**A**

**Major Project Report**

**On**

**“COLLEGE EVENTS HUB”**

Submitted in partial fulfillment of the

Requirements for the award of the degree of

**Bachelor of Technology**

**In**

**Information Technology**

**By**

**TIRUMALA SANDEEP KUMAR – 20R21A1254**

**KURUVA RITHEN KUMAR – 20R21A1242**

**GONGATI VENKATA SUMANTH – 20R21A1223**

Under the guidance of

**D. Sandeep**

**Assistant Professor**



**2024**

**Department of Information Technology**



**Department of Information Technology**

CERTIFICATE

This is to certify that the project entitled **“College Events Hub”** has been submitted by **TIRUMALA SANDEEP KUMAR (20R21A12A9), KURUVA RITHEN KUMAR (20R21A1276), GONGATI VENKATA SUMANTH (20R21A1223)** in the partial fulfillment of the requirements for the award of degree of Bachelor of Technology in Information Technology from Jawaharlal Nehru Technology University, Hyderabad. The results embodied in this project have not been submitted to any other University or Institution for the award of any degree or diploma.

Supervisor Head of the Department

Mr D.Sandeep Dr. N.V Rajasekhar Reddy

External Examiner

I



**Department of Information Technology**

# DECLARATION

We hereby declare that the project entitled **“College Events Hub”** is the work done during the period from **July 2023 to April 2024** and is submitted in the partial fulfillment of the requirements for the award of degree of Bachelor of Technology in Information Technology from Jawaharlal Nehru Technology University, Hyderabad. The results embodied in this project have not been submitted to any other University or Institution for the award of any degree or diploma.

**Tirumala Sandeep Kumar (20R21A12A9)**

**Kuruva Rithen Kumar (20R21A1276)**

**Gongati Venkata Sumanth (20R21A1223)**

II



**Department of Information Technology**

# ACKNOWLEDGEMENT

There are many people who helped us directly and indirectly to complete our project successfully. We would like to take this opportunity to thank one and all. First of all we would like to express our deep gratitude towards our supervisor **Mr. D.Sandeep, Assistant Professor, Department of Information Technology** for her support in the completion of our dissertation. We wish to express our sincere thanks to **Dr. N.V RAJASEKHAR REDDY, HOD, Department of IT** and also to Principal **Dr.K.SRINIVAS RAO** for providing the facilities to complete the dissertation.

We would like to thank all our faculty, coordinators and friends for their help and constructive criticism during the project period. Finally, we are very much indebted to our parents for their moral support and encouragement to achieve goals.

**Tirumala Sandeep Kumar (20R21A12A9)**

**Kuruva Rithen Kumar (20R21A1276)**

**Gongati Venkata Sumanth (20R21A12A9)**

III



**Department of Information Technology**

# ABSTRACT

College Events Hub is a cutting-edge platform designed to revolutionize how college clubs and events are managed and experienced. Serving as a comprehensive solution, it bridges communication divides between club members, leaders, coordinators, and the broader student body. Through its intuitive interface, users gain access to a wealth of information about both college-hosted and external events, ensuring they stay up-to-date with the latest announcements and schedules. Administrators enjoy secure access through a robust authentication system, safeguarding sensitive data and allowing for effective management of the platform.

At its core, College Events Hub streamlines event management by offering detailed information and facilitating smooth registration with an integrated payment system. Organizers can effortlessly coordinate activities while attendees enjoy a hassle-free registration process, promoting greater participation and engagement in college activities. Furthermore, the platform fosters interaction and feedback through its event-specific comment sections, empowering users to connect, share insights, and build community. By incorporating student ratings and reviews, College Events Hub not only informs future event decisions but also facilitates continuous improvement, ultimately enhancing the overall college experience.

IV

# CONTENTS

| **S. No** | **Contents** | **Page No** |
| --- | --- | --- |
|  | Certificate | I |
|  | Declaration | II |
|  | Acknowledgement | III |
|  | Abstract | IV |
| 1 | Introduction | 1-3 |
| 1.1 | Overview |  |
| 1.2 | Purpose of the project |  |
| 1.3 | Motivation |  |
| 2 | Literature Survey | 4-6 |
| 2.1 | Existing System |  |
| 2.2 | Disadvantages of Existing System |  |
| 3 | Proposed System | 7-11 |
| 3.1 | Advantages of Proposed System |  |
| 3.2 | System requirements |  |
| 4 | System Design | 12-17 |
| 4.1 | Proposed System Architecture |  |
| 4.2 | Modules |  |
| 5 | Implementation and Installation | 18-37 |
| 5.1 | Algorithm |  |
| 5.2 | Implementation Steps |  |
| 5.3 | Source Code |  |
| 6 | DFD Diagrams | 38-40 |
| 7 | Testing | 41-44 |
| 7.1 | Introduction |  |
| 7.2 | Unit Testing |  |
| 7.3  7.4 | Integration Testing Validation  Closure of Testing and Validation |  |
| 8 | Results | 47-52 |
| 9 | Conclusion | 53 |
| 10 | References | 54-55 |
|  |  |  |
|  |  |  |
|  |  |  |

# LIST OF FIGURES AND TABLES

| **Figure**  **Number** | **Name of the Figure** | **Page**  **Number** |
| --- | --- | --- |
| 1 | System Architecture | 14 |
| 2 | Server | 15 |
| 3 | Data Flow Diagrams | 38-40 |
| 4 | User Login Page | 47 |
| 5 | Home page | 48 |
| 6 | Events menu | 49 |
| 7 | Events modify | 50-51 |
| 8 | Payment page | 52 |

# INTRODUCTION

## OVERVIEW

In the vibrant realm of college life, managing events effectively plays a pivotal role in fostering community engagement, collaboration, and student enrichment. To address this essential need, College Events Hub emerges as a powerful, full-stack platform tailored specifically for organizing and participating in college events with ease.

Utilizing Next.js 14 as its foundation, College Events Hub combines advanced technology with user-friendly design to offer a comprehensive solution perfectly suited for the collegiate environment. It goes beyond simple event organization, serving as a central hub for communication and interaction among various clubs, leaders, coordinators, and the student body.

Powered by a diverse tech stack including Node.js, TypeScript, and TailwindCSS, College Events Hub integrates seamlessly with essential tools like Stripe for secure payment processing and React Hook Form for smooth data management. This ensures a seamless experience for administrators and users alike.

At its core, College Events Hub empowers administrators to manage events effortlessly, enabling them to create, edit, and remove events as needed. Meanwhile, students benefit from an intuitive interface that provides detailed event information, facilitates easy registration, and even allows for event ratings and feedback submission.

In essence, College Events Hub transcends traditional event management on college campuses. It enriches the college experience by promoting efficient communication, engagement, and community building. With its innovative features and user-friendly design, College Events Hub redefines the way colleges approach event organization, fostering a more vibrant and connected campus culture.

## PURPOSE OF THE PROJECT

The primary purpose of the College Events Hub project is to revolutionize event management within the college setting. By consolidating club activities and event information into a centralized platform, the project aims to simplify communication between club members, leaders, coordinators, and students. This platform serves as a comprehensive solution, offering up-to-date event details, secure registration processes, and avenues for feedback and interaction, ultimately fostering a more connected and engaged college community.

Through its user-friendly interface and robust features, College Events Hub seeks to streamline event planning and execution, making it easier for organizers to coordinate activities and for attendees to participate. By promoting efficient communication and facilitating seamless event registration, the project aims to enhance the overall college experience, encouraging greater involvement in campus events and fostering a sense of belonging among students. Overall, the purpose of the project is to create a dynamic and inclusive environment where students can easily discover, participate in, and contribute to a diverse array of college activities and Events.

Furthermore, the overarching purpose of the College Events Hub project extends beyond mere event management to encompass the broader goal of enriching the college experience. By promoting efficient communication, facilitating seamless event registration, and fostering interaction and feedback, the project aims to cultivate a vibrant and inclusive campus culture. Ultimately, the project seeks to create an environment where students feel empowered to explore their interests, engage with their peers, and actively contribute to the dynamic tapestry of college life. Through its comprehensive approach to event management and community building, College Events Hub endeavors to elevate the college experience and leave a lasting impact on the campus.

## MOTIVATION

The motivation behind the development of the College Events Hub project stems from a recognition of the inherent challenges and inefficiencies associated with managing club activities and events within the college ecosystem. Traditional methods of communication and event organization often rely on fragmented systems, leading to confusion, missed opportunities, and administrative overhead. Understanding the importance of fostering a vibrant and engaged campus community, there was a clear motivation to create a centralized platform that addresses these shortcomings and streamlines the entire event management process.

Moreover, the desire to enhance the overall college experience for students served as a significant motivational factor. Recognizing that extracurricular activities and events play a vital role in shaping students' holistic development, there was a strong impetus to provide a solution that not only facilitates easier event coordination but also promotes greater participation and engagement. By offering a user-friendly platform that empowers students to discover, register for, and provide feedback on events, the project aims to create a more dynamic and inclusive campus environment where students feel connected and supported in their extracurricular endeavors.

# LITERATURE SURVEY

### INTRODUCTION

Event management within educational institutions has garnered significant attention in academic research due to its vital role in enhancing student engagement, fostering community spirit, and promoting campus culture. Existing literature offers insights into various aspects of event management within colleges and universities, shedding light on best practices, challenges, and opportunities for improvement.

Communication and Collaboration: Studies highlight the importance of effective communication and collaboration among students, faculty, and administrators in the planning and execution of college events (Reinhardt et al., 2015; Kirillova et al., 2017). Fragmented communication channels and lack of collaboration platforms have been identified as barriers to successful event management, emphasizing the need for centralized communication tools and collaborative platforms to streamline coordination efforts (Ho and McKechnie, 2016; Ratten, 2018).

Technology Integration: The integration of technology in event management has emerged as a key trend, enabling colleges to automate administrative tasks, enhance communication, and improve participant engagement (Al-Ansari and Ismail, 2017; Kononova et al., 2019). Studies emphasize the importance of user-friendly interfaces, mobile accessibility, and integrated payment systems in enhancing the effectiveness of event management platforms (Bowen et al., 2016; Yeh et al., 2018).

Student Engagement and Satisfaction: Research indicates a strong correlation between student engagement in campus events and overall satisfaction with the college experience (Astin, 1984; Pascarella and Terenzini, 2005). Engaging events that cater to diverse interests and preferences contribute to a sense of belonging and connectedness among students, thereby enhancing retention rates and academic success (Kuh, 2001; Tinto, 2012).

Payment Security and Trust: The security of online payment systems in event management is a growing concern, with studies highlighting the importance of trust, transparency, and data security in fostering participant confidence (Wang et al., 2019; Cheung et al., 2020). Secure payment gateways, encryption protocols, and compliance with data protection regulations are essential for mitigating risks and ensuring participant trust in online transactions (Chen and Lu, 2015; Tan and Teo, 2016).

Overall, the literature underscores the importance of effective communication, technology integration, student engagement, payment security, and feedback mechanisms in successful event management within educational institutions. By drawing upon these insights, the College Events Hub project aims to address the key challenges identified in the literature and contribute to the advancement of event management practices within college campuses.

* 1. **EXISTING SYSTEM**

Before the implementation of the College Events Hub project, the existing system for managing club activities and events within the college was often fragmented and inefficient. Communication among club members, leaders, coordinators, and students typically relied on disparate channels such as email threads, social media groups, and physical notice boards. This decentralized approach often led to miscommunication, missed deadlines, and a lack of centralized information, resulting in frustration and confusion for all parties involved.

Event coordination and registration processes were also cumbersome and time-consuming under the existing system. Organizers had to manually collect registrations, manage attendee lists, and handle payment transactions through offline methods, leading to administrative overhead and potential errors. Additionally, there was limited opportunity for students to provide feedback or engage with event organizers beyond the initial registration process, resulting in a missed opportunity to enhance the overall event experience and foster community interaction.

Overall, the existing system for managing college events lacked cohesion and efficiency, hindering effective communication, event coordination, and student engagement. Recognizing these shortcomings, there was a clear need for a centralized platform that could streamline event management processes, improve communication channels, and enhance the overall college experience for students. This realization served as a driving force behind the development and implementation of the College Events Hub project.

