1. INTRODUCTION

1.1 Problem Statement

Kristu Jayanti College, one of the leading higher education institutions, has a diverse student body of nearly fifteen thousand students spread across multiple departments, programs, and academic years. While the college thrives academically and culturally, there exists a significant communication and connectivity gap among students. Most students interact primarily within their immediate circle, classmates, departmental peers, or friends from familiar groups — leaving little room for inter-departmental collaboration or exploration of shared interests beyond their own disciplines.

In an institution this large, there are countless students with varied passions such as filmmaking, coding, music, photography, startups, dance, and creative writing. However, due to the absence of a unified communication channel, students often find it difficult to connect with others who share similar talents or project ideas. For instance, a student from the Computer Science department wishing to create a short film might need actors from the Humanities department, a scriptwriter from English, and a photographer from Commerce. Despite all of these students being on the same campus, their paths rarely cross because there is no structured way to discover or reach one another. This lack of exposure and accessibility results in missed opportunities for collaboration, learning, and innovation.

Currently, students rely on informal channels like WhatsApp groups, Instagram pages, or word-of-mouth to seek collaborators or spread event updates. These methods are often unreliable, unverified, and restricted to small groups. Furthermore, they lack any security or authentication layer to ensure that participants are genuine members of the college community. This raises issues related to privacy, misinformation, and exclusivity. College-wide events, workshops, and opportunities often fail to reach the right audience because announcements are confined to departments or notice boards, leading to a fragmented flow of information.

There is a growing need for a dedicated, secure, and user-friendly digital platform that bridges this gap by bringing together all students under a single verified ecosystem. Such a platform should allow students to discover peers with shared interests, collaborate on creative or technical projects, and stay informed about campus activities. The solution must ensure that only authenticated Kristu Jayanti College students — verified through their official college email IDs — can register and participate, maintaining privacy and trust within the network.

KJ_Connect is designed precisely to address these needs. It is a web application that enables

students to connect, communicate, and collaborate based on their interests and talents. Users can create or join groups aligned with areas like filmmaking, coding, dance, startups, or photography. The platform includes secure authentication, personalized profiles, real-time chat, and event updates, creating a digital ecosystem that mirrors the vibrancy of the physical campus. It serves as a bridge between departments, encouraging cross-disciplinary teamwork and fostering a sense of community beyond academic boundaries.

By promoting interaction among diverse groups, KJ_Connect transforms the student experience from isolated departmental engagement to holistic campus connectivity. It empowers students to explore new opportunities, discover hidden talents, and work together on initiatives that extend beyond classrooms. In essence, KJ_Connect aims to cultivate a collaborative and innovative college environment where every student can contribute, connect, and create without barriers.

1.2 Project Description

KJ_Connect is an innovative web-based platform designed to unify the large and diverse student community of Kristu Jayanti College under a single digital network. With more than 14,000 students spread across various departments and courses, the college has a vast pool of untapped talent and creativity. However, due to limited interaction across departments, students often miss out on opportunities to collaborate, share ideas, and participate in multidisciplinary projects. KJ_Connect aims to bridge this gap by providing a secure and verified space where students can connect, communicate, and collaborate based on shared interests and skills.

The platform allows students to create personal profiles by signing up with their official college email IDs, ensuring that only authenticated members of the college can access the application. Once verified, users can showcase their interests such as filmmaking, coding, startups, photography, dance, or music. They can then explore, create, or join groups aligned with these interests. Each group acts as a mini-community where members can post updates, discuss ideas, plan events, and collaborate on projects. This structure encourages students to move beyond departmental boundaries and form meaningful professional and creative relationships across the college.

The application includes a real-time chat system, enabling seamless communication between users. Students can send messages, participate in group discussions, or connect with peers one-on-one for project collaborations. A dedicated event section provides updates on campus activities, competitions, and workshops, ensuring that students never miss out on college-wide opportunities. This feature also helps college clubs and student organizations promote their

events more efficiently and reach a broader audience. The user-friendly interface is designed to provide a smooth navigation experience, making it easy for even first-time users to explore the platform's features.

KJ_Connect is developed using modern web technologies for performance, security, and scalability. The frontend is built with Next.js (v15.5.3) and React (v19.1.0), ensuring a dynamic and responsive user experience. TypeScript is used across both frontend and backend for enhanced code reliability and maintainability. The backend operates on Node.js, utilizing Next.js API routes to manage requests such as user authentication, profile creation, and message handling. MySQL, hosted locally via XAMPP, serves as the primary database for storing user details, group information, and chat records. The mysql2 package facilitates communication between the application and the database, while bcryptjs ensures secure password hashing during registration and login. Additionally, the OpenAI SDK is integrated for AI-based suggestions or intelligent automation in future enhancements.

The system architecture is designed to maintain a clear separation between frontend and backend processes, ensuring smooth data flow and consistent performance. Users interact with the interface through browser-based pages, while API routes handle server-side operations like authentication, data retrieval, and message synchronization. The use of environment variables (.env) ensures sensitive data such as database credentials and API keys remain secure.

Beyond its technical foundation, the project's real impact lies in its purpose — building a stronger, more connected student community. KJ_Connect fosters interdisciplinary communication, allowing students to discover hidden talents, form new friendships, and collaborate on creative or technical initiatives. It also helps students gain exposure to diverse fields, encouraging them to think beyond academics and explore practical applications of their skills. In the long run, the platform can evolve into an official digital hub for Kristu Jayanti College, integrating club management, project showcases, and event registrations.

In essence, KJ_Connect is more than just a social platform — it is a community-building ecosystem that empowers students to connect, collaborate, and create. By blending technology with campus culture, it transforms how students interact and grow together, ensuring that every Jayantian has the opportunity to network, innovate, and make an impact within their college community.

2. SYSTEM STUDY

2.1 Existing System

At present, there is no centralized digital platform within Kristu Jayanti College that allows students from different departments and courses to connect, collaborate, or share opportunities based on mutual interests. Communication and networking are limited to informal, fragmented channels that are often inefficient, unreliable, and lack security. The existing system primarily relies on the following methods:

1. Departmental and Class-Based WhatsApp Groups:

Most students interact through WhatsApp groups created for specific classes or departments. These groups are useful for academic updates but do not promote collaboration across different courses or faculties. Students from other departments or academic years rarely get included, creating silos of communication within the campus.

2. Social Media Platforms (Instagram, Facebook, etc.):

Students occasionally use public platforms like Instagram or Facebook to share information about events, clubs, or creative projects. However, these platforms are not college-exclusive, allowing anyone outside the institution to access or comment on posts. This raises privacy and authenticity concerns and makes it difficult to verify if participants are genuine students of Kristu Jayanti College.

3. Word-of-Mouth and Offline Communication:

Many students rely on personal connections or offline conversations to find collaborators for projects, competitions, or events. This process is slow and depends heavily on mutual acquaintances, making it difficult for students to discover new talents or opportunities beyond their existing circles.

4. Limited Event Visibility:

Information about college events, workshops, or competitions is often shared only through departmental notices, faculty announcements, or posters on campus boards. As a result, many students remain unaware of events happening outside their department or area of study, missing out on valuable participation opportunities.

5. Lack of Verified Student Network:

There is no system that ensures only current Kristu Jayanti College students can register or participate in discussions. Consequently, it becomes challenging to maintain a safe and verified network where users can confidently interact and collaborate without external interference.

Due to these limitations, the existing system fails to provide an integrated and secure

environment where students can explore interdisciplinary collaborations or stay informed about campus-wide opportunities. It restricts communication to small, closed groups and prevents the formation of a unified student community.

This disconnected environment leads to missed chances for creative projects, limited awareness of talent across departments, and underutilization of the college's vibrant student potential. These drawbacks highlight the need for a dedicated platform like KJ_Connect, which centralizes communication, ensures authenticity through verified college IDs, and provides a structured digital ecosystem for collaboration, networking, and campus engagement.

2.2 Proposed System

The proposed system, KJ_Connect, is a secure and collaborative web application developed to connect the diverse student community of Kristu Jayanti College under a single digital platform. With over 14,000 students enrolled across various departments, there exists a strong need for a structured and verified system that enables students to discover, interact, and collaborate beyond academic boundaries. KJ_Connect addresses this need by offering a college-exclusive network that allows users to connect based on shared interests, form groups, and communicate in real-time — all within a safe and authenticated environment. The system ensures that only genuine students of Kristu Jayanti College can register and use the platform by implementing college email ID verification. Once registered, users can create detailed profiles highlighting their personal information, department, skills, and interests. Based on these interests, students can create or join groups in areas such as filmmaking, coding, startups, photography, dance, and music. Each group serves as a digital community where members can interact, post updates, organize events, and collaborate on creative or technical initiatives. The platform is designed with a real-time chat system, allowing students to communicate effectively within groups or through private messages. Additionally, a centralized event and announcement section enables students and clubs to post updates about workshops, competitions, or cultural programs. This ensures that every user remains aware of ongoing and upcoming college events, promoting active participation and engagement across all departments.

Key Features of KJ_Connect

- **Secure Authentication:**
  Only students with verified @kristujayanti.com email addresses can register.
  Passwords are securely encrypted using bcryptjs for enhanced data protection.

- **User Profile Management:**
  Students can create personalized profiles displaying their department, batch, and interests.
  Profiles help others identify potential collaborators for projects or events.
- **Interest-Based Groups:**
  Students can create or join groups based on shared hobbies, skills, or goals.
  Each group has its own discussion feed and chat feature for seamless collaboration.
- **Real-Time Chat System:**
  Integrated chat functionality using Next.js API routes and Node.js backend.
  Allows group discussions and private messaging between verified users.
- **Event and Announcement Section:**
  Displays all upcoming college events, workshops, and activities.
  Enables clubs or student coordinators to reach a wider audience efficiently.
- **Search and Discovery:**
  Advanced search functionality to find students, groups, or activities by keyword or interest.
  Helps students connect with peers across departments.
- **Admin and Moderation Controls:**
  Admin users can verify content, manage reports, and oversee group creation to maintain platform integrity.
- **Scalable Architecture:**
  Built with Next.js (v15.5.3) and React (v19.1.0) for dynamic front-end performance.
  Backend powered by Node.js, with data stored in MySQL via mysql2.
  Uses TypeScript for type safety and maintainable code.
- **AI-Ready Integration (Future Scope):**
  Integration with OpenAI SDK for intelligent suggestions such as recommending groups or potential collaborators.

The KJ_Connect system effectively overcomes the limitations of the existing fragmented communication structure by providing a unified, secure, and student-centric networking environment. It encourages interdisciplinary collaboration, enhances participation in campus events, and fosters a culture of innovation and teamwork. By integrating modern technologies and a strong focus on verified connectivity, KJ_Connect transforms the college experience into an inclusive, engaging, and technologic ally empowered community network — ensuring every

student has the opportunity to connect, collaborate, and grow.

