed and stored in the backend using encryption techniques .  
○ Session-based authentication system for protected routes and role management (Student, Club Admin)

### ● Event Management (CRUD)

○ API endpoints to create, read, update, and delete events by authorized club admins  
○ Event fields include title, description, date, time, venue, category, poster.  
○ UI for club admins to manage their event listings through a dashboard  
○ Validation for event details before submission

### ●Event Registration System

○ UI for students to register for any event with a single click  
○ Backend logic to track registered users for each event  
○ Email confirmation using crons sent to students after successful registration  
○ Dashboard to view upcoming and past registered events

### ● Search and Filtering

○ Backend logic for searching events by upcoming, past, registered events.  
○ Frontend filters to allow users to refine events by category, club, or registration status  
○ Sorting options based on upcoming date or popularity

### ● Real-Time Notifications ○ Notifications display on email after registering for an event.

### ● Data Storage

○ Secure and scalable **SQL** database for storing user profiles, event details, registration lists, and favourites.  
○ Collection design to handle different user roles, event categories, and Nakshatra data.

## **Technological Requirement**

**🖥️Frontend**

* **React.js** – For building an interactive, component-based user interface.
* **Tailwind CSS** – For a modern, responsive, and easily customizable design system.

**🛠️ Backend**

* **Node.js + Express.js** – For creating RESTful APIs that handle user authentication, event creation, registrations, and more.
* **Nodemailer + CRON Jobs** – Used to schedule and automatically send:
  + Event registration confirmation emails
  + Club communications and reminders
* **Forget Password Feature** – Implemented using **Crypto** for secure token generation and **Nodemailer** with Gmail SMTP for sending reset password emails.

**🗄️ Database**

* **SQL** – A relational database used to store:
  + User information
  + Event and club details
  + Registration history
  + Authentication tokens (e.g., password reset)

**☁️ Hosting & Deployment**

* **Vercel** – For deploying and hosting the frontend application.
* **Render** – For deploying and hosting the backend server and APIs.

**🧪 Development & Testing Tools**

* **Visual Studio Code** – Main code editor used for development.
* **GIT & GitHub** – For version control and team collaboration.
* **Postman** – To test REST APIs and validate request/response cycles.
* **SQL GUI Tools** *(e.g., phpMyAdmin, TablePlus)* – For visualizing and managing the SQL database manually.

# Stakeholder:

|  |  |  |
| --- | --- | --- |
| **Role** | **Name(s)** | **Count** |
| Developers | Aditi, Shravani, Sayali | 3 |
| DB Designer | Sayali | 1 |
| Testers | Pooja, Janhavi, Shravani P, Jayesh Raut | 3 |
| Project Management | Aditi | 1 |
| Documentation | Shravani (Creator), Sayali (Reviewer) | 2 |
| Hosting Provider | Render, Vercel | 1 |
| Event Admins | All MITAOE clubs | - |
| Students (Users) | All MITAOE students | - |

Resources Needed:

**🔧 Development Stack**

* React.js – For building the frontend user interface.
* Tailwind CSS – For responsive and customizable UI design.
* Express.js – For creating backend REST APIs and handling server-side logic.
* MongoDB – For storing user data, event details, and registration records.
* Nodemailer – For sending:
* Event registration confirmation emails
* Password reset emails using Gmail SMTP (with Crypto for secure token generation)
* Scheduled reminders using Node-Cron

**🧪 Dev & Testing Tools**

* Postman – For testing and validating API endpoints.
* Git & GitHub – For version control and seamless team collaboration.
* Visual Studio Code – Main code editor used for both frontend and backend development.

**☁️ Hosting & Deployment**

* Vercel – For deploying and hosting the frontend React application.
* Render – For deploying and hosting the backend Node.js/Express APIs.
* Human Resource

|  |  |  |
| --- | --- | --- |
| **Role** | **Count** | **Name(s)** |
| UI/UX Designer | 1 | Shravani Ingle |
| Frontend Developer | 2 | UI Development – Shravani, API Integration – Sayali |
| Backend Developer | 1 | Sayali |
| DB Designer | 1 | Shravani |
| Project Management | 1 | Aditi |
| Documentation | 2 | Creator – Shravani , Reviewer – Sayali Deshmukh |
| Tester | 2 | Pooja, Janhavi, Shravani P, Jayesh Raut |

# PESTEL Analysis:

* Political
* No direct government constraints for a college-level internal platform.
* Requires alignment with MITAOE’s institutional IT and data usage policies.
* Economic
* Utilizes a low-cost, open-source tech stack (React, Node.js, SQL) suitable for student projects.
* Affordable hosting via platforms like Vercel (frontend) and Render (backend) helps keep operational costs minimal.
* Social
* Encourages student engagement by centralizing event discovery and registration.
* Enhances the visibility and reach of club events and college fests like Nakshatra.
* Technological
* Built using scalable and well-supported technologies (React, Node.js, SQL).
* Mobile-responsive UI ensures accessibility across various devices.
* Uses CRON jobs with Nodemailer for scheduled email notifications.
* Environmental
* Promotes a paperless culture by digitizing event promotion and registration.
* Legal
* Must ensure secure storage of student data and protect user credentials.
* Requires user consent for receiving email notifications and follows basic data privacy practices.