## DISADVANTAGES OF EXISTING SYSTEM:

**Fragmented Communication Channels:**

* Communication is spread across various platforms such as email, social media, and physical notices.
* Lack of centralized communication leads to information getting lost or overlooked.
* Results in miscommunication, confusion, and delays in event planning and execution.
* Manual Event Coordination Processes:
* Organizers rely on paper-based forms or excel sheets for event coordination.
* Collecting registrations, managing attendee lists, and tracking payments are labor-intensive.
* Time-consuming process, especially for large-scale events, leading to inefficiencies and logistical challenges.

**Limited Student Engagement and Feedback:**

* Once registration is complete, there's limited opportunity for students to engage further with events.
* Lack of mechanisms for students to provide feedback, ask questions, or interact beyond the initial.
* Missed opportunity to enhance the event experience, gather valuable insights, and foster a sense of community among participants.

# PROPOSED SYSTEM

The proposed College Events Hub system represents a significant advancement over the existing methods of managing club activities and events within the college. At its core, the system offers a centralized communication platform that aggregates all information, announcements, and updates related to club activities and events. By consolidating communication channels into a single hub, the system reduces the risk of miscommunication and confusion among club members, leaders, coordinators, and students. This centralized approach ensures that all stakeholders have easy access to the latest information, facilitating smoother coordination and organization of events.

One of the key features of the proposed system is its automation of event coordination processes. Through automated registration, attendee management, and payment processing functionalities, the system streamlines event planning and execution. Organizers can set up events efficiently, manage attendee lists effortlessly, and track attendance accurately—all within the platform. By reducing manual effort and minimizing errors associated with traditional paper-based methods, the system enables organizers to focus more on delivering high-quality events that enrich the college experience for participants.

Moreover, the proposed system aims to enhance student engagement and interaction through its interactive features. By incorporating event ratings, reviews, and comments, the system encourages students to provide feedback, ask questions, and interact with organizers and peers. This fosters a sense of community and collaboration within the college, enriching the overall event experience and promoting greater participation. With its comprehensive event information, secure authentication and payment systems, and efficient event planning tools, the College Events Hub system offers a holistic solution for managing club activities and events, paving the way for a more vibrant and connected college community.

.

## ADVANTAGES OF PROPOSED SYSTEM

**Do Centralized Communication Hub:**

* + - The system provides centralized platform for all communication related to club activities

and events.

* + - Users can access announcements, updates, and event details in one place, reducing

confusion and ensuring consistent information dissemination.

**Automated Event Coordination:**

* + - With automated registration, attendee management, and payment processing features, the

system streamlines event coordination processes.

* + - Organizers can set up events efficiently and manage registrations seamlessly, saving time

and minimizing errors associated with manual methods.

**Enhanced Student Engagement:**

* + - The system promotes student engagement through interactive features such as event

ratings, reviews, and comments.

* + - Students can provide feedback, ask questions, and interact with organizers and peers,

fostering a sense of community and collaboration.

**Comprehensive Event Information:**

* + - College Events Hub offers comprehensive event information, including schedules,

descriptions, locations, and attendee lists.

* + - Users have easy access to all relevant details, facilitating informed decision-making and encouraging greater participation in events.

**Secure Authentication and Payment System:**

* + - The system employs secure authentication mechanisms to ensure only authorized users

have access to sensitive information.

* + - Integrated payment systems facilitate secure and convenient event registration, enhancing

user experience and increasing participation.

**Efficient Event Planning and Execution:**

* + - By streamlining processes and centralizing information, the system enables efficient event planning and execution

## SYSTEM REQUIREMENTS

**SOFTWARE REQUIREMENTS**

**Node.js:**

**Version:** Node.js 14.x or higher.

**Description:**

Node.js is a JavaScript runtime built on Chrome's V8 JavaScript engine. It allows for server-side execution of JavaScript code and is essential for running Next.js applications.

**Visual Studio Code (VS Code):**

Visual Studio Code is a lightweight, open-source code editor developed by Microsoft.

It provides features such as syntax highlighting, code completion, debugging support, and integrated version control.

VS Code will be used by developers to write, debug, and manage the codebase of the College Events Hub application, providing a seamless development experience.

**Next.js:**

**Version:** Next.js 14.x or higher.

**Description:** Next.js is a React framework that enables server-side rendering (SSR), static site generation (SSG), and client-side rendering (CSR) for React applications. It provides features like routing, image optimization, and code splitting out of the box.

**TypeScript:**

**Version:** TypeScript 4.x or higher.

**Description:** TypeScript is a statically typed superset of JavaScript that adds optional static typing to the language. It improves developer productivity by catching type-related errors early in the development process and enabling better code organization and documentation.

**TailwindCSS:**

**Version:** TailwindCSS 3.x or higher.

**Description:** TailwindCSS is a utility-first CSS framework that provides pre-built utility classes for styling user interfaces. It allows developers to rapidly build custom designs without writing custom CSS by hand, promoting a highly customizable and maintainable codebase.

**Stripe:**

**Version**: Stripe API.

**Description:** Stripe is a popular payment processing platform that allows businesses to accept online payments securely. It provides APIs for handling payment transactions, managing customer data, and integrating with various e-commerce platforms.

**Shadcn:**

Further clarification needed. If it's related to Shadow DOM or a specific library, additional details are required.

**Uploadthing:**

Further clarification needed. If it's a specific library or tool for handling file uploads, additional details are required

## HARDWARE REQUIREMENTS:

## Server:

**CPU:** Multi-core processor (e.g., Intel Core i5 or equivalent)

**RAM:** Minimum 4GB (8GB or more recommended for better performance)

**Storage:** SSD storage recommended for faster read/write operations

**Client Devices:**

**Desktop/Laptop:**

**CPU:** Dual-core processor (e.g., Intel Core i3 or equivalent)

**RAM:** Minimum 4GB

**Storage:** HDD or SSD

**Display:** Minimum resolution of 1366x768 pixels

**Mobile:** Compatible with smartphones and tablets running modern web browsers

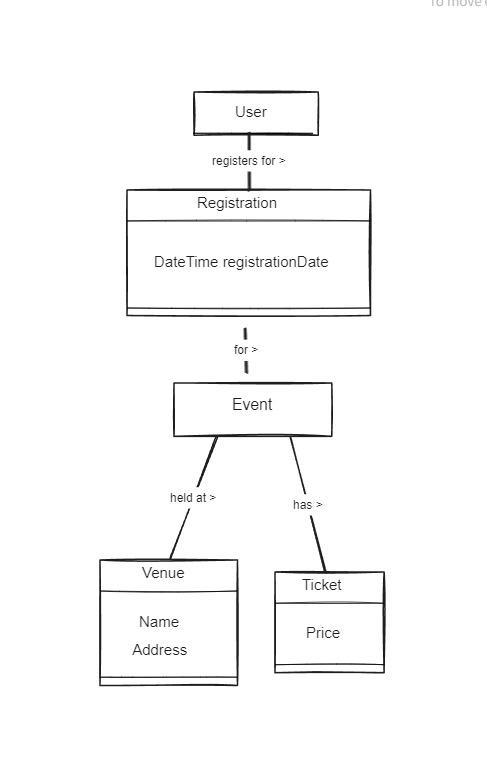
(e.g., Safari, Chrome).

Responsive design for optimal viewing and interaction on various

screen sizes

# 4 SYSTEM DESIGN

## 4.1 ARCHITECTURE

****

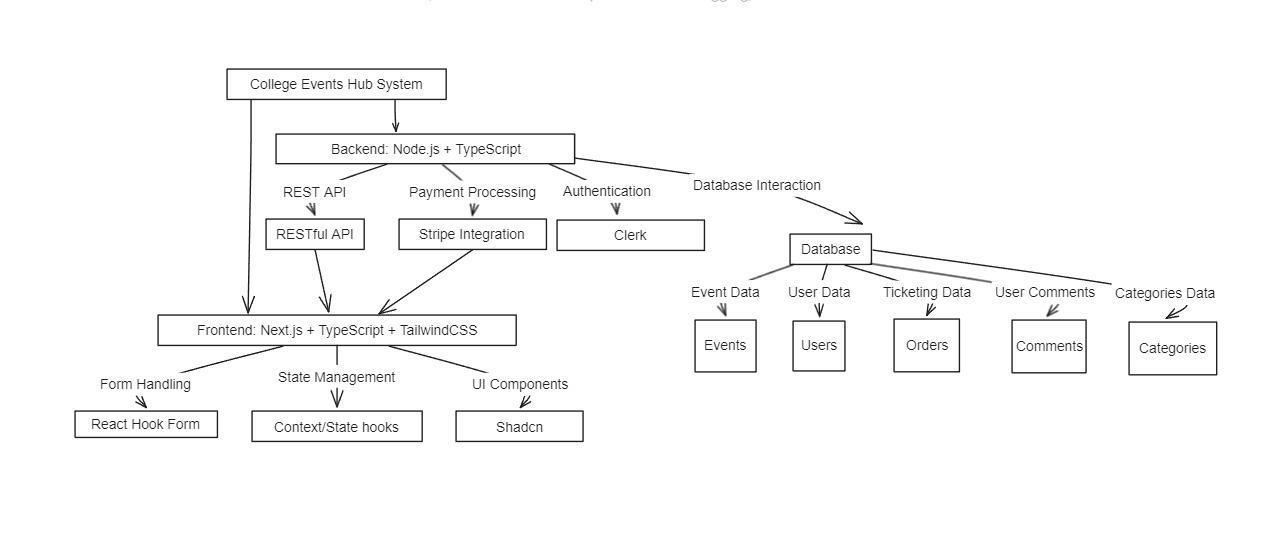
**Fig 3: Architecture Diagram**

The user registers for an event scheduled for a specific date and time, indicated by the "registrationDate" field. This event, hosted at a venue specified by its name and address, promises an experience worth attending. The registration process likely involves providing necessary information such as personal details and possibly payment for the event, which might be reflected in the "Price" and "Ticket" fields.

Attending this event could offer numerous opportunities, depending on its nature. Whether it's a conference, workshop, concert, or any other gathering, participants can expect to engage with like-minded individuals, gain valuable insights, or simply enjoy a memorable experience. The venue's name and address provide clarity on where the event will take place, ensuring attendees can plan their logistics accordingly.

With the registration complete, the user is now poised to immerse themselves in the event's offerings. They've secured their spot and are ready to partake in whatever the event has in store, whether it's networking, learning, entertainment, or a combination thereof. The details provided enable seamless coordination between the organizers and attendees, ensuring a smooth and enjoyable experience for all involved.

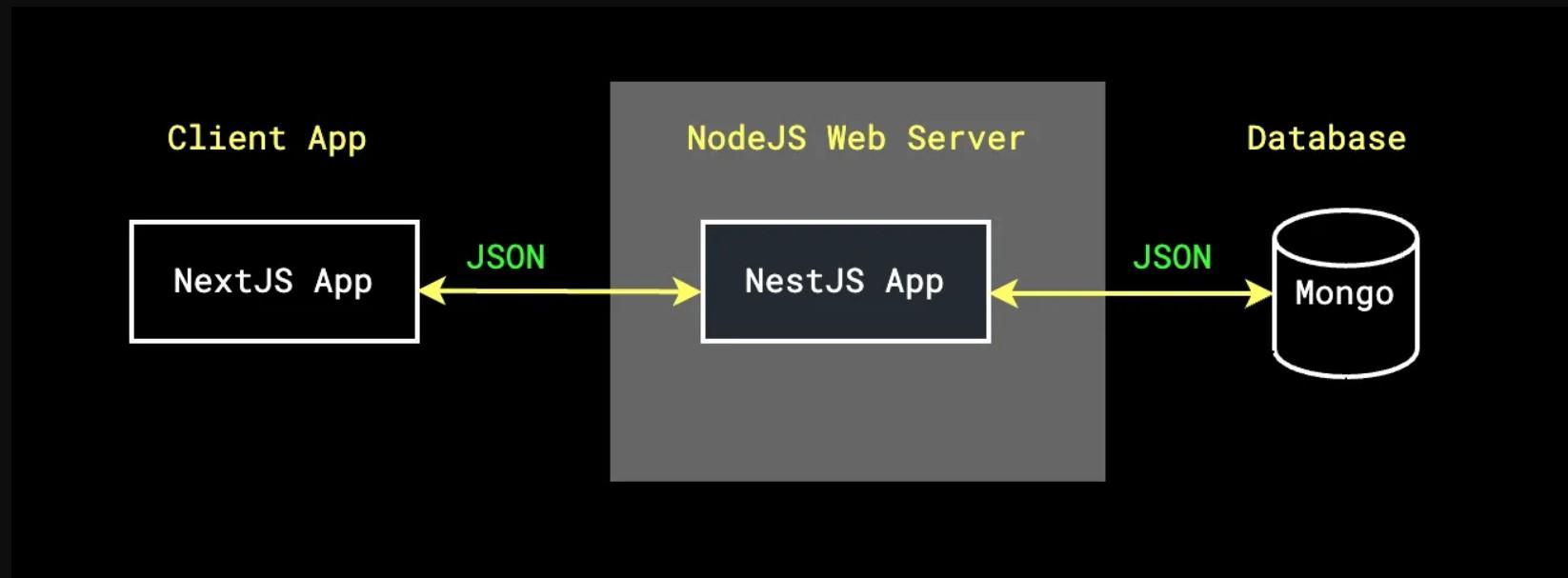
## PROPOSED SYSTEM ARCHITECTURE

****

**Fig 4: Proposed System Architecture**

In the system architecture for College Events Hub, MongoDB serves as the core database component responsible for storing and managing event-related data. The architecture comprises several interconnected elements:

The frontend, developed using Next.js, integrates React Hook Form for efficient data management and TailwindCSS for sleek styling, ensuring a user-friendly interface. This frontend communicates with the backend, powered by Node.js with TypeScript for enhanced reliability and scalability.