2.3 ER Diagram

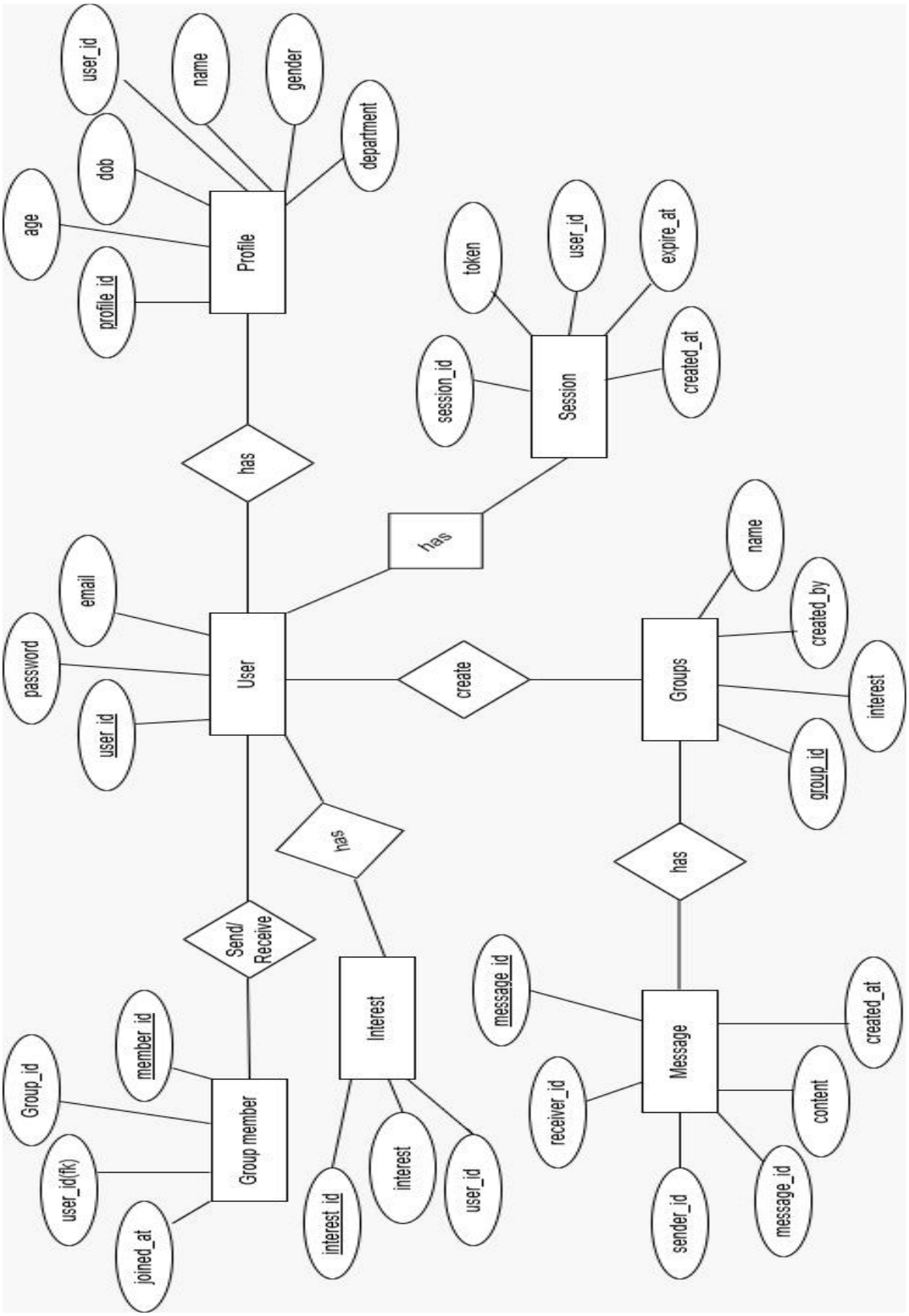


Figure 2.1 – ER Diagram

2.4 Use Case Diagram

A Use Case Diagram highlights the functional requirements of the system by showing how different actors (users) interact with the system's various functions. In KJ_Connect, the primary actor is the Student, and the system use cases include Register, Login, Create/Join Groups, Send Messages, Explore Events, and Logout.

Key Points:

- Represents the relationship between users and system features.
- Useful for understanding user interactions and system boundaries.
- Helps developers and stakeholders grasp system functionality at a glance.

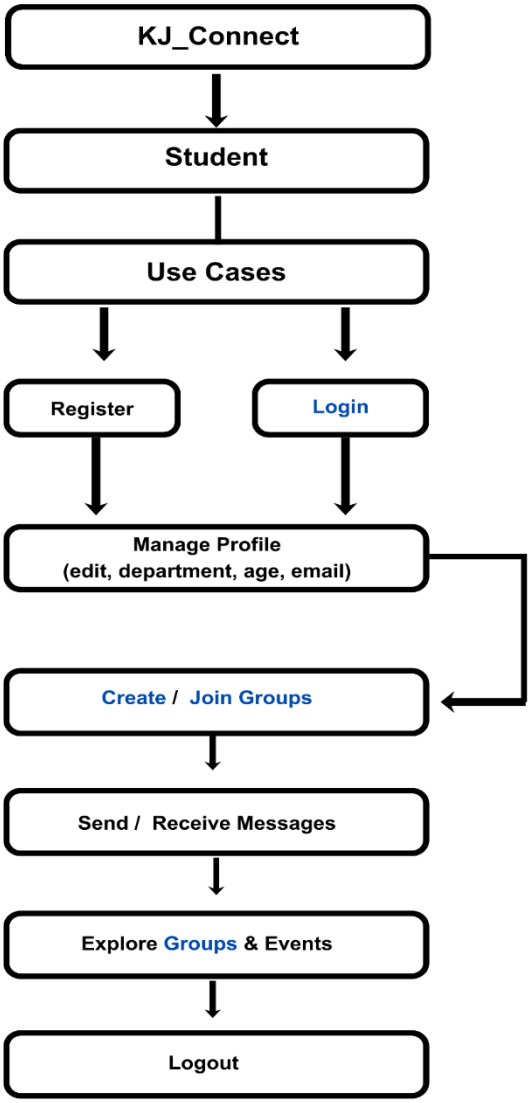


Figure 2.2 – Use Case Diagram

2.5 Sequential Diagram

A Sequence Diagram shows the interaction between different system components (like frontend, backend, and database) in a time-sequenced order. It visualizes how messages are exchanged to complete a particular operation.

Key Points:

- Focuses on the chronological flow of messages between components.
- Demonstrates how data moves through the system.
- Useful for identifying communication logic and dependencies.
- In KJ_Connect, it outlines processes such as User Login, Data Retrieval, and Message Exchange between Frontend → Backend → Database.

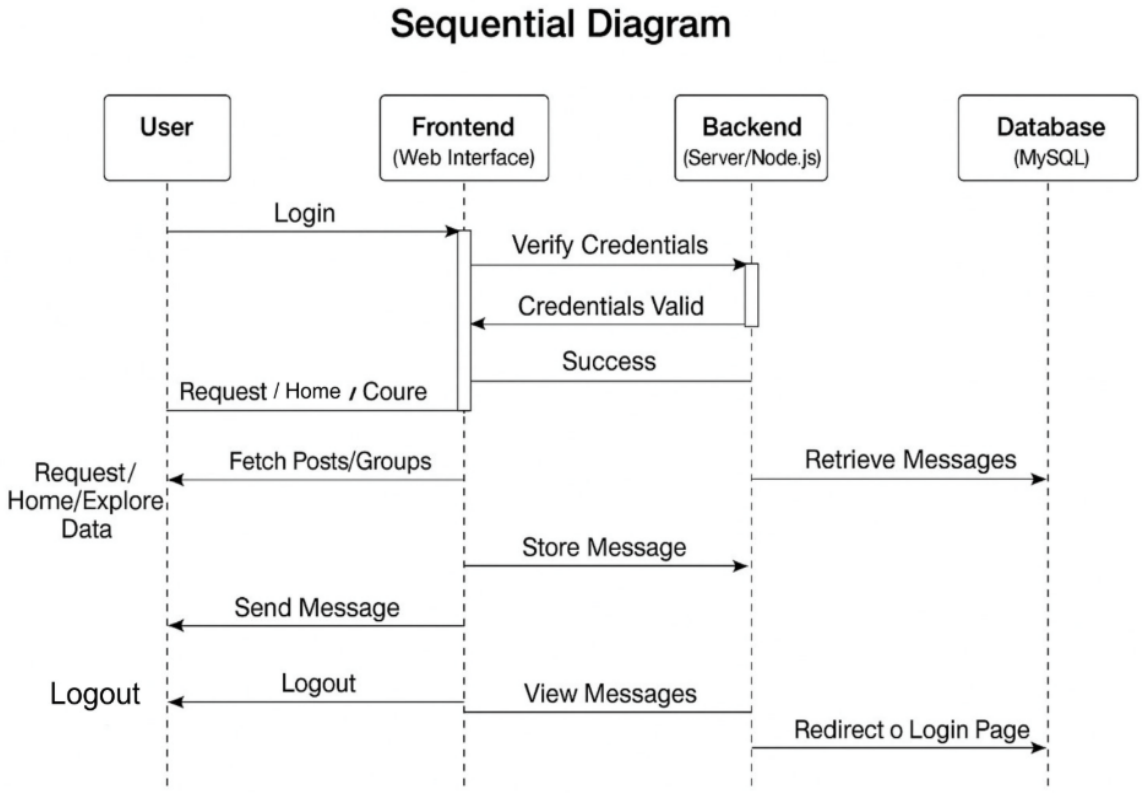


Figure 2.3 – Sequential Diagram

2.6 Activity Diagram

An Activity Diagram represents the workflow or step-by-step execution of processes within the system. It visually depicts how a user interacts with the application from start to end — for example, logging in, exploring groups, sending messages, and logging out. It helps in understanding the flow of control and decision-making in the system, making it easier to visualize user activities and system responses.

Key Points:

- Shows the sequence of actions performed in a process.
- Includes decision nodes and control flows.
- Helps identify process logic and user interaction paths.
- In KJ_Connect, it maps activities like Login/Register → Explore → Join Groups → Message → Logout.