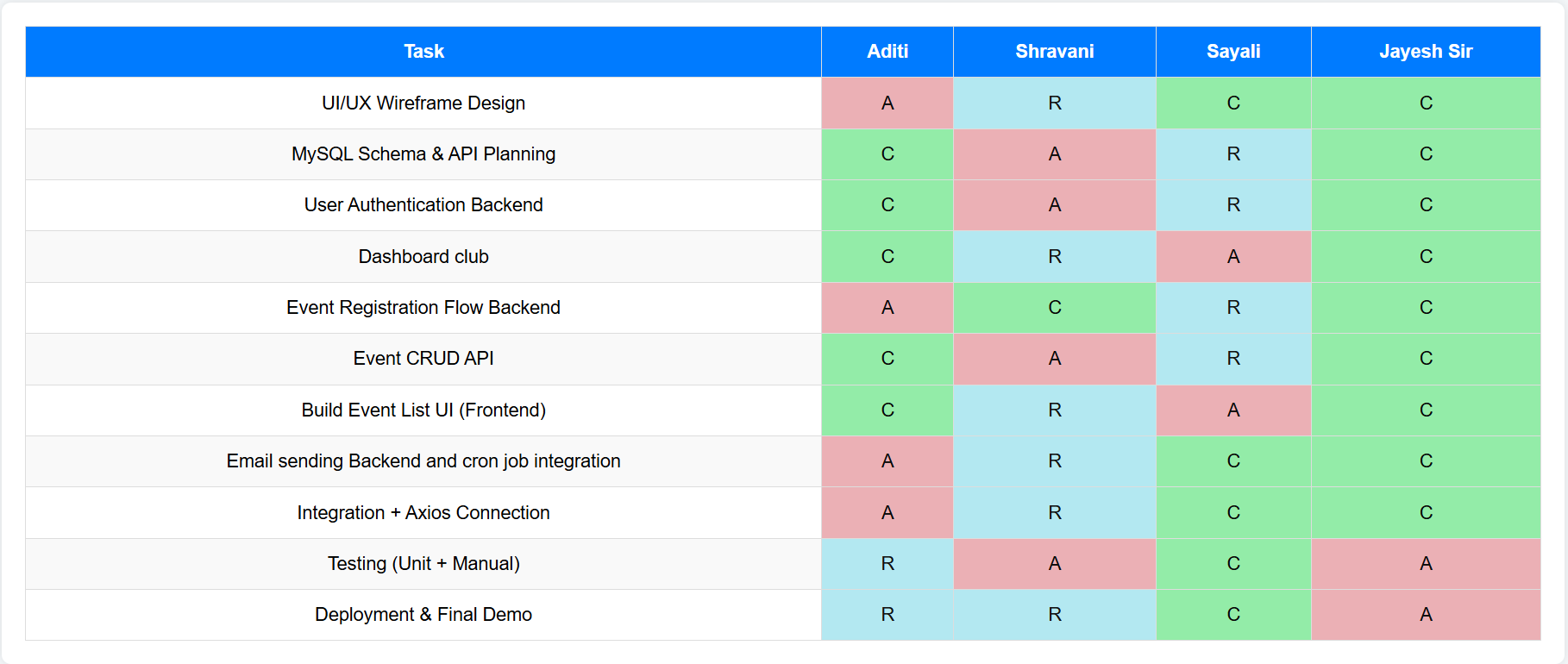
Risk Analysis:

|  |  |  |
| --- | --- | --- |
| **Risk** | **Description** | **Mitigation** |
| **Low Adoption** | • Students may not know about the platform.  • They may continue using posters/WhatsApp. | • Promote via college campaigns.  • Introduce it during club orientations. |
| **Data Misuse** | • Organizers may access data from other clubs.  • Risk of unintentional data exposure. | • Implement role-based access.  • Restrict access per user role. |
| **Feature Overload** | • Too many features can confuse users.  • MVP may get delayed or buggy. | • Focus on essential features first.  • Plan extras for later versions. |
| **Server Overload** | • App may lag or crash during large events.  • Increased simultaneous usage can cause downtime. | • Use pagination & caching.  • Optimize backend queries. |
| **Security Vulnerabilities** | • Risk of unauthorized admin access.  • Sensitive data could be compromised. | • Use session and input validation. • Maintain secure logs and headers. |

Timeline / Milestone:

|  |  |  |
| --- | --- | --- |
| **Phase** | **Tasks (Detailed)** | **Timeline** |
| **Requirement & Planning** | • Identify all stakeholders (students, club admins).  • Gather feature requirements (event posting, registration, filtering, etc.).  • Create user flow diagrams and design wireframes for key screens (event list, event creation, registration, dashboard). | **Week 1** |
| **Backend Development** | • Set up Express.js project structure.  • Develop RESTful APIs for event CRUD, user registration/login, and student-event registration.  • Implement session-based authentication and role-based access (student, club admin).  • Integrate Node mailer with cron jobs for confirmation emails. | **Week 2** |
| **DB & API Design** | • Design database schema for users, events, registrations.  • Establish entity relationships (e.g., user ↔ event, club ↔ event).  • Define API endpoints for user auth, event management, and registration workflows. | **Week 3–4** |
| **Frontend Development** | • Build UI components using React.js and Tailwind CSS (event list, registration form, admin dashboard, event creation/editing form).  • Add filtering, search, and category selection features. | **Week 5–6** |
| **Integration & Testing** | • Connect frontend with backend APIs using Axios/fetch.  • Test complete workflows: event creation, registration.  • Perform unit testing and manual testing across different roles and scenarios.  • Fix bugs and improve UX. | **Week 7** |
| **Final Deployment** | • Deploy the frontend on Vercel.  • Deploy backend on Render.  • Configure environment variables (API keys, DB connection).  • Conduct live testing and final review before presentation/submission. | **Week 8** |

**RACI Chart:**

****

* **R = Responsible (executes the task)**
* **A = Accountable (owns the outcome)**
* **C = Consulted (input provider)**