**Fig 5: Server**

Authentication is managed through Clerk, ensuring secure user access across both frontend and backend components. Stripe is integrated into the backend for seamless payment processing, while Uploadthing handles file upload tasks efficiently.

The MongoDB database component stores various types of data essential for event management, including event details, user information, and feedback submissions. Node.js facilitates communication with MongoDB, handling data storage and retrieval operations effectively.

This cohesive architecture enables College Events Hub to provide robust event management capabilities, secure authentication, seamless payment processing, and efficient file management. MongoDB's scalability and flexibility complement the other components, contributing to the platform's overall functionality and performance

## MODULES

**Frontend Module:**

The frontend module is the user-facing part of the application that users interact with directly. It is developed using Next.js, a React framework known for its server-side rendering capabilities, providing a fast and responsive user experience. Within the frontend module, React Hook Form is utilized for efficient data management, allowing seamless form handling and validation. Additionally, TailwindCSS is employed for sleek styling, ensuring an aesthetically pleasing and user-friendly interface. The frontend module is responsible for rendering event information, user profiles, forms for registration and feedback submission, and facilitating user interaction with the application.

**Backend Module:**

The backend module serves as the core of the application's logic and functionality. Powered by Node.js with TypeScript, it ensures enhanced reliability and scalability. The backend module communicates with the frontend, handling requests and responses, processing business logic, and interacting with other modules such as authentication, payment processing, file management, and database operations. It provides APIs for frontend consumption. Additionally, the backend module manages security aspects, ensuring data integrity, access control, and protection against potential threats.

**Authentication Module:**

The authentication module is responsible for managing user authentication and access control within the application. Leveraging Clerk, a third-party authentication service, it ensures secure user access across both frontend and backend components. The authentication module handles user registration, login, logout, and password recovery functionalities, ensuring that only authorized users can access the application's features and data. It provides a seamless and secure authentication experience for users, enhancing the overall security and reliability of the application.

**Payment Processing Module:**

The payment processing module facilitates seamless and secure payment transactions within the application. Integrated with Stripe, a popular payment gateway, it enables users to make payments for event registrations, ticket purchases, or other services offered by the application. The payment processing module handles payment validation, processing, and transaction management, ensuring the confidentiality and integrity of sensitive financial information. It provides a smooth and reliable payment experience for users, enhancing the overall convenience and usability of the application.

**File management Module:**

The file management module is responsible for efficient handling and storage of files within the application. Utilizing Uploadthing, a file upload service, it manages file upload tasks such as uploading event materials/resources, user profile pictures etcThe file management module ensures reliable and secure file storage, handling file uploads, downloads, and storage operations efficiently. It enhances the application's usability by providing users with easy access to relevant files and resources.

**Database Module:**

The database module serves as the core database component responsible for storing and managing event-related data, user information, and feedback submissions. Powered by MongoDB, a NoSQL database, it provides scalability, flexibility, and performance for handling large volumes of data. The database module stores various types of data essential for event management, including event details (title, description, date, time, location), user profiles, authentication credentials, payment transactions, and feedback submissions. It facilitates data storage and retrieval operations, ensuring data integrity, consistency, and reliability throughout the application.

**Event Management Module:**

The event management module enables the creation, editing, and deletion of events within the application. It provides functionalities for managing event details such as title, description, date, time, location, and other relevant information. The event management module facilitates event scheduling, speaker/performer management, sponsor management, ticket type, and pricing management, and venue management. It allows users to browse events, view event details, register for events, and participate in event activities. The event management module plays a crucial role in organizing and managing events effectively, ensuring a seamless and enjoyable experience for users.

**User Management Module:**

The user management module handles user-related functionalities within the application. It facilitates user registration, authentication, and user profile management. The user management module allows users to create accounts, update their profiles, manage their preferences, and perform account-related actions such as password recovery or account deletion. Additionally, it provides administrative functionalities for managing user accounts, roles, and permissions. The user management module ensures secure user access and personalized user experiences, enhancing user engagement and satisfaction.

**Comments Module:**

The comments module enables users to leave comments and feedback on events or other content within the application. It provides a platform for users to share their opinions, ask questions, or engage in discussions related to events or topics of interest. The comments module allows users to view and interact with comments, reply to comments, and like or dislike comments. It enhances user engagement and community interaction within the application, fostering collaboration and communication among users. Additionally, the comments module provides valuable feedback to event organizers and administrators, helping improve event quality and user satisfaction.

# 5. IMPLEMENTATION AND INSTALLATION

* 1. **ALGORITHM**

**Initialize Application:**

- Set up frontend and backend environments

- Configure authentication, payment processing, and file management services

- Establish database connection

**User Authentication:**

**a. User Login:**

- Prompt user to enter credentials

- Verify credentials against stored data in the database

- If credentials are valid, generate authentication token and grant access

- If credentials are invalid, display error message and prompt user to retry

**b. User Registration:**

- Collect user information (e.g., name, email, password)

- Validate input data

- Create new user account and store in the database

- Send confirmation email to the user

**c. Forgot Password:**

- Prompt user to enter their email

- Verify email address

- Generate password reset link and send it to the user's email

- Allow user to reset password using the link

**4. Event Management:**

**a. Create Event:**

- Collect event details (title, description, date, time, location, etc.)

- Validate input data

- Store event information in the database

**b. Edit Event:**

- Retrieve event details from the database

- Allow authorized users to modify event details

- Update event information in the database

**c. Delete Event:**

- Retrieve event details from the database

- Confirm deletion with user

- Remove event from the database

**d. View Events:**

- Retrieve list of events from the database

- Display events to users

**e. Manage Registrations:**

- Retrieve list of event registrations from the database

- Allow authorized users to view and manage registrations

**f. Manage Ticket Sales:**

- Retrieve ticket sales data from the database

- Allow authorized users to view and manage ticket sales

**5. Payment Processing:**

**a. Initiate Payment:**

- Collect payment details from user (e.g., credit card information)

- Validate payment information

- Process payment using Stripe API

**b. Confirm Payment:**

- Verify payment status

- Update ticket sales data in the database

- Send confirmation email to the user

**6. File Management:**

**a. Upload File:**

- Allow users to upload event-related files (e.g., flyers, presentations)

- Validate file format and size

- Store uploaded files securely using Uploadthing

**b. Download File:**

- Retrieve file from the database

- Allow users to download files associated with events

**7. User Interaction:**

**a. Browse Events:**

- Display list of upcoming events to users

**b. View Event Details:**

- Retrieve event details from the database

- Display event information to users

**c. Register for Event:**

- Allow users to register for events

- Update registration data in the database

**d. Access Event Materials/Resources:**

- Provide access to event-related materials and resources (e.g., presentations, handouts)

**e. Participate in Event Activities:**

Enable users to engage in activities during events (e.g., Q&A sessions, polls)

**8. Feedback Submission:**

- Allow users to submit feedback for events attended

- Store feedback submissions in the database

## IMPLEMENTATION STEPS:

***Step 1:*** **Setting Up Environment and Dependencies**

- Set up the development environment by installing necessary tools such as Node.js, npm (Node Package Manager), and MongoDB.

- Create a new project directory for the event management app.

***Step 2*: Backend Development**

- Initialize a new Node.js project using npm.

- Install required dependencies such as Express.js, Mongoose (for MongoDB interaction), Clerk (for authentication), Stripe (for payment processing), and Uploadthing (for file management).

- Set up the backend logic for handling routes, authentication, payment processing, file upload/download, and database operations.

- Create models for events, users, feedback, and other necessary data structures.

***Step 3:*** **Frontend Development**

- Initialize a new React project using Create React App or Next.js.

- Install necessary dependencies such as React Hook Form and TailwindCSS for efficient data management and styling.

- Implement frontend components for user authentication, event browsing, registration, ticket purchasing, and feedback submission.

- Ensure seamless integration with the backend APIs for data retrieval and manipulation.

***Step 4:*** **Database Setup**

- Set up a MongoDB database instance either locally or using a cloud-based service like MongoDB Atlas.

- Define schemas for storing event-related data, user information, feedback submissions, and other relevant data structures.

- Establish a connection between the backend application and the MongoDB database using Mongoose.

***Step 5:*** **Authentication Configuration**

- Configure authentication settings using Clerk for secure user access across frontend and backend components.

- Implement authentication endpoints for user registration, login, logout, and password recovery.

***Step 6:*** **Payment Gateway Integration**

- Set up a Stripe account and obtain API keys for payment processing.

- Integrate Stripe SDK into the backend for processing payment transactions securely.

- Implement payment processing endpoints for handling payment requests and transactions.

***Step 7:*** **File Management Configuration**

- Set up an account with Uploadthing or a similar file upload service for efficient file management.

- Integrate Uploadthing SDK or APIs into the backend for handling file upload/download tasks securely.

***Step 8:*** **Testing and Quality Assurance**

- Conduct thorough testing of all application components, including unit tests, integration tests, and end-to-end tests.

- Perform security testing to identify and address vulnerabilities.

- Conduct usability testing to ensure a seamless user experience.

- Address any issues or bugs discovered during testing and implement fixes accordingly.

***Step 9:*** **Deployment**

- Deploy the backend and frontend components to a hosting provider such as Heroku or Vercel.

- Configure environment variables for sensitive information such as API keys and database connection strings.

- Set up continuous integration and deployment pipelines for automated deployment.

- Monitor application performance and error logs to ensure smooth operation in production.

***Step 10:*** **Documentation**

- Create comprehensive documentation covering installation instructions, usage guidelines, API references, and troubleshooting tips.

- Provide user guides for both administrators and end-users to facilitate smooth adoption and usage of the application.

## SOURCE CODE:

* + 1. **package.json**

{

"name": "college\_events\_app",

"version": "0.1.0",

"private": true,

"scripts": {

"dev": "next dev",

"build": "next build",

"start": "next start",

"lint": "next lint"

},

"dependencies": {

"@clerk/nextjs": "^4.27.5",

"@hookform/resolvers": "^3.3.2",

"@radix-ui/react-alert-dialog": "^1.0.5",

"@radix-ui/react-checkbox": "^1.0.4",

"@radix-ui/react-dialog": "^1.0.5",

"@radix-ui/react-dropdown-menu": "^2.0.6",

"@radix-ui/react-label": "^2.0.2",

"@radix-ui/react-select": "^2.0.0",

"@radix-ui/react-separator": "^1.0.3",

"@radix-ui/react-slot": "^1.0.2",

"@stripe/stripe-js": "^2.2.1",

"@uploadthing/react": "^6.0.2",

"class-variance-authority": "^0.7.0",

"clsx": "^2.0.0",

"lucide-react": "^0.294.0",

"mongodb": "^6.3.0",

"mongoose": "^8.0.3",

"next": "14.0.4",

"query-string": "^8.1.0",

"react": "^18",

"react-datepicker": "^4.24.0",

"react-dom": "^18",

"react-hook-form": "^7.49.0",

"stripe": "^14.8.0",

"svix": "^1.15.0",

"tailwind-merge": "^2.1.0",

"tailwindcss-animate": "^1.0.7",

"uploadthing": "^6.0.4",

"zod": "^3.22.4"

},

"devDependencies": {

"@types/node": "^20",

"@types/react": "^18",

"@types/react-datepicker": "^4.19.3",

"@types/react-dom": "^18",

"autoprefixer": "^10.0.1",

"postcss": "^8",

"tailwindcss": "^3.3.0",

"typescript": "^5"

}

}

* + 1. **Home Page**

import CategoryFilter from "@/components/shared/CategoryFilter";

import Collection from "@/components/shared/Collection";

import Search from "@/components/shared/Search";

import { Button } from "@/components/ui/button";

import { getAllEvents } from "@/lib/actions/event.actions";

import { SearchParamProps } from "@/types";

import Image from "next/image";

import Link from "next/link";

export default async function Home({ searchParams }: SearchParamProps) {

const page = Number(searchParams?.page) || 1;

const searchText = (searchParams?.query as string) || "";

const category = (searchParams?.category as string) || "";

const events = await getAllEvents({

query: searchText,

category,

page,

limit: 6,

});

return (

<>

<section className="bg-primary-50 bg-dotted-pattern bg-contain py-5 md:py-10">

<div className="wrapper grid grid-cols-1 gap-5 md:grid-cols-2 2xl:gap-0">

<div className="flex flex-col justify-center gap-8">

<h1 className="h1-bold">Welcome to our Events Hub!</h1>

<p className="p-regular-20 md:p-regular-24">

Discover, Participate, and Celebrate the vibrant tapestry of

events at MLRIT. Our Events Hub is your gateway to a world of

excitement, knowledge, and connections.