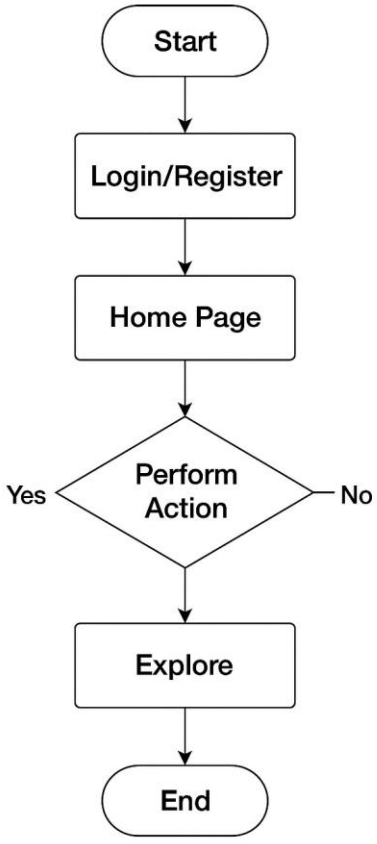


Figure 2.4 – Activity Diagram

3. SYSTEM CONFIGURATION

The KJ_Connect web application is built using advanced and efficient technologies that combine both hardware and software resources to deliver a seamless and responsive user experience. Since the project encompasses a variety of core features such as user registration and authentication, group creation and management, real-time chat, and event updates, it demands a well-structured and optimized system configuration. This ensures the platform operates smoothly during all phases — development, testing, and deployment — without performance bottlenecks or compatibility issues.A properly defined system configuration guarantees smooth performance, data security, high scalability, and cross-device responsiveness. The application is designed to adapt efficiently across desktops, laptops, tablets, and smartphones, ensuring consistent usability for all students. The use of Next.js and React allows for fast rendering and dynamic UI components, while Node.js and MySQL handle backend processing and data management reliably.To maintain a stable environment, both development and production setups must remain consistent. This minimizes the chances of runtime errors, improves debugging efficiency, and ensures accurate database connectivity. Proper synchronization between frontend, backend, and database layers also enables the implementation of modern web features such as real-time chat, notifications, and interactive group updates.

Additionally, the configuration supports multi-user concurrency, allowing numerous students to access the platform simultaneously without lag or server overload. Efficient use of caching, optimized database queries, and asynchronous programming through Node.js contribute to faster data processing and reduced latency. The inclusion of environment variables (.env) ensures secure storage of sensitive information like database credentials and API keys, strengthening overall application security.

In essence, the system configuration forms the backbone of the KJ_Connect project. It ensures that every component — from frontend design to backend logic and database communication — functions in perfect coordination. By combining robust hardware resources with a modern software stack, KJ_Connect delivers a stable, efficient, and future-ready platform that enhances connectivity, collaboration, and engagement among students of Kristu Jayanti College.

3.1 Hardware Configuration

A higher-end configuration enhances frontend rendering, database querying, and build speed during development. It also allows developers to test multiple sessions or features simultaneously without system lag.

Component	Minimum Requirement	Recommended Requirement
Processor	Intel Core i3 or equivalent	Intel Core i5 / i7 or higher
RAM	8 GB	16 GB or higher
Hard Disk	250 GB HDD	500 GB SSD (for faster builds)
Operating System	Windows 10, macOS, or Linux	Latest Windows, macOS, or Linux
Internet Connection	Minimum 5 mbps	20 Mbps or higher (for SDKs)

Table 3.1 Hardware Configuration

3.2 Software Configuration

The software configuration of *KJ_Connect* defines the essential tools, frameworks, and technologies required to develop, test, and deploy the application efficiently. The system is built on a full-stack JavaScript environment that ensures scalability, performance, and ease of maintenance.

The frontend is developed using Next.js (v15.5.3) and React (v19.1.0) for fast rendering, routing, and creating interactive user interfaces. The use of TypeScript enhances code readability and minimizes runtime errors through static typing. The backend is powered by Node.js, which handles API requests, authentication, and communication with the MySQL database managed via XAMPP. The mysql2 package connects the backend to the database, while bcryptjs ensures secure password hashing for user authentication.

Component	Specification / Technology Used	Purpose
Operating System	Windows 10 / 11, Linux, or macOS	Development & Deployment environment
Frontend Technology	React, Next.js, TypeScript, HTML, CSS	User interface and interaction
Backend Technology	Node.js,Next.js	API and business logic
Database	MySQL (via XAMPP)	Data storage and retrieval
Development Tools	VS CODE	Coding, testing, and debugging

Table 3.2 Software Configuration

4. SOFTWARE DESCRIPTION

The KJ_Connect web application is a full-stack platform designed to connect students of Kristu Jayanti College based on their interests. It is developed using Next.js, React, and TypeScript for the frontend, providing a fast, interactive, and responsive user interface. The backend is powered by Node.js and MySQL, ensuring secure data management, smooth authentication, and efficient API handling. Supporting tools like bcryptjs for password encryption, mysql2 for database connectivity, and dotenv for environment configuration enhance the project's reliability and security. The system allows users to register with their college email, create or join interest-based groups, chat in real time, and stay updated on events — offering a seamless and secure networking experience for students.

4.1 Overview Frontend

The frontend of *KJ_Connect* serves as the visual and interactive interface that connects students to the platform's core functionalities such as registration, group creation, chatting, and exploring college events. It is designed to provide a smooth, responsive, and user-friendly experience that works seamlessly across all devices — from mobile phones to desktops.

Built using Next.js, React, and TypeScript, the frontend ensures high performance, modular design, and long-term maintainability. Next.js offers server-side rendering and efficient routing, which enhances page loading speed and overall performance. React allows the creation of reusable and dynamic UI components, enabling smooth transitions and real-time updates without reloading the entire page. The integration of TypeScript improves code reliability by providing static typing and better debugging support during development.

For design and layout, CSS Modules and Tailwind CSS are used to create a modern, clean, and responsive user interface. These tools help maintain consistency and scalability while ensuring that the application adapts perfectly to different screen sizes. HTML5 is used for structural elements and semantic layouts, making the design accessible and organized.

The frontend communicates securely with the backend through Next.js API routes, handling user authentication, data updates, and chat interactions efficiently. This setup allows for real-time messaging, event updates, and a personalized experience for each user.

Technologies Used

- Next.js (v15.5.3): Provides server-side rendering, fast routing, and optimized builds.
- React (v19.1.0): Handles dynamic and reusable UI components for smooth interaction.
- TypeScript: Adds type safety, improves maintainability, and reduces errors.

- CSS / CSS Modules: Defines consistent styles and component-level customization.
- Tailwind CSS: Enables responsive, modern, and visually appealing layouts.
- HTML5: Provides semantic structure and supports accessibility standards.

Key Features

- Fully responsive design for all screen sizes (desktop, tablet, and mobile).
- Smooth navigation and fast page transitions using Next.js routing.
- Real-time updates for chats, notifications, and group interactions.
- Secure frontend-backend communication for user authentication and data handling.
- Visually consistent UI with reusable components and minimal reloading.

In summary, the frontend of KJ_Connect combines modern technologies and efficient design principles to create an attractive, interactive, and high-performing interface. It ensures that students can easily connect, collaborate, and explore opportunities within the platform while enjoying a visually appealing and seamless user experience.

4.2 Overview Backend

The backend of *KJ_Connect* forms the core functional layer of the application, responsible for handling all server-side operations, database management, and API communications. It ensures that the data flow between the frontend and database is smooth, secure, and reliable. The backend manages key processes such as user authentication, group creation, real-time chat handling, and event management, making the system efficient and interactive for users.

Developed using Node.js and Next.js API routes, the backend provides a fast and scalable runtime environment capable of handling multiple concurrent requests from thousands of students. Node.js executes JavaScript code on the server side, processing user inputs and requests efficiently. Next.js API routes are used to define RESTful endpoints that manage operations such as registration, login validation, group creation, and message retrieval.

The MySQL database (managed locally via XAMPP) is used to store and manage structured data — including user profiles, group details, chat messages, and event updates. The connection between Node.js and MySQL is established using the mysql2 package, which offers asynchronous and optimized database queries. To ensure security, bcryptjs is used for password hashing, while dotenv manages sensitive configuration variables like database credentials and API keys.The backend is designed for security, performance, and scalability, using JSON Web Tokens (JWT) to manage user sessions and prevent unauthorized access. It also supports real-time functionalities such as chatting and group updates, ensuring an engaging and responsive user experience.

Technologies Used

- Node.js: Server-side runtime environment for executing JavaScript code.
- Next.js API Routes: Used to define backend endpoints for authentication and data operations.
- MySQL (via XAMPP): Relational database for storing user, group, and chat data.
- mysql2: Node.js library for fast and efficient database connectivity.
- bcryptjs: Ensures secure password hashing for user authentication.
- dotenv: Manages environment variables securely.
- JWT (JSON Web Token): Handles secure session management and authorization.
- OpenAI SDK (optional): Enables AI-based features like interest-based group suggestions.

Key Functions of the Backend

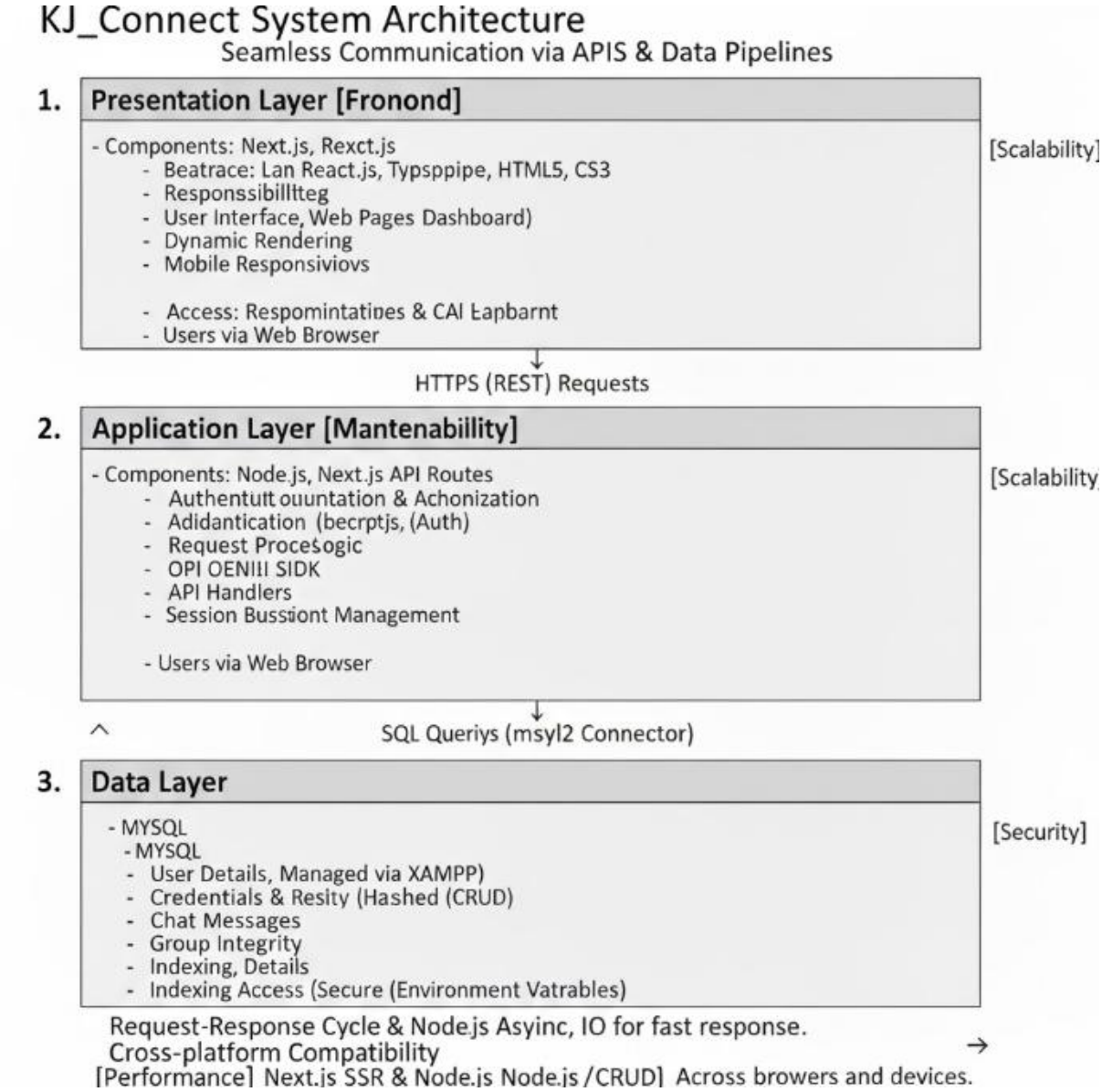
- User Authentication: Handles registration, login, and password encryption securely.
- Database Management: Stores and retrieves user profiles, messages, and event data.
- Group Management: Allows creation, joining, and updating of interest-based groups.
- Real-Time Chat Handling: Manages user-to-user and group messaging efficiently.
- API Integration: Facilitates communication between the frontend and database.
- Security Enforcement: Protects data using JWT tokens and hashed passwords.
- Error Handling & Logging: Monitors and records errors to ensure smooth performance.

In summary The backend of KJ_Connect serves as the backbone of the system, ensuring that all operations — from authentication to chat functionality — work seamlessly. With Node.js and MySQL at its core, it delivers a robust, secure, and scalable infrastructure that efficiently supports thousands of students. By integrating advanced technologies and best security practices, the backend ensures that KJ_Connect remains reliable, fast, and safe for all its users within the Kristu Jayanti College community.

5. SYSTEM DESIGN

The system design of *KJ_Connect* outlines how the components of the web application interact with each other to deliver a seamless and efficient user experience. It includes both architectural and functional design aspects — defining how data flows, how users interact with the system, and how backend services process requests. The design ensures scalability, security, and smooth integration between the frontend and backend components.

5.1 Architectural Diagram



5.1 Figure Architectural diagram

The system architecture of *KJ_Connect* is designed with a modern three-tier web architecture, ensuring high scalability, maintainability, and security. The three primary layers—Presentation

Layer, Application Layer, and Data Layer—work together seamlessly to provide a responsive and efficient platform for the college community. Each layer has a distinct responsibility and interacts through clearly defined interfaces and APIs.

1. Presentation Layer (Frontend Layer)

The Presentation Layer is responsible for all user interactions and visual elements of the application. It is developed using Next.js (React framework) with TypeScript, ensuring a dynamic, component-based, and fast-loading interface.

- Technologies Used: Next.js, React.js, TypeScript, HTML5, CSS3, JavaScript
- Key Responsibilities:
    - Displaying web pages for login, registration, dashboard, chat, and event updates.
    - Handling user inputs and sending API requests to the backend.
    - Rendering responses dynamically using React components.
    - Ensuring mobile responsiveness through CSS modules and flexible layouts.
    - Managing state efficiently across pages using React hooks and context APIs.

This layer is directly accessible to users through a web browser, providing a seamless interface to create groups, chat, or explore events. It communicates with the backend via Next.js API routes using secure HTTP (REST) requests.

2. Application Layer (Backend Layer)

The Application Layer is the core logic handler of *KJ_Connect*. It acts as an intermediary between the frontend and the database, processing all incoming requests, performing authentication, executing business logic, and returning structured responses.

- Technologies Used: Node.js, Next.js API Routes, bcryptjs, OpenAI SDK (for smart features)
- Key Responsibilities:
    - Managing user authentication and authorization using hashed passwords.
    - Handling requests for group creation, joining, and event management.
    - Performing validations and data processing before interacting with the database.
    - Facilitating chat functionality through message APIs.
    - Managing sessions securely using tokens and environment-based configurations.

The backend uses Next.js server-side routes for RESTful APIs and handles asynchronous operations using Node.js's event-driven model. This ensures fast and non-blocking performance, even under multiple concurrent user requests.

3. Data Layer (Database Layer)

The Data Layer serves as the foundation for storing and retrieving all persistent data in the system. It uses MySQL, managed through XAMPP, providing a reliable and relational data structure for all entities.

- Technologies Used: MySQL, mysql2 (Node.js connector), SQL Scripts
- Key Responsibilities:
    - Storing user details, authentication credentials, chat messages, and group information.
    - Ensuring data integrity through normalization and relational constraints.
    - Handling CRUD (Create, Read, Update, Delete) operations through SQL queries.
    - Providing efficient indexing for faster search and retrieval of user or group data.
    - Maintaining security using limited database access through environment variables.

The database is connected to the backend using the mysql2 package, which allows prepared statements to prevent SQL injection and ensures secure and optimized query execution.

4. Integration and Communication

All three layers interact through well-defined APIs and data flow pipelines:

- The frontend sends requests (like login, group creation, or chat messages) to the backend APIs.
- The backend processes the request, performs necessary operations, and interacts with the MySQL database.
- The database returns the required information, which the backend then formats and sends back to the frontend for display.

This request-response cycle ensures that every user interaction—whether it's logging in, chatting, or exploring events—flows smoothly and securely through the system.

5. Advantages of the Architecture

- Scalability: Each layer can be upgraded or scaled independently as the number of users grows.
- Security: Data access is restricted through authentication, HTTPS communication, and encrypted storage.
- Maintainability: Separation of concerns allows for easy updates or bug fixes in one layer without affecting others.
- Performance: Next.js's server-side rendering (SSR) and Node.js's asynchronous I/O

provide fast response times.

- Cross-platform Compatibility: The system works seamlessly across browsers and devices, ensuring accessibility for all students.

5.2 Input Design

5.2.1 Registration page

A clear, step-driven form that collects only required fields (name, email, password, confirm password) with inline validation (format checks, password strength meter, realtime feedback). Use progressive disclosure for optional profile info (avatar, interests) and show contextual helper text and privacy hints next to sensitive controls. Make controls keyboard-accessible, label every input, and show focus states; validate on blur and prevent submission until required inputs are valid.

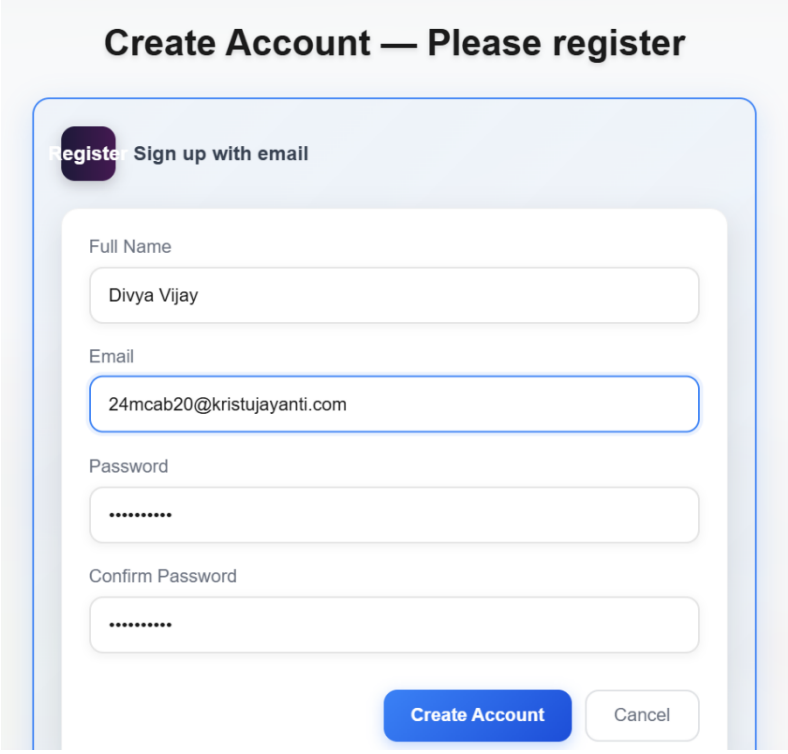


Figure 5.2 – Registration Page

5.2.2 Login page

A compact, single-column form with email/username and password fields, a prominent primary action, and auxiliary controls (Remember me, Forgot password). Provide immediate field-level feedback for incorrect credentials, support social SSO buttons where applicable, and rate-limit UI feedback for security. Ensure accessible labels, visible

focus outlines, and ARIA live announcements for authentication errors.

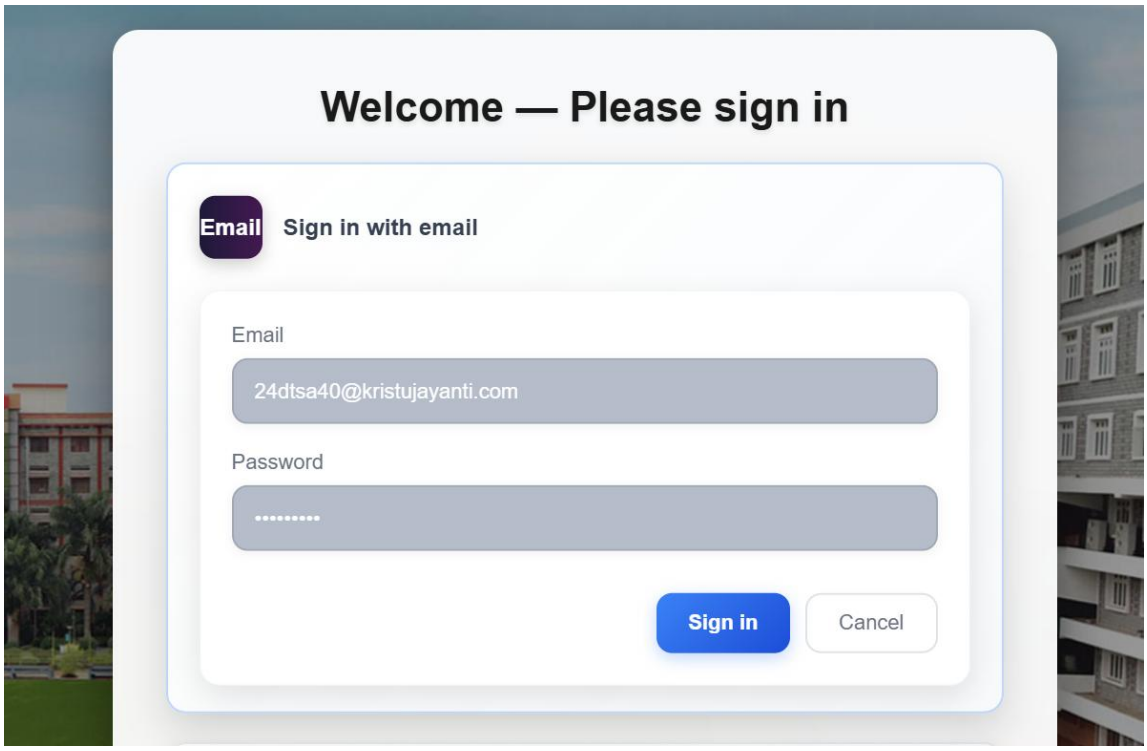


Figure 5.3. – Login Page

5.3 Output Design

5.3.1 Kristu Jayanti domain restriction message

When a user attaches an image to a message, show a thumbnail preview with file name, size, and simple edit tools (crop/rotate/remove). Display an optimized upload progress indicator and an accessibility caption/alt-text input so recipients and assistive tech can understand the image. After upload, render the image inline in the message with a lightweight lightbox/viewer on click and a fallback placeholder if loading fails.