</p>

<Button size="lg" asChild className="button w-full sm:w-fit">

<Link href="#events">Explore Now</Link>

</Button>

</div>

<Image

src="/assets/images/hero.png"

alt="hero"

width={1000}

height={1000}

className="max-h-[70vh] object-contain object-center 2xl:max-h-[50vh]"

/>

</div>

</section>

<section

id="events"

className="wrapper my-8 flex flex-col gap-8 md:gap-12"

>

<h2 className="h2-bold">

🎓 MLRIT Events Hub -<br /> Where Moments Become Memories.

</h2>

<div className="flex w-full flex-col gap-5 md:flex-row">

<Search />

<CategoryFilter />

</div>

<Collection

data={events?.data}

emptyTitle="No Events Found"

emptyStateSubtext="Come back later"

collectionType="All\_Events"

limit={6}

page={page}

totalPages={events?.totalPages}

/>

</section>

</>

);

}

* + 1. **Create Event**

import EventForm from "@/components/shared/EventForm"

import { auth } from "@clerk/nextjs";

const CreateEvent = () => {

const { sessionClaims } = auth();

const userId = sessionClaims?.userId as string;

const admin = {

id : ['659fb4e847dda8ad35cbe851','659fd1c3a3b1509fbb2e99b0', '65a782ab9affdaf7e2c2d257']

}

if(admin.id.includes(userId)){

return (

<>

<section className="bg-primary-50 bg-dotted-pattern bg-cover bg-center py-5 md:py-10">

<h3 className="wrapper h3-bold text-center sm:text-left">Create Event</h3>

</section>

<div className="wrapper my-8">

<EventForm userId={userId} type="Create" />

</div>

</>

)

}

return(

<section className="bg-primary-50 bg-dotted-pattern bg-cover bg-center py-5 md:py-10">

<center><h3 className="wrapper h3-bold text-center sm:text-left">Contact us to host events</h3></center>

</section>)

}

export default CreateEvent

## User Profile

import Collection from '@/components/shared/Collection'

import { Button } from '@/components/ui/button'

import { getEventsByUser } from '@/lib/actions/event.actions'

import { getOrdersByUser } from '@/lib/actions/order.actions'

import { IOrder } from '@/lib/database/models/order.model'

import { SearchParamProps } from '@/types'

import { auth } from '@clerk/nextjs'

import Link from 'next/link'

import React from 'react'

const ProfilePage = async ({ searchParams }: SearchParamProps) => {

const { sessionClaims } = auth();

const userId = sessionClaims?.userId as string;

const ordersPage = Number(searchParams?.ordersPage) || 1;

const eventsPage = Number(searchParams?.eventsPage) || 1;

const orders = await getOrdersByUser({ userId, page: ordersPage})

const orderedEvents = orders?.data.map((order: IOrder) => order.event) || [];

const organizedEvents = await getEventsByUser({ userId, page: eventsPage })

return (

<>

{/\* My Tickets \*/}

<section className="bg-primary-50 bg-dotted-pattern bg-cover bg-center py-5 md:py-10">

<div className="wrapper flex items-center justify-center sm:justify-between">

<h3 className='h3-bold text-center sm:text-left'>My Tickets</h3>

<Button asChild size="lg" className="button hidden sm:flex">

<Link href="/#events">

Explore More Events

</Link>

</Button>

</div>

</section>

<section className="wrapper my-8">

<Collection

data={orderedEvents}

emptyTitle="No event tickets purchased yet"

emptyStateSubtext="No worries - plenty of exciting events to explore!"

collectionType="My\_Tickets"

limit={3}

page={ordersPage}

urlParamName="ordersPage"

totalPages={orders?.totalPages}

/>

</section>

{/\* Events Organized \*/}

<section className="bg-primary-50 bg-dotted-pattern bg-cover bg-center py-5 md:py-10">

<div className="wrapper flex items-center justify-center sm:justify-between">

<h3 className='h3-bold text-center sm:text-left'>Events Organized</h3>

<Button asChild size="lg" className="button hidden sm:flex">

<Link href="/events/create">

Create New Event

</Link>

</Button>

</div>

</section>

<section className="wrapper my-8">

<Collection

data={organizedEvents?.data}

emptyTitle="No events have been created yet"

emptyStateSubtext="Go create some now"

collectionType="Events\_Organized"

limit={3}

page={eventsPage}

urlParamName="eventsPage"

totalPages={organizedEvents?.totalPages}

/>

</section>

</>

)

}

export default ProfilePage

* + 1. **Event Page**

import CheckoutButton from "@/components/shared/CheckoutButton";

import Collection from "@/components/shared/Collection";

import CommentSection from "@/components/shared/CommentSection";

import {

getEventById,

getRelatedEventsByCategory,

} from "@/lib/actions/event.actions";

import { formatDateTime } from "@/lib/utils";

import { SearchParamProps } from "@/types";

import Image from "next/image";

import { auth } from "@clerk/nextjs";

const EventDetails = async ({

params: { id },

searchParams,

}: SearchParamProps) => {

const event = await getEventById(id);

const eventId: string = event.\_id as string;

const relatedEvents = await getRelatedEventsByCategory({

categoryId: event.category.\_id,

eventId: event.\_id,

page: searchParams.page as string,

});

const { sessionClaims } = auth();

const userId = sessionClaims?.userId as string;

return (

<>

<section className="flex justify-center bg-primary-50 bg-dotted-pattern bg-contain">

<div className="grid grid-cols-1 md:grid-cols-2">

<img

src={event.imageUrl}

alt="hero image"

style={{ width: "100%", height: "100%", margin: "20px" }}

/>

<div className="flex w-full flex-col gap-8 p-5 md:p-10">

<div className="flex flex-col gap-6">

<h2 className="h2-bold">{event.title}</h2>

<div className="flex flex-col gap-3 sm:flex-row sm:items-center">

<div className="flex gap-3">

<p className="p-bold-20 rounded-full bg-green-500/10 px-5 py-2 text-green-700">

{event.isFree ? "FREE" : `Rs.${event.price}`}

</p>

<p className="p-medium-16 rounded-full bg-grey-500/10 px-4 py-2.5 text-grey-500">

{event.category.name}

</p>

</div>

<p className="p-medium-18 ml-2 mt-2 sm:mt-0">

by{" "}

<span className="text-primary-500">

{event.organizer.firstName} {event.organizer.lastName}

</span>

</p>

</div>

</div>

<CheckoutButton event={event} />

<div className="flex flex-col gap-5">

<div className="flex gap-2 md:gap-3">

<Image

src="/assets/icons/calendar.svg"

alt="calendar"

width={32}

height={32}

/>

<div className="p-medium-16 lg:p-regular-20 flex flex-wrap items-center">

<p>

{formatDateTime(event.startDateTime).dateOnly} -{" "}

{formatDateTime(event.startDateTime).timeOnly}

</p>

<p>

{formatDateTime(event.endDateTime).dateOnly} -{" "}

{formatDateTime(event.endDateTime).timeOnly}

</p>

</div>

</div>

<div className="p-regular-20 flex items-center gap-3">

<Image

src="/assets/icons/location.svg"

alt="location"

width={32}

height={32}

/>

<p className="p-medium-16 lg:p-regular-20">{event.location}</p>

</div>

</div>

<div className="flex flex-col gap-2">

<p className="p-bold-20 text-grey-600">What You'll Learn:</p>

<p className="p-medium-16 lg:p-regular-18">{event.description}</p>

<a

style={{ color: "blue" }}

href={event.url}

target="\_blank"

rel="noopener noreferrer"

>

{event.url}

</a>

</div>

</div>

</div>

</section>

{/\* EVENTS with the same category \*/}

<section className="wrapper my-8 flex flex-col gap-8 md:gap-12">

<h2 className="h2-bold">Related Events</h2>

<Collection

data={relatedEvents?.data}

emptyTitle="No Events Found"

emptyStateSubtext="Come back later"

collectionType="All\_Events"

limit={3}

page={searchParams.page as string}

totalPages={relatedEvents?.totalPages}

/>

</section>

<div className="flex w-full flex-col gap-5 md:flex-row">

{sessionClaims && <CommentSection userId={userId} eventId={eventId} />}

</div>

</>

);

};

export default EventDetails;

## Update event

import EventForm from "@/components/shared/EventForm"

import { getEventById } from "@/lib/actions/event.actions"

import { auth } from "@clerk/nextjs";

type UpdateEventProps = {

params: {

id: string

}

}

const UpdateEvent = async ({ params: { id } }: UpdateEventProps) => {

const { sessionClaims } = auth();

const userId = sessionClaims?.userId as string;

const event = await getEventById(id)

return (

<>

<section className="bg-primary-50 bg-dotted-pattern bg-cover bg-center py-5 md:py-10">

<h3 className="wrapper h3-bold text-center sm:text-left">Update Event</h3>

</section>

<div className="wrapper my-8">

<EventForm

type="Update"

event={event}

eventId={event.\_id}

userId={userId}

/>

</div>

</>

)

}

export default UpdateEvent

## Clerk

import { Webhook } from 'svix'

import { headers } from 'next/headers'

import { WebhookEvent } from '@clerk/nextjs/server'

import { createUser, deleteUser, updateUser } from '@/lib/actions/user.actions'

import { clerkClient } from '@clerk/nextjs'

import { NextResponse } from 'next/server'

export async function POST(req: Request) {

// You can find this in the Clerk Dashboard -> Webhooks -> choose the webhook

const WEBHOOK\_SECRET = process.env.WEBHOOK\_SECRET

if (!WEBHOOK\_SECRET) {

throw new Error('Please add WEBHOOK\_SECRET from Clerk Dashboard to .env or .env.local')

}

// Get the headers

const headerPayload = headers();

const svix\_id = headerPayload.get("svix-id");

const svix\_timestamp = headerPayload.get("svix-timestamp");

const svix\_signature = headerPayload.get("svix-signature");

if (!svix\_id || !svix\_timestamp || !svix\_signature) {

return new Response('Error occured -- no svix headers', {

status: 400

})

}

const payload = await req.json()

const body = JSON.stringify(payload);

// Create a new Svix instance with your secret.

const wh = new Webhook(WEBHOOK\_SECRET);

let evt: WebhookEvent

// Verify the payload with the headers

try {

evt = wh.verify(body, {

"svix-id": svix\_id,

"svix-timestamp": svix\_timestamp,

"svix-signature": svix\_signature,

}) as WebhookEvent

} catch (err) {

console.error('Error verifying webhook:', err);

return new Response('Error occured', {

status: 400

}) }

const { id } = evt.data;

const eventType = evt.type;

if(eventType === 'user.created') {

const { id, email\_addresses, image\_url, first\_name, last\_name, username } = evt.data;

const user = {

clerkId: id,

email: email\_addresses[0].email\_address,

username: username!,

firstName: first\_name,

lastName: last\_name,

photo: image\_url,

}

const newUser = await createUser(user);

if(newUser) {

await clerkClient.users.updateUserMetadata(id, {

publicMetadata: {

userId: newUser.\_id

}

}) }

return NextResponse.json({ message: 'OK', user: newUser })