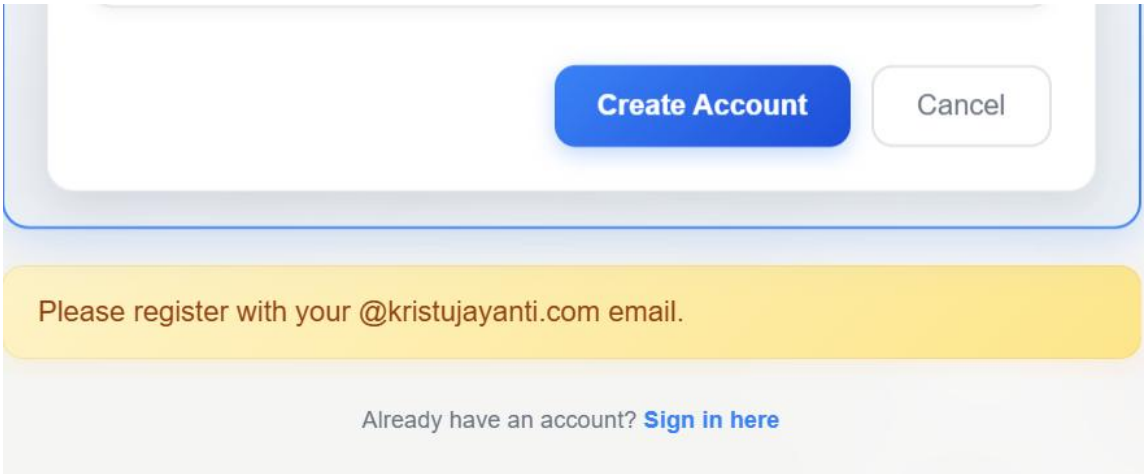


Figure 5.4 – Domain Restriction Message

5.3.2 Registration successful message

After sign-up, show a brief, dismissible confirmation banner or toast: concise success copy, next-step CTA (e.g., “Complete profile” or “Go to dashboard”), and an unobtrusive link to email verification or help. Persist a non-blocking success state and expose it to screen readers (ARIA live) so assistive tech announces the result; provide an undo or contact support option if account creation fails or needs changes.

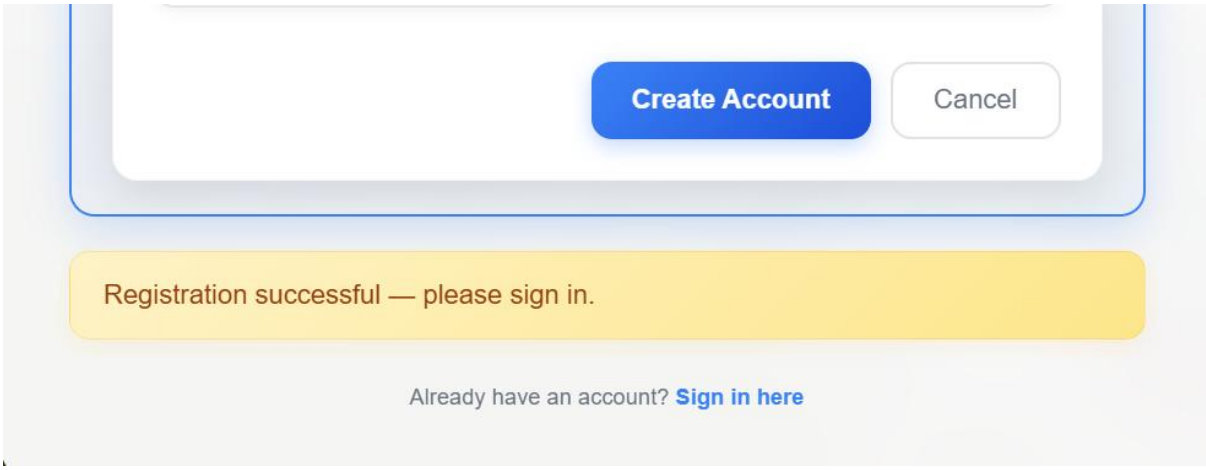


Figure 5.5 – Registration Successful Message

5.3.3 Feed/Dashboard

Present posts as consistent cards with author avatar/name, group tag, timestamp, text, and media; prioritize readability with clear hierarchy and generous spacing. Actions (Like, Comment, Share) are visible and update optimistically; support inline comments preview and an expand-to-thread interaction. Implement lazy-loading / infinite scroll with skeleton loaders, show real-time updates unobtrusively, and ensure all interactive elements are keyboard and screen-reader accessible.

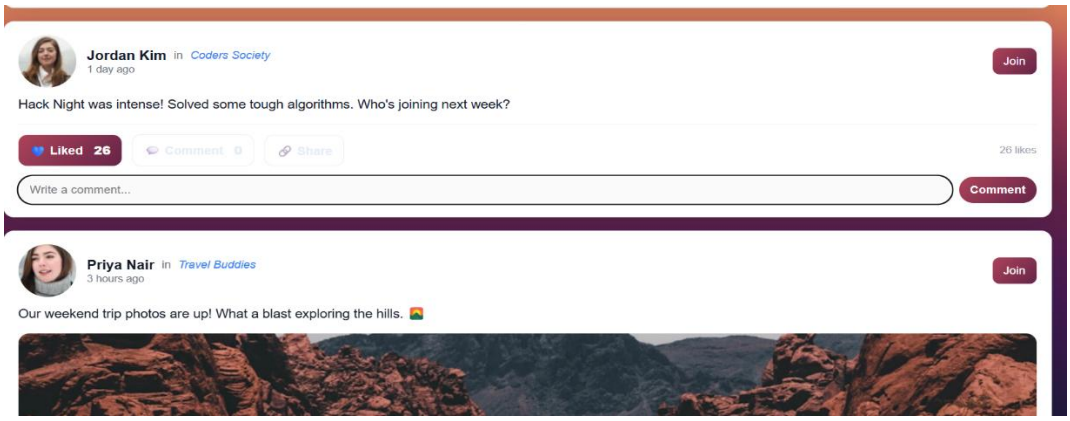


Figure 5.6 – Feed/Dashboard Page

5.4 Database Design

5.4.1 Users Table

This table stores all registered user details collected from registration form and profile page.

Field Name	Data Type	Constraints/ Description
Id	Int	Primary Key
Email	Varchar(255)	Unique, users login email
Password_hash	Varchar(255)	Encrypted password
Name	Varchar(255)	Users name
Dob	Date	Date of birth
Age	Int	Age of the user
Gender	ENUM(male, female, not prefer to say)	Gender
Department	Varchar(210)	Department
Created_at	Timestamp	Timestamp of creation
Updated_at	Timestamp	Timestamp of updation

Table 5.1: Users Table

5.4.2 Session Table

This table stores the session details of a user login and expiry, and as tokens

Field Name	Data Type	Constraints/Description
Id	Int	Primary Key
User_id	Int	Foreign Key --> users(id)
Token	Varchar(44)	Unique
Created_at	Timestamp	Timestamp of creation
Expires_at	Timestamp	Timestamp of expiry

Table 5.2: Session Table

5.4.3 Groups Table

This table stores the groups created details, members details and contents, group table stores information related to groups.

Field Name	Data Type	Constraints/Description
Id	Int	Primary Key
Name	Varchar(255)	Name of the group
Description	Text	Description of group
Interest	Varchar (150)	Based on Interest
Creator_id	Timestamp	Timestamp of creator
Created_at	Timestamp	Timestamp of creation

Table 5.3: Groups Table

5.4.4 Group_members Table

This table stores the details of the members or users existing or joined in a group thus this keeps and stores data.

Field Name	Data Type	Constraints/Description
Id	Int	Primary key
Group_id	Int	Forgien key --> groups(id)
User_id	int	Forgien key --> users(id)
Joined_at	Timestamp	Timestamp of users joined

Table 5.4: Group_members Table

5.4.5 User_interest Table

This table stores the details of users interest where which group or activity can be shown to explore for users.

Field Name	Data Type	Constraints/Description
Id	Int	Primary key
User_id	Int	Forgien key --> users(id)
Interest	Varchar(200)	Interest based on user

Table 5.5: User_interest Table

5.4.6 Messages Table

This table is used to store sender and receiver user details along with the content, time, if the messages is read or not.

Field Name	Data Type	Constraints/Description
Id	Int	Primary Key
Sender_id	Int	Foreign key --> users(id)
Reciver_id	Int	Foreign key --> users(id)
Group_id	Int	Foreign key --> groups(id)
Content	Text	Any text content sent between users or in groups
Created_at	Timestamp	Timestamp of creation
Is_read	Boolean	Messages read or not by users

Table 5.6: Messages Table

6. SOURCE CODE

Login Page

```
"use client";
import React, { useState, useRef, useEffect } from "react";
import Link from 'next/link';
import styles from "./page.module.css";

export default function LoginPage() {
  const [expanded, setExpanded] = useState<string | null>(null);
  const [email, setEmail] = useState("");
  const [password, setPassword] = useState("");
  const [message, setMessage] = useState<string | null>(null);
  const emailRef = useRef<HTMLInputElement | null>(null);

  const toggle = (key: string) => {
    setMessage(null);
    setExpanded(prev => (prev === key ? null : key));
  };

  const submitEmail = (e: React.FormEvent) => {
    e.preventDefault();
    if (!email) {
      setMessage("Please enter your email.");
      return;
    }
    if (!password) {
      setMessage("Please enter your password.");
      return;
    }

    // Enforce domain: only allow @kristujayanti.com
    const domain = '@kristujayanti.com';
    if (!email.toLowerCase().endsWith(domain)) {
```

```
    setMessage(`Please sign in with your ${domain} email.`);
    return;
  }

  // Call server auth route
  fetch('/api/auth', { method: 'POST', headers: { 'Content-Type': 'application/json' }, body:
JSON.stringify({ email, password }) })
    .then(async res => {
      const data = await res.json();
      if (!res.ok) throw new Error(data?.error || 'Login failed');
      // On success the server sets a cookie; redirect to dashboard
      window.location.href = '/dashboard';
    })
    .catch(err => setMessage(String(err)));
 };

 return (
  <div className={styles.container}>
   <div className={styles.card}>
    <h1 className={styles.title}>Welcome — Please sign in</h1>

    <ul className={styles.options}>
     <li
      className={`${styles.option} ${expanded === 'email' ? styles.active : ''}`}
      onClick={() => toggle('email')}
      role="button"
      tabIndex={0}
     >
      <div className={styles.optionHeader}>
       <span className={styles.badge} style={{ background: 'linear-gradient(90deg,var(--
pal-6),var(--pal-5))' }}>Email</span>
       <span className={styles.optionText}>Sign in with email</span>
      </div>
```

```
          {expanded === 'email' && (
           <form  className={styles.panel}  onSubmit={submitEmail}  onClick={(e)  =>
e.stopPropagation()}>
            <label className={styles.label}>
             <span style={{color:'var(--muted)'}}>Email</span>
             <input ref={emailRef} className={styles.input} value={email} onChange={e =>
setEmail(e.target.value)} type="email" placeholder="rollno@kristujayanti.com" />
            </label>
            <label className={styles.label}>
             <span style={{color:'var(--muted)'}}>Password</span>
             <input  className={styles.input}  value={password}  onChange={e  =>
setPassword(e.target.value)} type="password" />
            </label>
            <div className={styles.actions}>
             <button className={styles.submit} type="submit">Sign in</button>
             <button  type="button"  className={styles.cancel}  onClick={()  =>
setExpanded(null)}>Cancel</button>
            </div>
           </form>
          )}
        </li>

        <li
          className={`${styles.option} ${expanded === 'google' ? styles.active : ''}`}
          onClick={() => toggle('google')}
          role="button"
          tabIndex={0}
        >
          <div className={styles.optionHeader}>
           <span className={styles.badge} style={{ background: 'linear-gradient(90deg,var(--
pal-2),var(--pal-1))' }}>G</span>
           <span className={styles.optionText}>Continue with Google</span>
          </div>
          {expanded === 'google' && (
```

```
            <div className={styles.panel} onClick={(e) => e.stopPropagation()}>
              <p style={{color:'var(--foreground-dark)'}}>Google login is mocked in this demo. Click to continue.</p>
              <div className={styles.actions}>
                <button className={styles.submit} onClick={() => setMessage('Google sign-in (mock)')}>Continue</button>
              </div>
            </div>
          )}
        </li>

        {/* GitHub option removed per request */}
      </ul>

      {message && <div className={styles.message}>{message}</div>}

      <div className={styles.footer}>
        <small>New here? <Link href="/register">Create an account</Link></small>
      </div>
    </div>
  </div>
 );
}
```

Profile Page

```
"use client";

import React, { useState, useEffect } from 'react';
import { useRouter } from 'next/navigation';
import styles from '../login/page.module.css';

interface User {
 id: number;
 email: string;
```

```
  name: string;
  dob: string;
  age: number;
  gender: string;
  department: string;
}

export default function ProfilePage() {
  const [user, setUser] = useState<User | null>(null);
  const [loading, setLoading] = useState(true);
  const [saving, setSaving] = useState(false);
  const [message, setMessage] = useState<string | null>(null);
  const router = useRouter();

  // Form state
  const [formData, setFormData] = useState({
    name: '',
    dob: '',
    age: '',
    gender: 'prefer_not_to_say',
    department: ''
  });

  useEffect(() => {
    fetchUserData();
  }, []);

  const fetchUserData = async () => {
    try {
      const response = await fetch('/api/user');
      if (response.status === 401) {
        router.push('/login');
        return;
      }
```

```
      const data = await response.json();
      if (data.user) {
        setUser(data.user);
        setFormData({
          name: data.user.name || '',
          dob: data.user.dob || '',
          age: data.user.age || '',
          gender: data.user.gender || 'prefer_not_to_say',
          department: data.user.department || ''
        });
      }
    } catch (error) {
      console.error('Error fetching user data:', error);
      setMessage('Error loading profile data');
    } finally {
      setLoading(false);
    }
  };

  const handleInputChange = (e: React.ChangeEvent<HTMLInputElement | HTMLSelectElement>) => {
    const { name, value } = e.target;
    setFormData(prev => ({
      ...prev,
      [name]: value
    }));
  };

  const calculateAge = (dob: string) => {
    if (!dob) return '';
    const today = new Date();
    const birthDate = new Date(dob);
    let age = today.getFullYear() - birthDate.getFullYear();
```

```
  const monthDiff = today.getMonth() - birthDate.getMonth();
  if (monthDiff < 0 || (monthDiff === 0 && today.getDate() < birthDate.getDate())) {
    age--;
  }
  return age.toString();
};

const handleDobChange = (e: React.ChangeEvent<HTMLInputElement>) => {
  const dob = e.target.value;
  setFormData(prev => ({
    ...prev,
    dob,
    age: calculateAge(dob)
  }));
};

const handleSubmit = async (e: React.FormEvent) => {
  e.preventDefault();
  setSaving(true);
  setMessage(null);

  try {
    const response = await fetch('/api/user', {
      method: 'PUT',
      headers: {
        'Content-Type': 'application/json',
      },
      body: JSON.stringify(formData),
    });

    if (response.status === 401) {
      router.push('/login');
      return;
    }
```

```
    const data = await response.json();
    if (data.ok) {
      setMessage('Profile updated successfully!');
      // Update user state
      if (user) {
        setUser({
          ...user,
          ...formData,
          age: parseInt(formData.age) || 0
        });
      }
    } else {
      setMessage(data.error || 'Update failed');
    }
  } catch (error) {
    console.error('Error updating profile:', error);
    setMessage('Error updating profile');
  } finally {
    setSaving(false);
  }
};

const handleLogout = async () => {
  try {
    await fetch('/api/auth', { method: 'DELETE' });
    router.push('/login');
  } catch (error) {
    console.error('Logout error:', error);
  }
};

if (loading) {
  return (
```

```
    <div className={styles.container}>
      <div className={styles.card}>
        <h1 className={styles.title}>Loading Profile...</h1>
      </div>
    </div>
  );
 }

 return (
  <div className={styles.container}>
    <div className={styles.card}>
     <div style={{ display: 'flex', justifyContent: 'space-between', alignItems: 'center', marginBottom: '30px' }}>
       <h1 className={styles.title}>Profile Settings</h1>
       <button
         onClick={handleLogout}
         style={{
           background: 'linear-gradient(135deg, #ef4444, #dc2626)',
           color: 'white',
           border: 'none',
           padding: '8px 16px',
           borderRadius: '8px',
           cursor: 'pointer',
           fontWeight: '500'
         }}
        >
         Logout
       </button>
     </div>

     <form className={styles.panel} onSubmit={handleSubmit}>
       <div style={{ display: 'grid', gridTemplateColumns: '1fr 1fr', gap: '20px' }}>
         <label className={styles.label}>
           <span style={{color:'var(--muted)'}}>Email</span>
```

```
            <input
              className={styles.input}
              value={user?.email || ''}
              disabled
              style={{ backgroundColor: '#f3f4f6', color: '#6b7280' }}
            />
          </label>

          <label className={styles.label}>
            <span style={{color:'var(--muted)'}}>Full Name</span>
            <input
              className={styles.input}
              name="name"
              value={formData.name}
              onChange={handleInputChange}
              placeholder="Enter your full name"
            />
          </label>
        </div>

        <div style={{ display: 'grid', gridTemplateColumns: '1fr 1fr', gap: '20px' }}>
          <label className={styles.label}>
            <span style={{color:'var(--muted)'}}>Date of Birth</span>
            <input
              className={styles.input}
              name="dob"
              type="date"
              value={formData.dob}
              onChange={handleDobChange}
            />
          </label>

          <label className={styles.label}>
            <span style={{color:'var(--muted)'}}>Age</span>
```

```
        <input
          className={styles.input}
          name="age"
          type="number"
          value={formData.age}
          onChange={handleInputChange}
          placeholder="Auto-calculated from DOB"
          min="1"
          max="120"
        />
      </label>
    </div>

    <div style={{ display: 'grid', gridTemplateColumns: '1fr 1fr', gap: '20px' }}>
      <label className={styles.label}>
        <span style={{color:'var(--muted)'}}>Gender</span>
        <select
          className={styles.input}
          name="gender"
          value={formData.gender}
          onChange={handleInputChange}
        >
          <option value="prefer_not_to_say">Prefer not to say</option>
          <option value="male">Male</option>
          <option value="female">Female</option>
          <option value="other">Other</option>
        </select>
      </label>

      <label className={styles.label}>
        <span style={{color:'var(--muted)'}}>Department</span>
        <input
          className={styles.input}
           name="department"
```

```
            value={formData.department}
            onChange={handleInputChange}
            placeholder="e.g., Computer Science, Electronics"
           />
          </label>
         </div>

         <div className={styles.actions}>
          <button
           className={styles.submit}
           type="submit"
           disabled={saving}
           style={{ opacity: saving ? 0.7 : 1 }}
          >
           {saving ? 'Saving...' : 'Update Profile'}
          </button>
          <button
           type="button"
           className={styles.cancel}
           onClick={() => router.push('/dashboard')}
          >
           Back to Dashboard
          </button>
         </div>
        </form>

        {message && <div className={styles.message}>{message}</div>}
      </div>
    </div>
  );
}
```

7. SYSTEM TESTING

System Testing is a crucial phase in the software development life cycle where the entire *KJ_Connect* web application is tested as a whole to verify that it meets all functional and non-functional requirements. The purpose of testing is to ensure that every module—from user registration to chat, group creation, and event updates—works correctly and integrates seamlessly with other components. It helps identify and eliminate bugs, performance issues, and security vulnerabilities before deployment.

1. Objective of System Testing

The main objectives of testing *KJ_Connect* include:

- Ensuring all modules (authentication, chat, group, events) work as expected.
- Validating smooth interaction between the frontend, backend, and database.
- Confirming that only authenticated users (college email domain) can access the system.
- Verifying system performance, usability, and scalability under different loads.
- Identifying and fixing defects before the final deployment.

2. Types of Testing Performed

To ensure reliability and quality, various testing techniques were applied during *KJ_Connect* development:

1.Unit Testing

- Each component (e.g., login form, registration form, chat module) was tested individually.
- Tools like **Jest** and **React Testing Library** were used to test frontend components.
- Backend routes (API endpoints) were tested using **Postman**.

2.Integration Testing

- Tested the interaction between frontend forms and backend APIs.
- Verified data flow between the application layer (Next.js APIs) and the MySQL database.
- Ensured the proper working of group and chat synchronization across users.

3.System Testing

- The entire *KJ_Connect* application was tested as a unified system.
- Checked whether the system met functional requirements such as user registration, login, chat, and event updates.
- Validated both normal and edge case behaviors (e.g., invalid inputs, duplicate registration).