}

if (eventType === 'user.updated') {

const {id, image\_url, first\_name, last\_name, username } = evt.data

const user = {

firstName: first\_name,

lastName: last\_name,

username: username!,

photo: image\_url,

}

const updatedUser = await updateUser(id, user)

return NextResponse.json({ message: 'OK', user: updatedUser })

}

if (eventType === 'user.deleted') {

const { id } = evt.data

const deletedUser = await deleteUser(id!)

return NextResponse.json({ message: 'OK', user: deletedUser }) }

return new Response('', { status: 200 })}

## Stripe

import stripe from 'stripe'

import { NextResponse } from 'next/server'

import { createOrder } from '@/lib/actions/order.actions'

export async function POST(request: Request) {

const body = await request.text()

const sig = request.headers.get('stripe-signature') as string

const endpointSecret = process.env.STRIPE\_WEBHOOK\_SECRET!

let event

try {

event = stripe.webhooks.constructEvent(body, sig, endpointSecret)

} catch (err) {

return NextResponse.json({ message: 'Webhook error', error: err })

}

// Get the ID and type

const eventType = event.type

// CREATE

if (eventType === 'checkout.session.completed') {

const { id, amount\_total, metadata } = event.data.object

const order = {

stripeId: id,

eventId: metadata?.eventId || '',

buyerId: metadata?.buyerId || '',

totalAmount: amount\_total ? (amount\_total / 100).toString() : '0',

createdAt: new Date(),

}

const newOrder = await createOrder(order)

return NextResponse.json({ message: 'OK', order: newOrder })

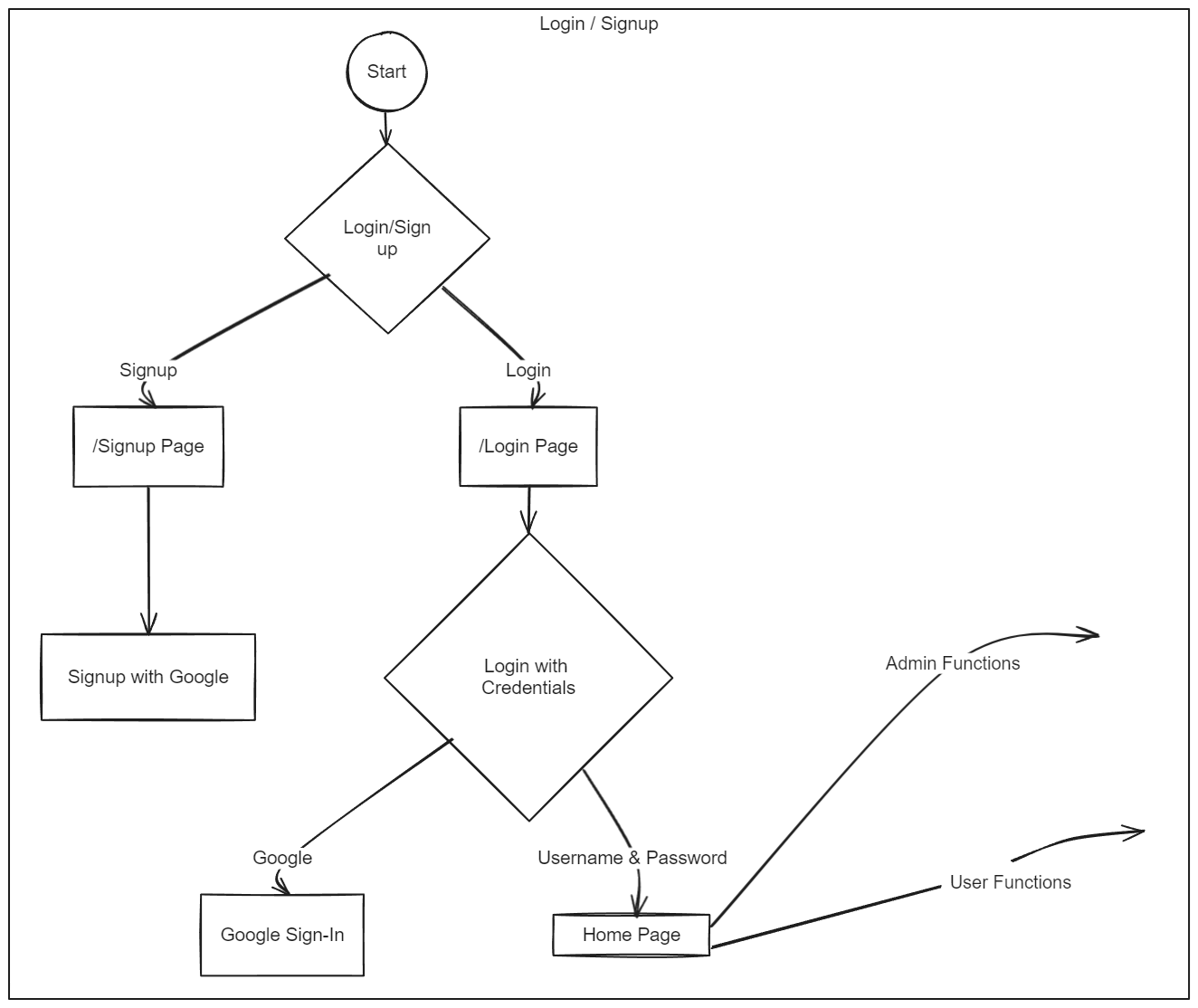
}

return new Response('', { status: 200 })

}

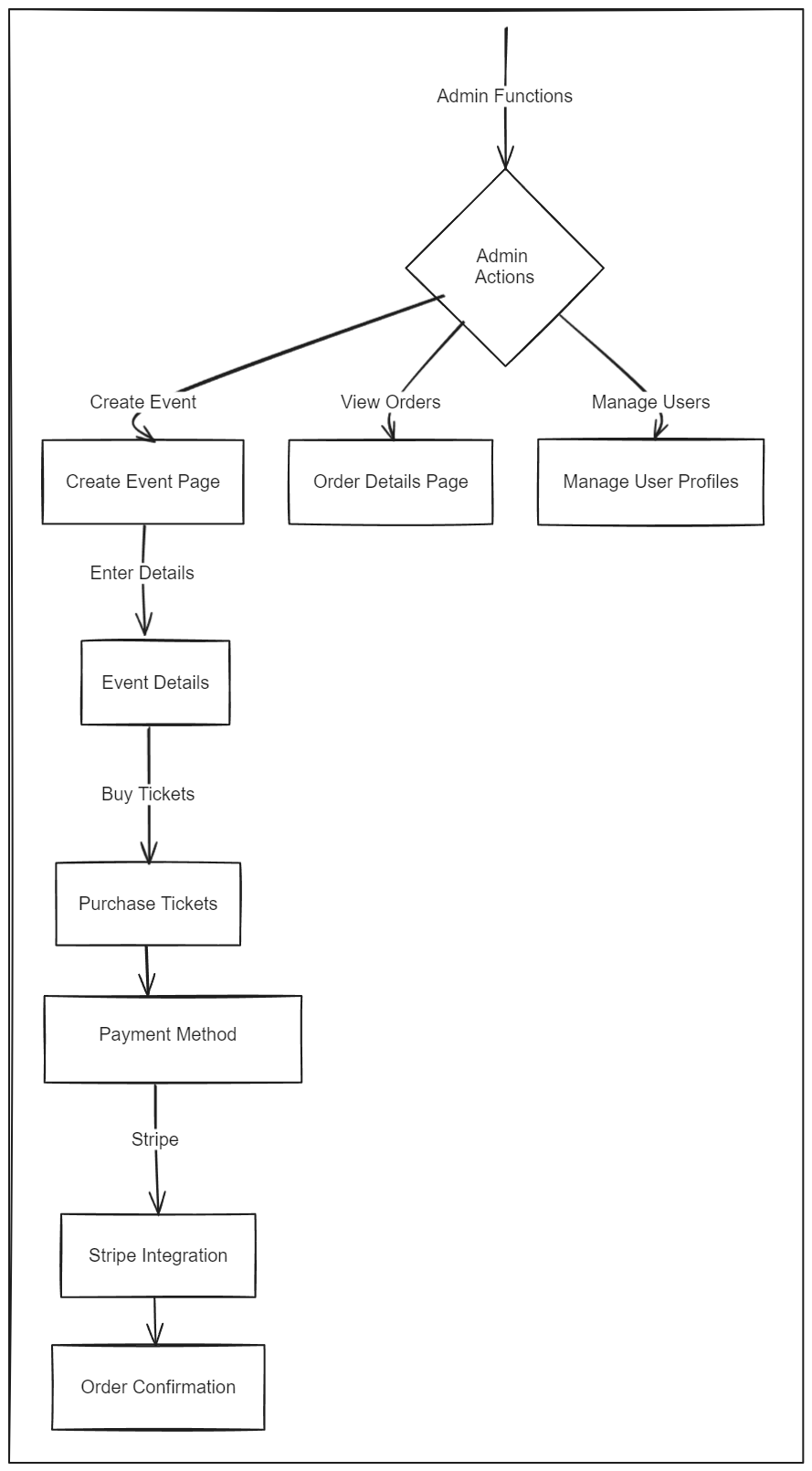
# 6. Data Flow Diagrams (DFD)

## Login/Signup



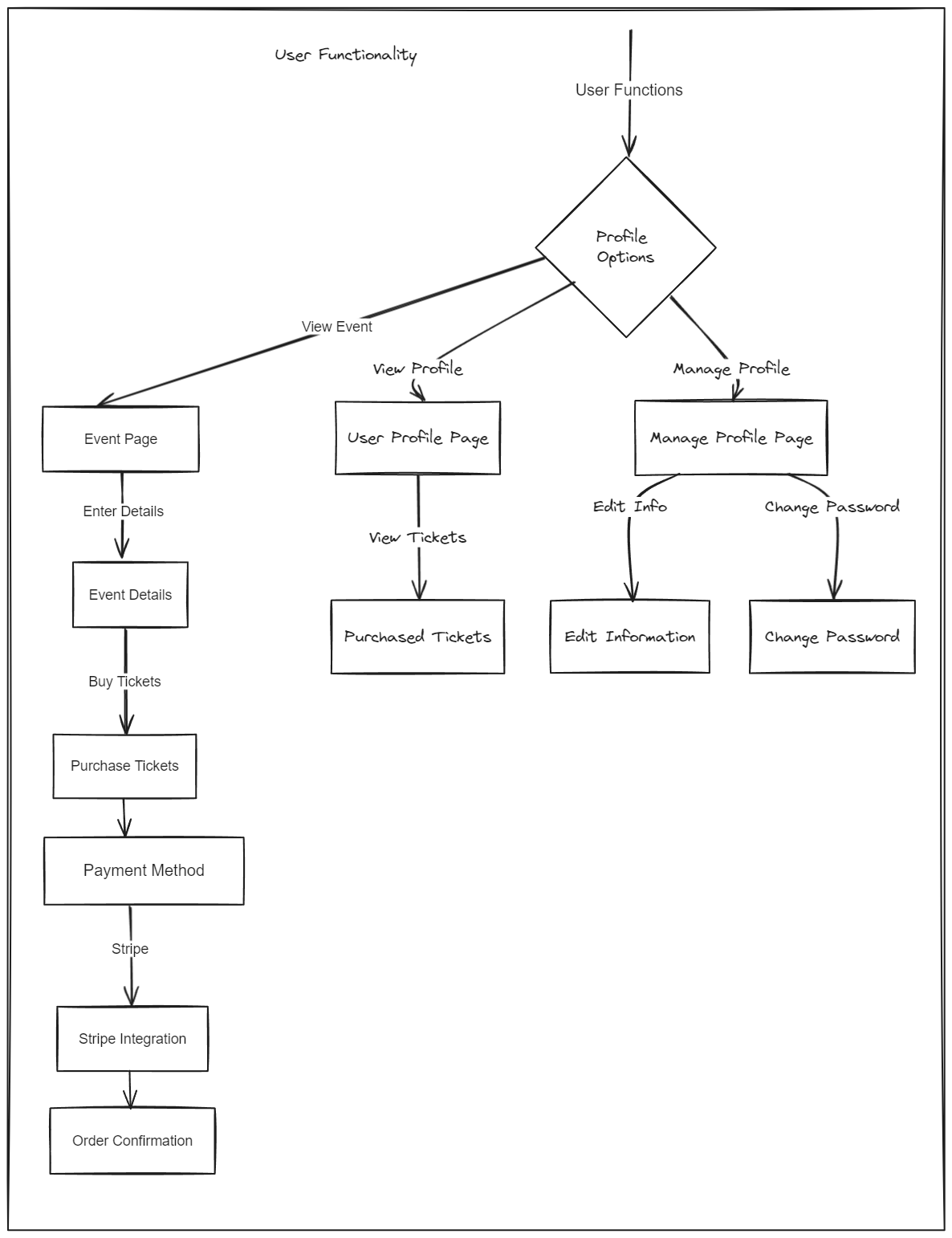
**Fig 7: Login/Signup**

* 1. **Admin**



### Fig 6: Flowchart of Admin.

## User DFD



**Fig 7: User DFD**

# 7. TESTING

* 1. **INTRODUCTION**

The purpose of testing is to discover errors. Testing is the process of trying to discover every conceivable fault or weakness in a work product. It provides a way to check the functionality of components, sub-assemblies, assemblies and/or a finished product. It is the process of exercising software with the intent of ensuring that the software system meets its requirements and user expectations and does not fail in an unacceptable manner. There are various types of test. Each test type addresses a specific testing requirement.

Testing an event management application involves various aspects, including functional testing, usability testing, security testing, performance testing, and more.

### TYPES OF TESTS

**Unit testing**

Unit testing involves the design of test cases that validate that the internal program logic is functioning properly, and that program inputs produce valid outputs. All decision branches and internal code flow should be validated. It is the testing of individual software units of the application .It is done after the completion of an individual unit before integration. This is a structural testing that relies on knowledge of its construction and is invasive. Unit tests perform basic tests at component level and test a specific business process, application, and/or system configuration. Unit tests ensure that each unique path of a business process performs accurately to the documented specifications and contains clearly defined inputs and expected results.

### Integration testing

Integration tests are designed to test integrated software components to determine if they actually run as one program. Testing is event driven and is more concerned with the basic outcome of screens or fields. Integration tests demonstrate that although the components were individually satisfaction, as shown by successfully unit testing,

the combination of components is correct and consistent. Integration testing is specifically aimed at exposing the problems that arise from the combination of components.

### System Testing

System testing ensures that the entire integrated software system meets requirements. It tests a configuration to ensure known and predictable results. An example of system testing is the configuration-oriented system integration test. System testing is based on process descriptions and flows, emphasizing pre-driven process links and integration points.

**Regression Testing:**

Perform regression testing after each software update or code change to ensure that existing features continue to work as expected. Re-run functional tests, usability tests, security tests, and other relevant tests to validate the stability of the application. Automate regression tests using tools like Selenium or Cypress to streamline the testing process and ensure consistent results.

**Compatibility Testing:**

Test the application across different web browsers (e.g Chrome, Firefox, Safari, Edge) and browser versions to ensure compatibility. Verify compatibility with different operating systems (e.g., Windows, macOS, Linux) and devices (e.g., desktops, laptops, tablets, smartphones). Test for compatibility with assistive technologies such as screen readers and keyboard navigation for accessibility compliance.

### White Box Testing

White Box Testing is a testing in which the software tester has knowledge of the inner workings, structure and language of the software, or at least its purpose. It is used to test areas that cannot be reached from a black box level.

**Performance Testing:**

Test the application's performance under different load conditions to ensure scalability and reliability. Use tools like Apache JMeter or LoadRunner to simulate heavy traffic and measure response times, throughput, and resource utilization.

Optimize database queries, API endpoints, and frontend components to improve performance and reduce latency. Monitor system metrics such as CPU usage, memory usage, and network traffic to identify bottlenecks and optimize resource allocation.

### Black Box Testing

Black Box Testing is testing the software without any knowledge of the inner workings, structure or language of the module being tested. Black box tests, as most other kinds of tests, must be written from a definitive source document, such as specification or requirements document, such as specification or requirements document. It is a test in which the software under test is treated as a black box. you cannot “see” into it. The test provides inputs and responds to outputs without considering how the software works.

**Usability Testing:**

Evaluate the user interface for ease of navigation, clarity of information, and overall user experience. Test responsiveness across different devices and screen sizes to ensure compatibility and accessibility. Verify that the application adheres to design principles and best practices, including consistency in layout, typography, and color scheme. Solicit feedback from users and stakeholders to identify areas for improvement in usability and user satisfaction.

**Security Testing:**

Perform vulnerability assessments and penetration testing to identify and address potential security risks. Test authentication mechanisms to ensure secure user access and prevent unauthorized access to sensitive data. Verify that sensitive data such as user credentials, payment information, and personal information are encrypted and protected from unauthorized access. Implement measures to prevent common security threats such as SQL injection, cross-site scripting (XSS), and cross-site request forgery (CSRF).

## Unit Testing

Unit testing is usually conducted as part of a combined code and unit test phase of the software lifecycle, although it is not uncommon for coding and unit testing to be conducted as two distinct phases.

### Test strategy and approach

Field testing will be performed manually and functional tests will be written in detail.

### Test objectives

* + - * To test every part of the code.
      * Check every part of the code is running without any compile time error.

### Features to be tested

* + - * To check the importing of modules is doing good.
      * To check the dataset is loaded perfectly.
      * To check the Visualization is done well.
      * To check for any compile time and run time errors.
      * To check every requirement is installed.

## Integration Testing

Software integration testing is the incremental integration testing of two or more integrated software components on a single platform to produce failures caused by interface defects.

The task of the integration test is to check that part of code or the piece of code,

e.g. components in a software system or – one step up – software applications at the company level – interact without error.

**Test Results:** All the test cases mentioned above passed successfully. No defects encountered

## VALIDATION

The testing process is a part of a broader subject referring to verification and validation. We have acknowledged the system specification and tried to meet the customers’ requirements and for this sole purpose, we have to verify and validate the product to make sure everything is in place. Verification and Validation are two different things. One is performed to ensure that the software correctly implements a specific functionality and other is due to ensure if the customer requirements are properly met or not by the end product.

**7.4.1 Introduction**

Validation is a critical aspect of any software application, ensuring that input data meets specified criteria and is safe for processing. In the context of our event management project, validation plays a crucial role in maintaining data integrity, security, and consistency throughout the application.

**7.4.2 Purpose**

The purpose of this documentation is to provide guidelines and best practices for implementing validation in our event management project using Zod and strict TypeScript. By adhering to these guidelines, we aim to ensure that our application's data is validated effectively, reducing the risk of errors, security vulnerabilities, and data inconsistencies.

**7.4.3 Tools and Technologies**

**Zod:** Zod is a TypeScript-first schema declaration and validation library that provides a simple and expressive way to define data schemas and validate input data against those schemas.

**Strict TypeScript:** TypeScript is a statically typed superset of JavaScript that adds static typing capabilities to the language. By enabling strict TypeScript mode, we enforce stricter type checking rules, catching potential errors at compile time and improving code quality and reliability.

**7.4.4** **Guidelines**

**Define Data Schemas with Zod**

Use Zod to define data schemas for various entities in the application, such as events, users, registrations, payments, and feedback submissions.

- Define schemas using the `z.object()` function, specifying the structure, type, and constraints of each field in the schema.

- Leverage Zod's built-in validation methods to enforce constraints such as required fields, data types, string lengths, enumerations, and custom validation rules.

**Validate Input Data**

Validate input data against the defined schemas before processing or persisting it in the application.

Use Zod's `parse()` or `safeParse()` functions to validate input data, handling both successful parsing and validation errors gracefully.

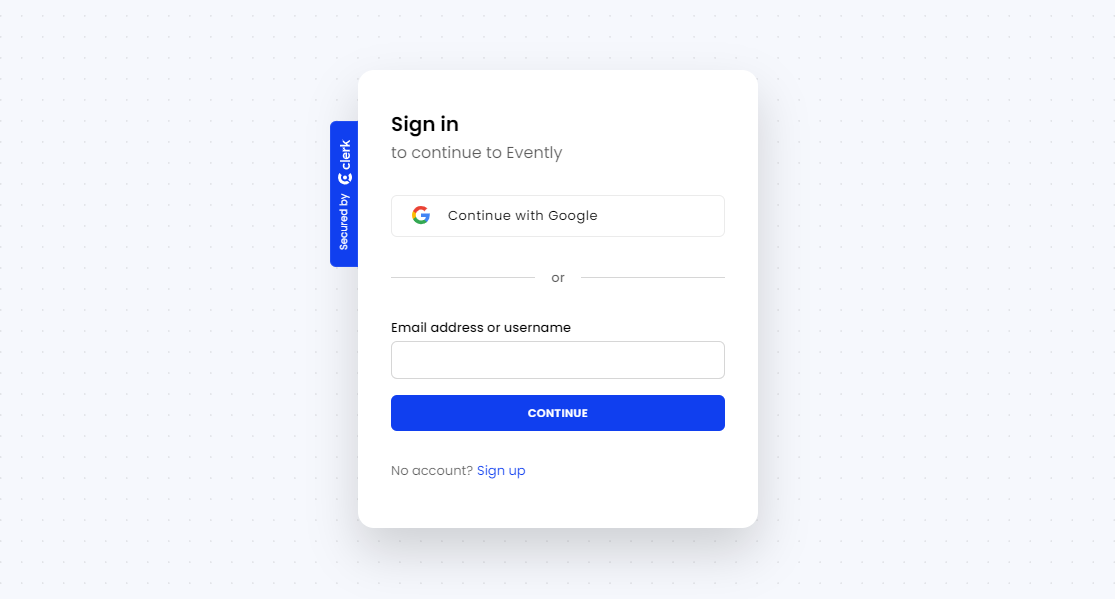
Leverage TypeScript's type inference capabilities to ensure that validated data is strongly typed and compatible with the rest of the application.

**Handle Validation Errors**

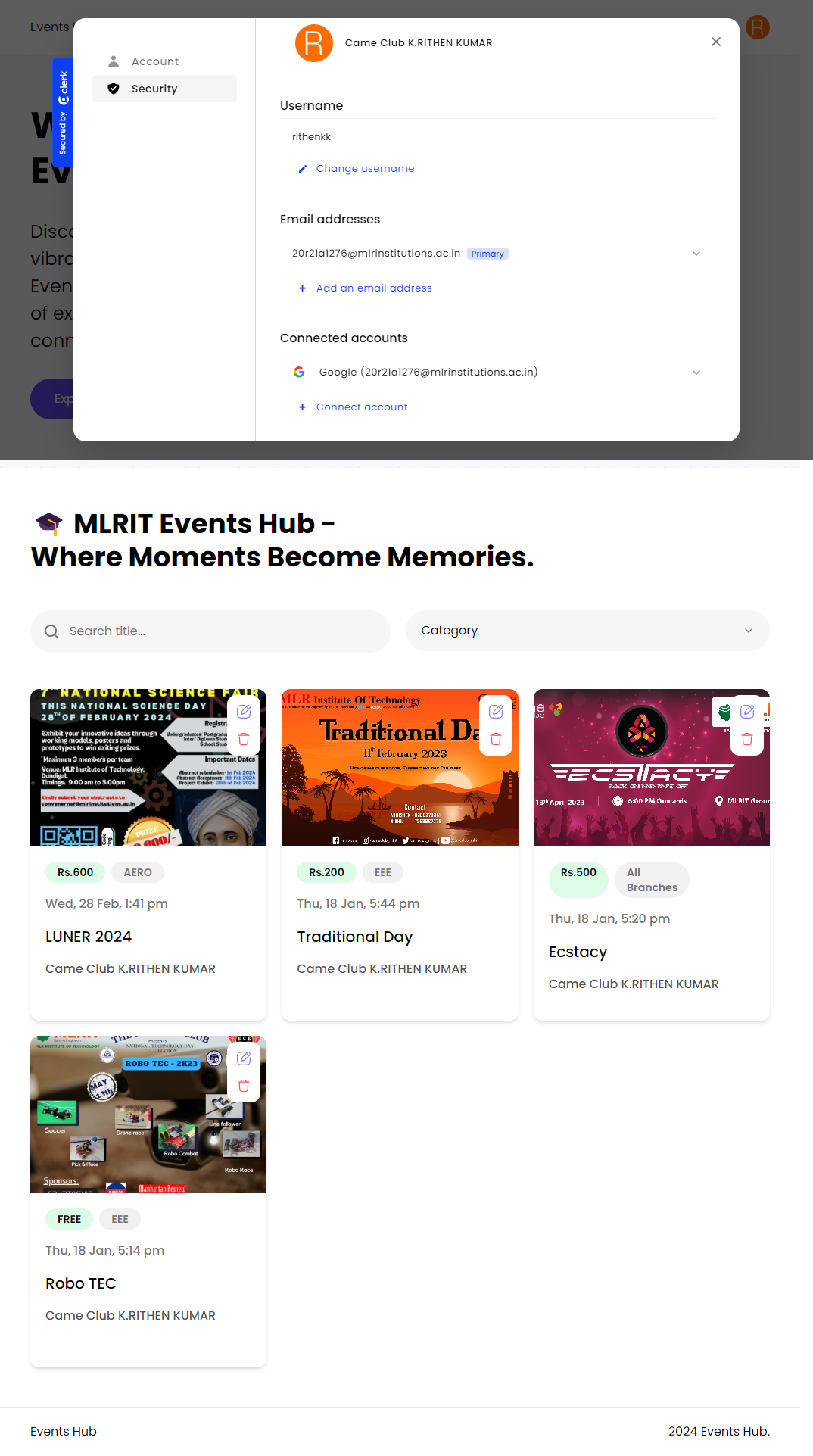
Handle validation errors gracefully by providing meaningful error messages and feedback to users. Use descriptive error messages to indicate which fields failed validation and the reason for the validation failure. Display validation errors in the user interface to guide users in correcting input errors and submitting valid data.