4.User Acceptance Testing (UAT)

- Conducted with a small group of students and faculty to ensure usability and feature clarity.
- Feedback was used to refine the interface, simplify navigation, and enhance chat responsiveness.

5. Performance Testing

- Evaluated response time, load capacity, and speed during multiple user requests.
- Ensured database queries and server-side rendering were optimized for fast performance.

6.Security Testing

- Tested authentication mechanisms using **bcryptjs** to ensure password safety.
- Verified that only valid *@kristujayanti.com* email domains could register.
- Checked for protection against SQL Injection, Cross-Site Scripting (XSS), and CSRF attacks.

7.1 Test Case

Module Registration

TC ID	Description	Input	Expected	Actual	Status
TC001	Newuser registration	Name:"Test", Email:roll@kristujayanti.com, Password: *******, Confirm password:*******	Account created	Works as expected	passed
TC002	Existing email	Name:"Test", Email:roll@kristujayanti.com, Password: *******, Confirm password:*******	Error msg:User already exists	Works as expected	passed

Table 7.1 Registration Module Test case

Module: login

TC ID	Description	Input	Excepted	Actual	Status
TC003	Valid login	Email:rollno@kristujayanti.com, Password: *******	Redirects to home page	Redirects to home page	**Passed**
TC003	Invalid Password	Email:rollno@kristujayanti.com, Password: wrong password	Error msg: invalid password	Works as expected	passed

Table 7.2 Login Module Test case

Module: message

TC ID	Description	Input	Excepted	Actual	Status
TC004	User message	Send hi to any user	Message gets delivered	Works as expected	passed
TC005	Create group	Create a group with grp name, des, interest	A group will be created	Works as expected	passed
TC006	Join grp	Join group created by other user	Will be able join	Works as expected	passed
TC007	Grp message	Send hi in group chat	Will be able send message in grp	Works as expected	passed

Table 7.3 Message Module Test case

8. SYSTEM IMPLEMENTATION

System Implementation is the stage where the *KJ_Connect* application is deployed and made operational after successful design and testing. This phase ensures that all components of the system — frontend, backend, and database — are correctly configured, integrated, and optimized for real-world use. The implementation focuses on stability, performance, and providing a seamless experience for students and administrators of Kristu Jayanti College.

1. Objective of System Implementation

The main objectives of implementing *KJ_Connect* are to:

- Integrate all modules into a single, functioning web platform.
- Deploy the application for real-time usage and interaction.
- Ensure data security and proper access control through verified authentication.
- Provide a responsive and scalable environment for future updates.

2. Implementation Process

The implementation of *KJ_Connect* was carried out in multiple stages to ensure smooth integration of different modules and technologies.

a) Environment Setup

- Installed Node.js and Next.js for full-stack JavaScript development.
- Configured XAMPP for hosting the MySQL database locally.
- Created a .env.local file to securely store environment variables (DB credentials, API keys).
- Set up Turbopack for faster builds and development efficiency.

b) Database Setup

- Designed the database schema with tables like users, groups, messages, and events.
- Established primary and foreign keys for maintaining data relationships.
- Used SQL scripts for table creation and data migration.
- Connected the database using the mysql2 Node.js library.

Frontend Implementation

The frontend of *KJ_Connect* provides the user interface and handles all interactions between the user and the system. It was built using Next.js (React Framework) with TypeScript, HTML, CSS, and JavaScript to create a responsive and modern interface.

Frontend Features

- User Interface Design: Designed using React components and Next.js pages for

modularity and ease of navigation.

- Responsive Layout: CSS Modules and media queries ensure compatibility with desktops, tablets, and mobile devices.
- Dynamic Routing: Implemented through Next.js to navigate seamlessly between pages (login, dashboard, chat, groups).
- Authentication Interface: Registration and login forms validate user inputs and connect to backend APIs.
- Dashboard and Chat UI: Displays groups, chat windows, and event updates dynamically based on user data.
- Error Handling & Validation: Frontend validation prevents incorrect inputs and enhances user experience.

Technologies Used

- Next.js v15.5.3 — Framework for server-side rendering and routing.
- React v19.1.0 — For building reusable UI components.
- TypeScript — Adds type safety and cleaner code.
- CSS Modules & global.css — For styling and responsive design.
- SVG Assets — For icons representing interests and categories.

Frontend Workflow

1. User opens the KJ_Connect application in a browser.
2. Frontend displays the login/register interface.
3. User actions trigger API calls to the backend for authentication or data retrieval.
4. The results (chat messages, groups, events) are dynamically rendered on the screen.

The frontend was thoroughly tested on major browsers like Chrome, Edge, and Firefox for consistency and performance.

Backend Implementation

The backend of *KJ_Connect* manages data processing, logic execution, and communication between the frontend and the database. It is built using Next.js API routes running on Node.js, ensuring fast and secure request handling.

Backend Features

- Authentication & Authorization: Uses bcryptjs for password encryption and validates only college domain emails.
- API Development: RESTful API endpoints handle operations such as login, registration, group management, and chat.

- Database Connectivity: Uses mysql2 for executing SQL queries and managing persistent data.
- Session Management: Maintains secure user sessions using tokens and cookies.
- Error & Exception Handling: Handles invalid inputs, missing data, and unauthorized access with proper error messages.
- Integration with OpenAI SDK: Provides future expansion for intelligent suggestions and message filtering.

Technologies Used

- Node.js — Server-side runtime for executing JavaScript code.
- Next.js API Routes — For handling HTTP requests and server-side logic.
- MySQL (via XAMPP) — Relational database for structured data management.
- mysql2 — Node.js library for MySQL integration.
- bcryptjs — Password hashing for secure authentication.
- dotenv — For environment variable management.

Backend Workflow

1. The frontend sends a request (e.g., login or group creation) via an API endpoint.
2. The backend validates the request, processes it, and interacts with the MySQL database.
3. The response (e.g., success message, user data) is returned to the frontend.
4. The frontend updates the user interface dynamically based on the backend's response.

3. Implementation Challenges

During implementation, a few challenges were encountered and successfully resolved:

- Database Connection Errors: Fixed by updating configuration parameters in .env.local.
- Authentication Issues: Implemented domain-based email validation to restrict access.
- Performance Optimization: Used caching and efficient query design to improve response time.
- CORS and Routing Conflicts: Managed through Next.js configuration and secure API headers.

4. Outcome of Implementation

After integration and testing, *KJ_Connect* successfully achieved:

- Seamless communication between frontend, backend, and database.
- Fast performance with secure authentication.
- Consistent user experience across devices and browsers.
- Reliable data management and real-time interaction for chat and events.

9. CONCLUSION AND FUTURE ENHANCEMENT

Conclusion

The *KJ_Connect* web application successfully addresses the challenge of limited student interaction and collaboration within Kristu Jayanti College. It provides a unified digital platform where students from various departments can connect, collaborate, and engage in activities based on shared interests. By leveraging secure authentication using the college's email domain, *KJ_Connect* ensures that only legitimate students can register and participate, maintaining the platform's integrity and safety. The integration of modern technologies such as Next.js**,** React**,** Node.js**,** and MySQL ensures a robust, responsive, and efficient system architecture. The application provides essential features like group creation**,** real-time chatting**,** interest-based connections**,** and event updates, thereby fostering networking, creativity, and collaboration among students. The seamless UI/UX design enhances usability, while secure backend operations guarantee data protection and reliability.

Overall, *KJ_Connect* bridges the communication gap among students, transforming how they interact beyond academic boundaries and helping them explore opportunities for creativity, innovation, and teamwork.

Future Enhancement

Although *KJ_Connect* fulfills its core objectives, there is significant scope for future improvement and scalability. Potential enhancements include:

1. **Mobile Application Integration:** Developing a cross-platform mobile app (Android/iOS) for easier accessibility and real-time notifications.
2. **AI-Based Recommendations:** Implementing AI algorithms to suggest groups, events, or collaborations based on a user's interests and past activity.
3. **Event Management Module:** Adding a system for creating, promoting, and managing college events within the app itself.
4. **Advanced Chat System:** Integrating multimedia messaging, voice/video calls, and message encryption for improved communication.
5. **Admin Dashboard:** Developing a comprehensive admin panel to monitor user activities, approve events, and manage content efficiently.

6. **Push Notifications:** Sending instant updates about new groups, events, or messages to keep users engaged.
7. **Cloud Deployment:** Migrating the system to a scalable cloud environment (e.g., AWS or Vercel) to support more users and enhance performance.
8. **Integration with College Portal:** Linking *KJ_Connect* with the official college portal for automated student verification and event synchronization.

With these future enhancements, *KJ_Connect* can evolve into a complete student networking ecosystem — not just connecting people, but empowering them to collaborate, innovate, and make lasting contributions within and beyond their college community.