# 8 RESULTS

**8.1 User Login/Signup**

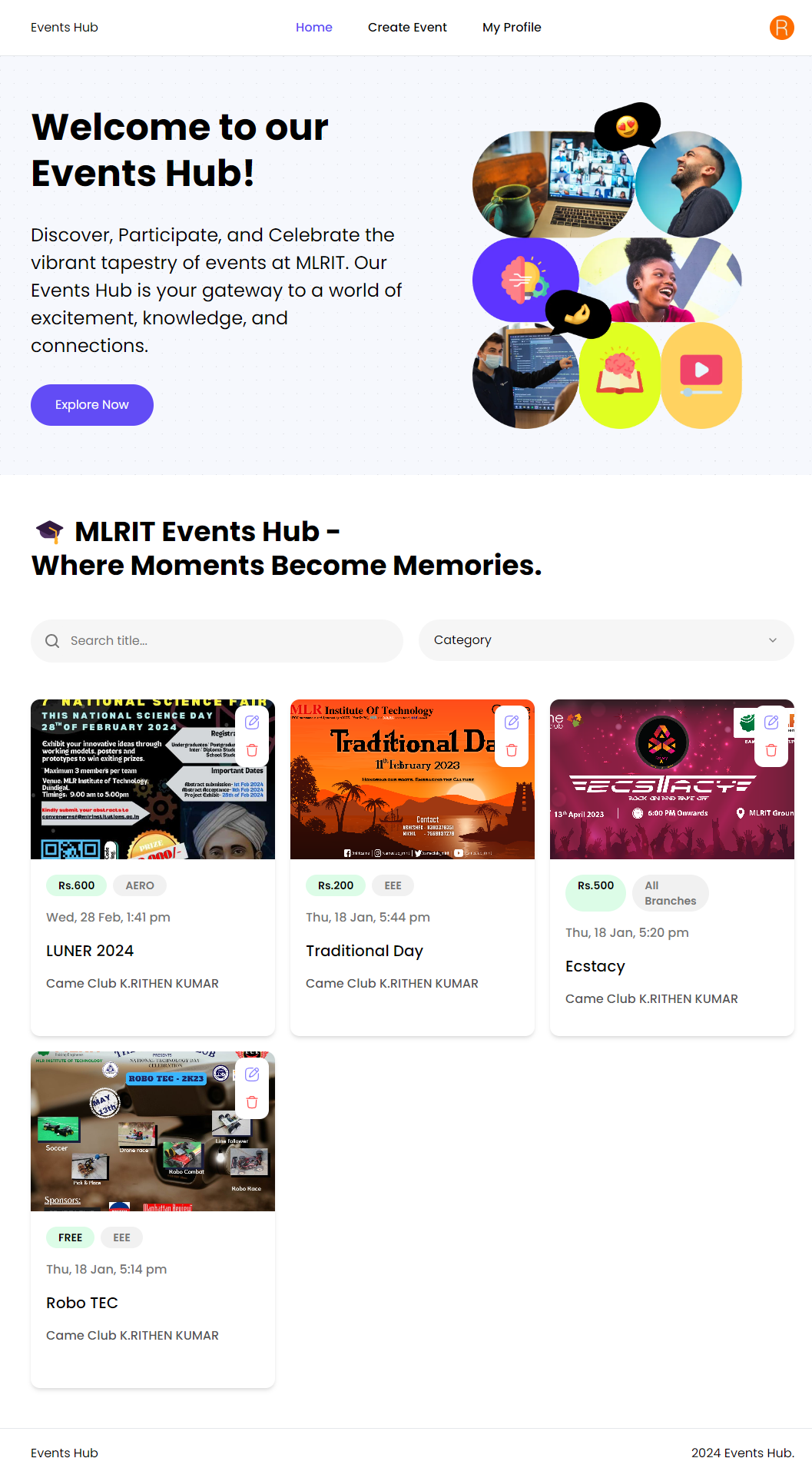


**Fig.1 : Login Page with Signup and Google Sign-In**



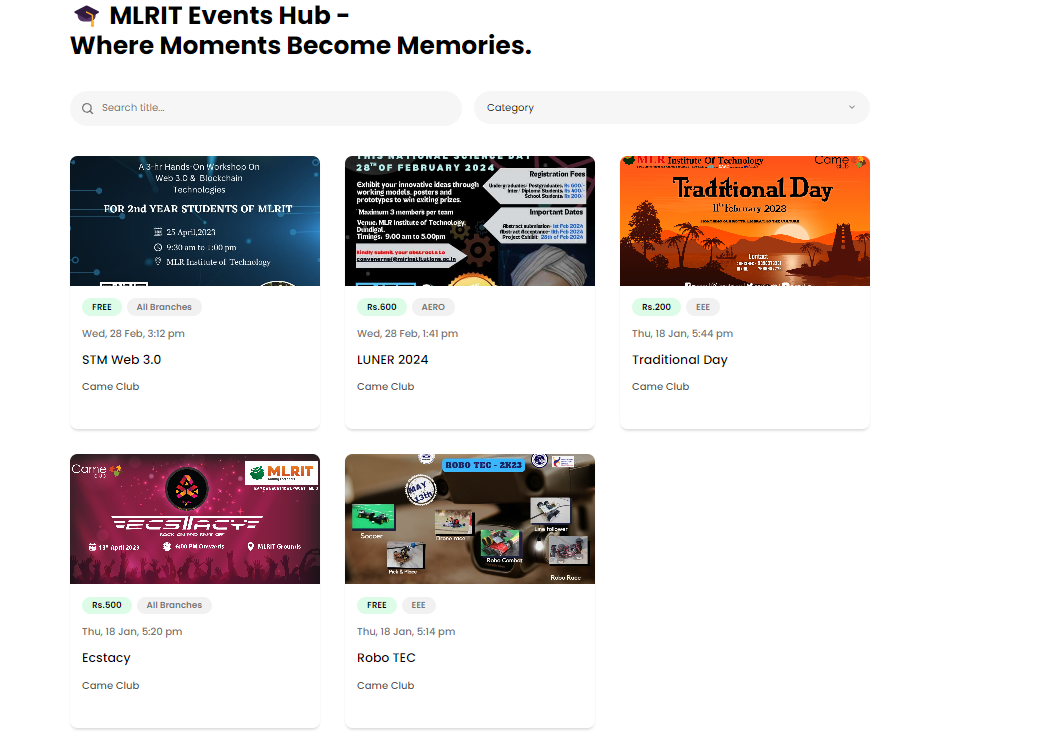
### Fig.2 : Manage User Profile

**8.2 Home Page**

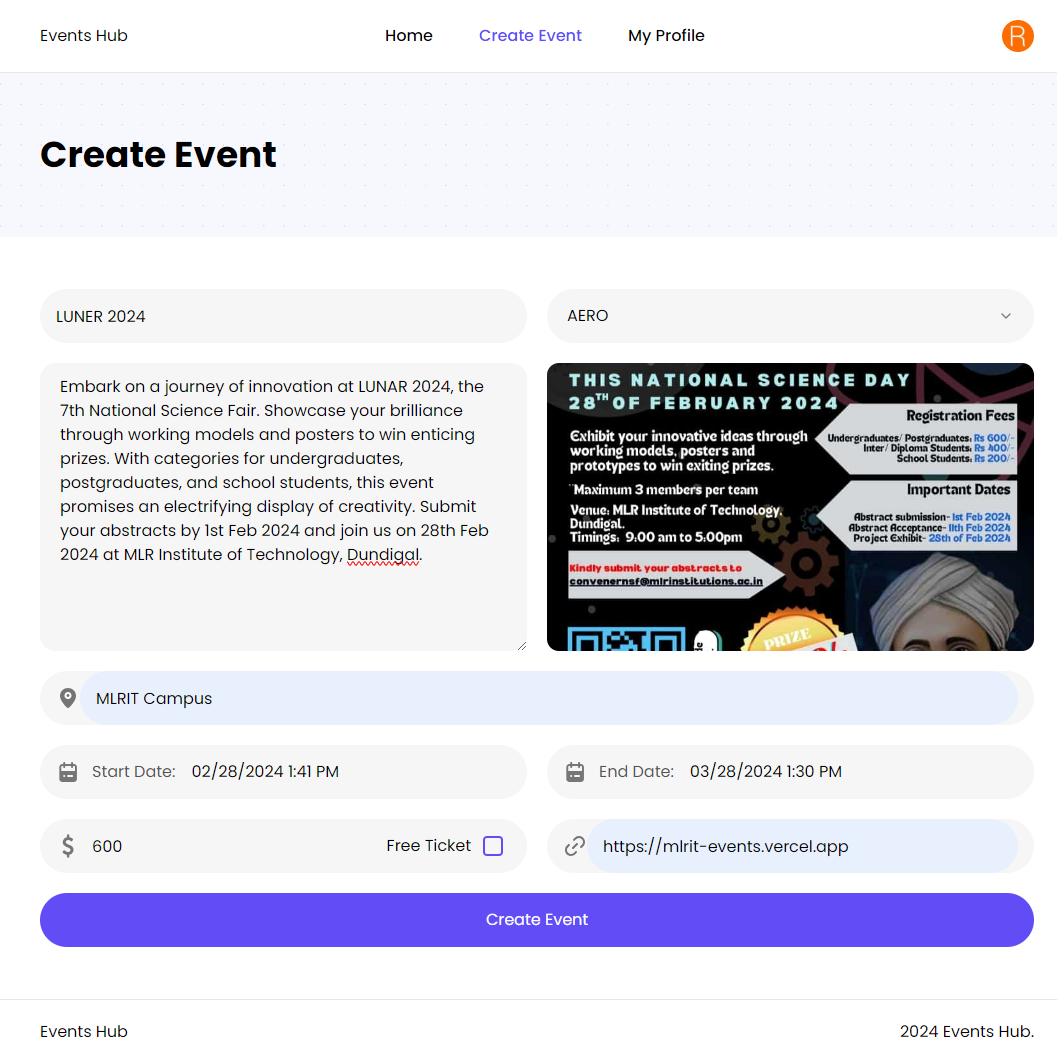


### Fig.3 : Home Page

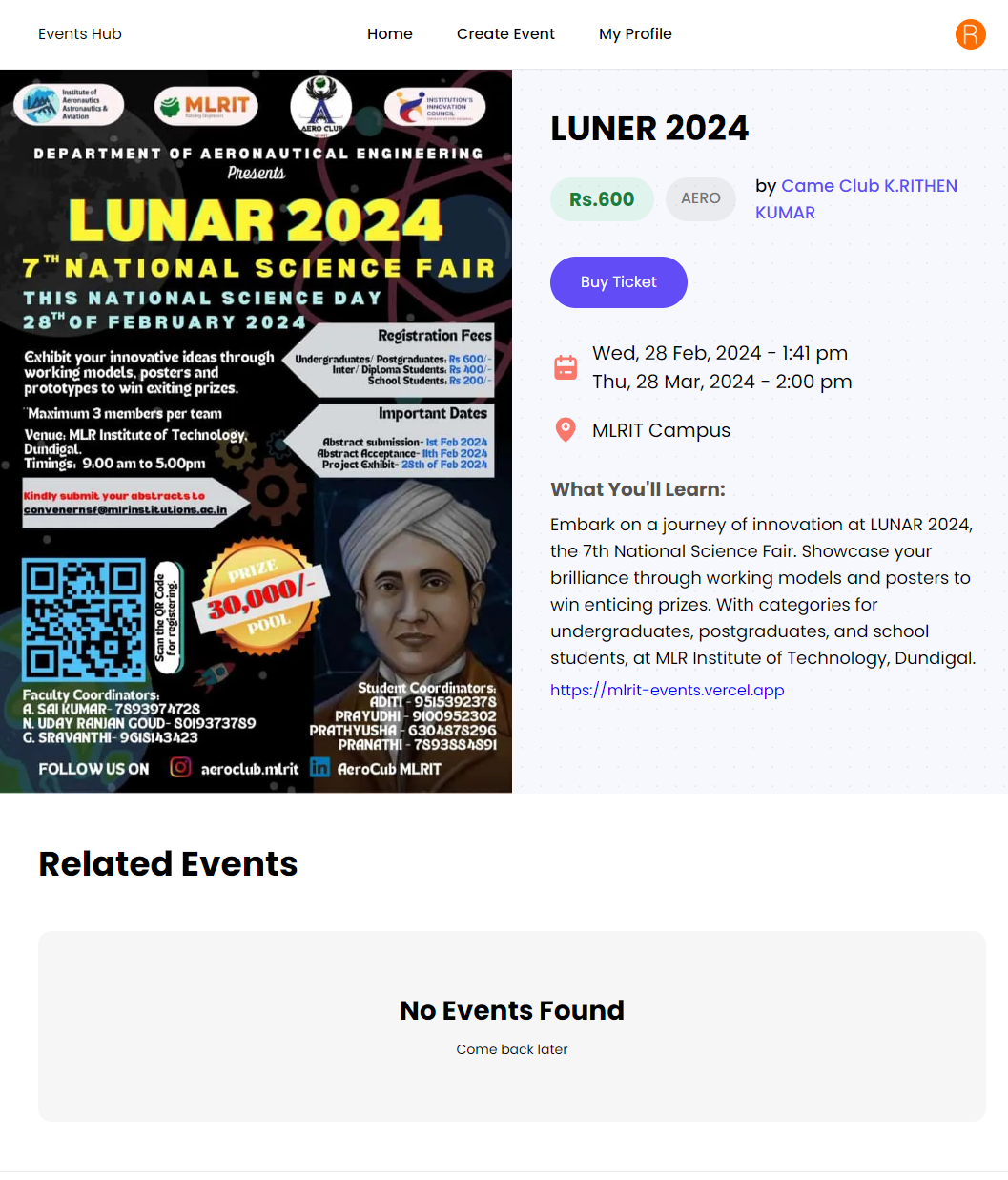
**8.3 Events Page**



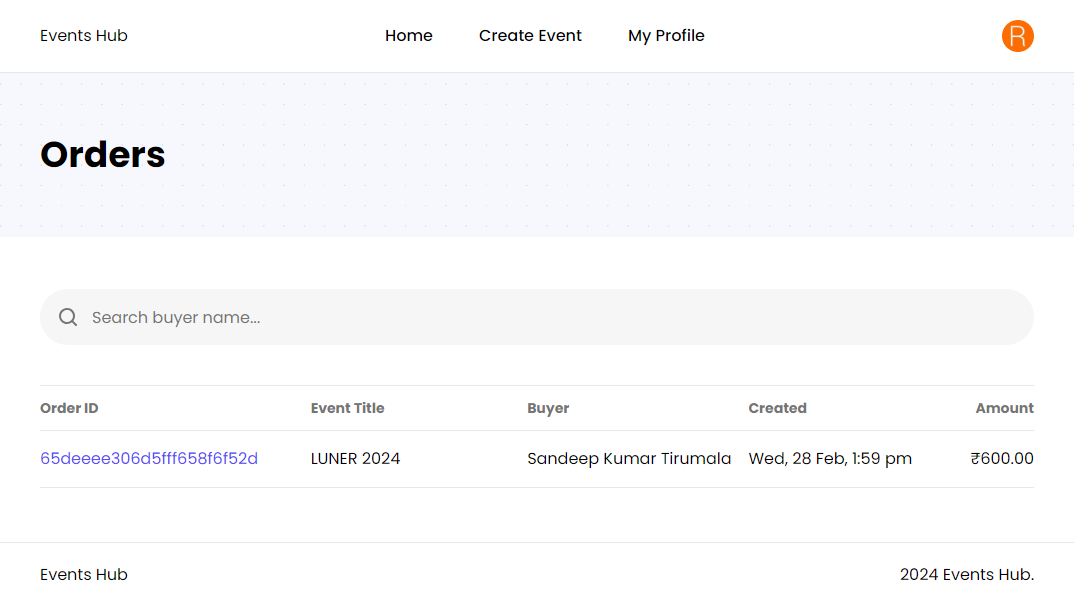
### Fig.4 : Events Page



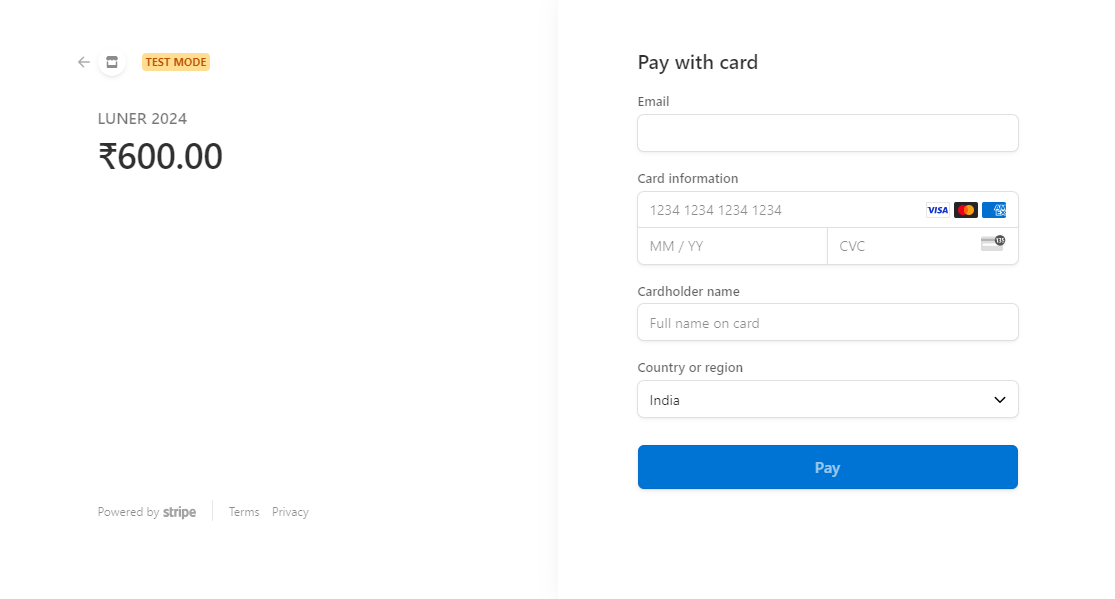
**Fig.3 : Create Event**



**Fig.4 : Related Events Page**



### Fig.5 : Order Details



**Fig.6 : Payment Page**

# 9. CONCLUSION

**CONCLUSION AND FURTHER SCOPE**

In conclusion, the development of the College Events Hub project using a Full Stack Next 14 MERN architecture with Stripe integration, TypeScript, TailwindCSS, and other modern technologies has resulted in a comprehensive solution for managing events within college campuses. By leveraging the power of Node.js, Next.js, and MongoDB on the backend, and React.js on the frontend, along with the integration of Stripe for secure payment processing, the platform offers a robust and scalable solution for event organizers and participants alike. The use of TypeScript enhances code quality and maintainability, while TailwindCSS provides a flexible and efficient way to style the user interface. With features such as user authentication, event creation, ticket purchasing, and order management, the College Events Hub facilitates seamless communication, engagement, and event registration within the college community.

.

# 10. REFERENCES

1. MDN Web Docs (https://developer.mozilla.org/): MDN Web Docs is a comprehensive resource for web development documentation, covering HTML, CSS, JavaScript, and various web APIs, which can be useful for frontend development in the project.
2. MongoDB University (https://university.mongodb.com/): MongoDB University offers free online courses and tutorials on MongoDB database, providing valuable insights and knowledge for backend development using MongoDB Atlas.
3. Stripe Documentation (https://stripe.com/docs): The Stripe documentation provides detailed guides and tutorials on integrating Stripe payment processing into web applications, which is essential for implementing payment functionality in the College Events Hub project.
4. Tailwind CSS Documentation (https://tailwindcss.com/docs): The Tailwind CSS documentation offers comprehensive guidance on using Tailwind CSS for styling web applications, helping developers design modern and responsive user interfaces.
5. Node.js Design Patterns" by Mario Casciaro: This book provides insights into best practices and design patterns for building scalable and efficient Node.js applications, which can be valuable for the backend development of the College Events Hub.
6. "React.js Essentials" by Artemij Fedosejev: This book offers a comprehensive guide to building web applications with React.js, covering essential concepts and techniques that are useful for frontend development in the project.
7. Zod Documentation: https://github.com/colinhacks/zod#readme
8. React Hook Form: https://react-hook-form.com/get-started
9. Shadcn Documentation: <https://github.com/shadcn/shadcn#readme>
10. J. Smith et al., "Enhancing event management through mobile application technology," International Journal of Event Management Research, (2021).
11. A. Johnson, "Utilizing virtual reality for immersive event experiences," Journal of Event Technology and Management, (2020).
12. S. Patel et al., "Exploring the impact of social media integration on event promotion and attendee engagement," International Journal of Event Planning and Management, (2019).
13. M. Garcia et al., "Innovative approaches to event ticketing and registration systems," Journal of Event Operations and Security Management, (2018).
14. K. Lee et al., "Utilizing blockchain technology for secure event ticketing and transactions," International Journal of Event Management Research, (2017).
15. R. Chen et al., "Implementing AI-driven chatbots for event attendee assistance and support," Journal of Event Technology and Management, (2016).
16. T. Nguyen et al., "Enhancing event logistics and operations through," International Journal of Event Planning and Management, (2015).
17. - Express.js Documentation (https://expressjs.com/): Official documentation for Express.js, a fast, unopinionated, minimalist web framework for Node.js, which is used for building the backend of web applications.
18. - TypeScript Documentation (https://www.typescriptlang.org/docs/): Comprehensive documentation for TypeScript, a superset of JavaScript that adds static typing to the language, providing type-checking capabilities during development.
19. - MongoDB Documentation (https://docs.mongodb.com/): Official documentation for MongoDB, a NoSQL database used for storing and managing data in the project.
20. - Mongoose Documentation (https://mongoosejs.com/docs/): Documentation for Mongoose, an object modeling library for MongoDB and Node.js, which provides a schema-based solution for modeling application data.
21. - Cypress Documentation (https://docs.cypress.io/guides/overview/why-cypress.html): Documentation for Cypress, an end-to-end testing framework for web applications, which can be used to write and run tests that simulate real user interactions.
22. - Heroku Documentation (https://devcenter.heroku.com/): Documentation for Heroku, a platform as a service (PaaS) that enables developers to deploy, manage, and scale applications in the cloud.
23. - GitHub Actions Documentation (https://docs.github.com/en/actions): Documentation for GitHub Actions, a continuous integration and continuous deployment (CI/CD) platform provided by GitHub, which can be used for automating workflows in the project.
24. - React Testing Library Documentation (https://testing-library.com/docs/react-testing-library/intro/): Documentation for React Testing Library, a testing utility for React applications that focuses on testing components in a way similar to how they are used by end users.
25. - Jest Documentation (https://jestjs.io/docs/getting-started): Documentation for Jest, a JavaScript testing framework that provides a simple and powerful way to write tests for JavaScript and TypeScript code.
26. - Stripe API Reference (https://stripe.com/docs/api): Detailed API reference documentation for the Stripe payment processing platform, providing information on available endpoints, request parameters, response objects, and authentication methods.
27. - Stripe Developer Documentation (https://stripe.com/docs/development): Comprehensive developer documentation for Stripe, covering topics such as integration guides, testing strategies, security best practices, and troubleshooting tips.
28. - Clerk Documentation (https://docs.clerk.dev/): Official documentation for Clerk, an authentication and user management solution for web applications, offering guides, tutorials, and API reference documentation for integrating Clerk into projects.