10.APPENDICES

10.1 Screenshots

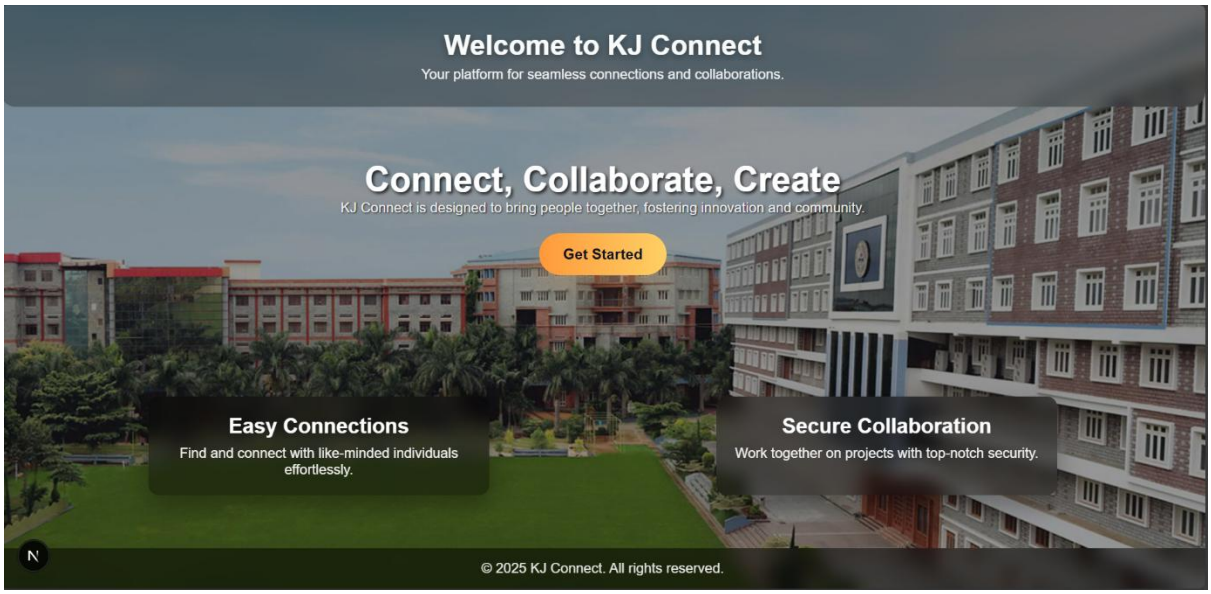


Figure - 10.1 - Landing page

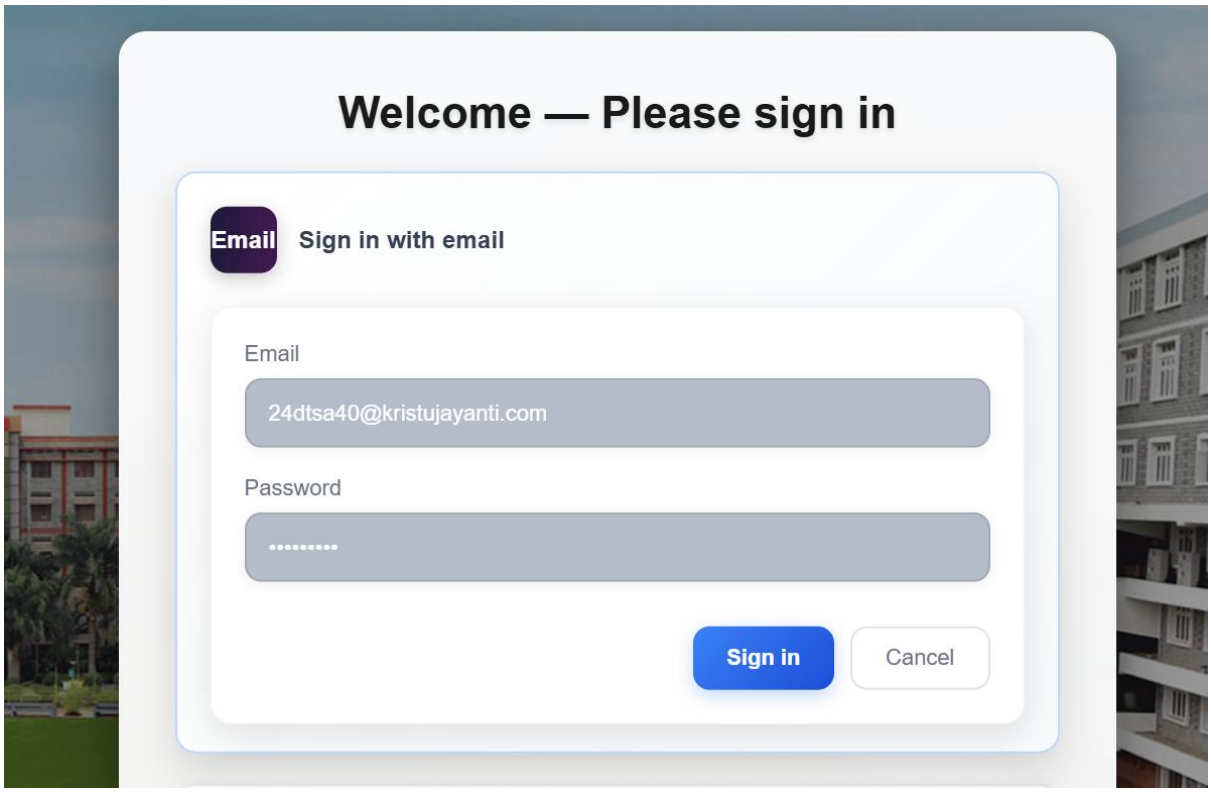


Figure - 10.2 - Login page

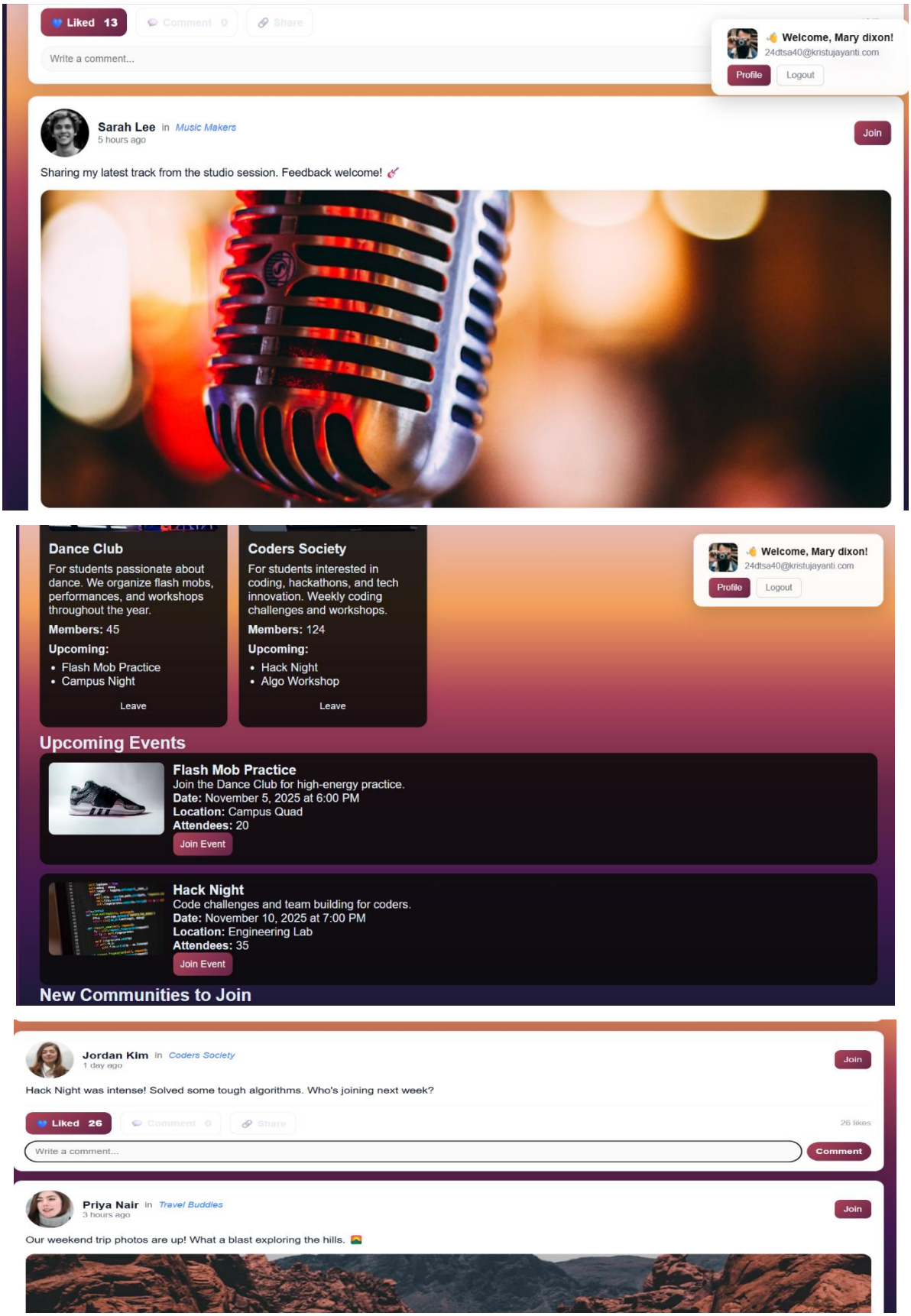


Figure - 10.3 - Home pages

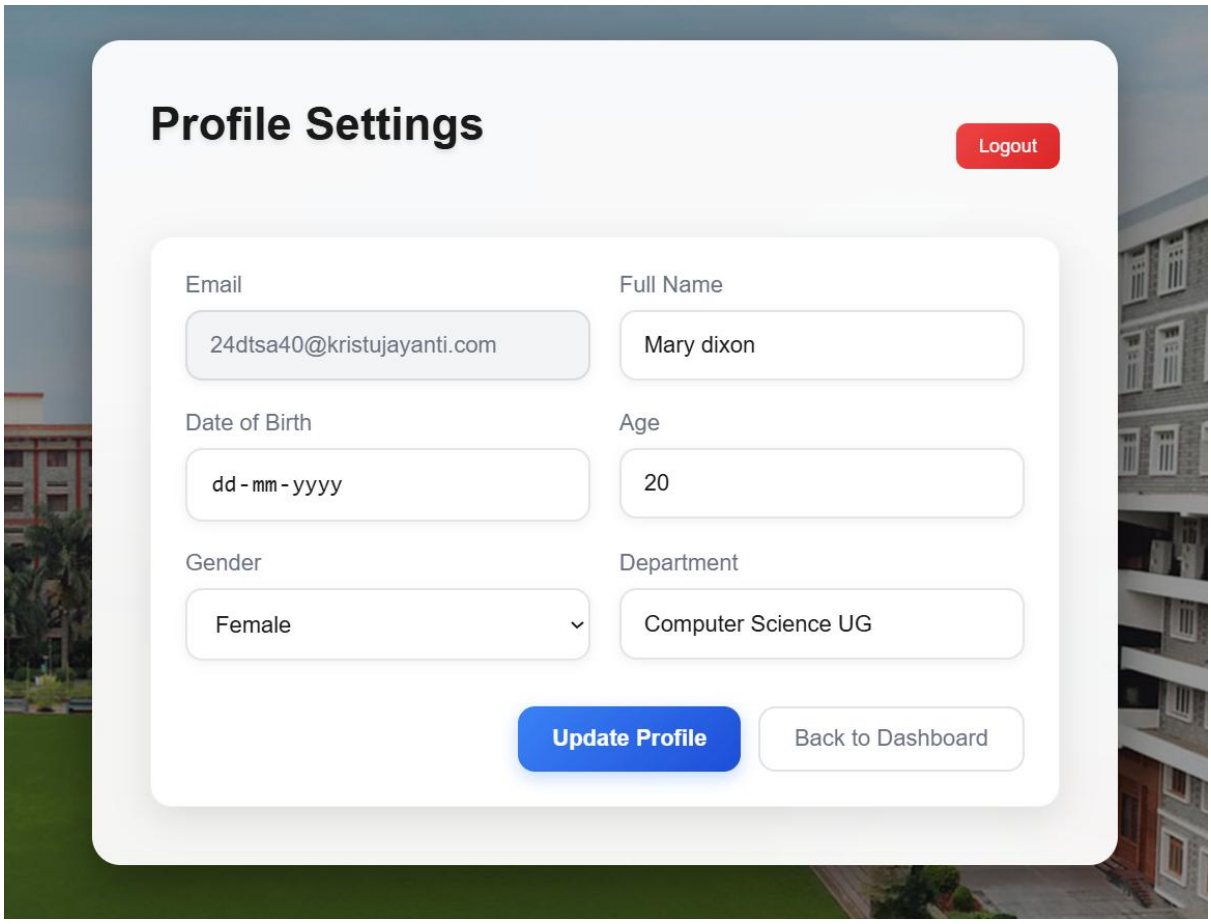


Figure - 10.4 - Profile edit page

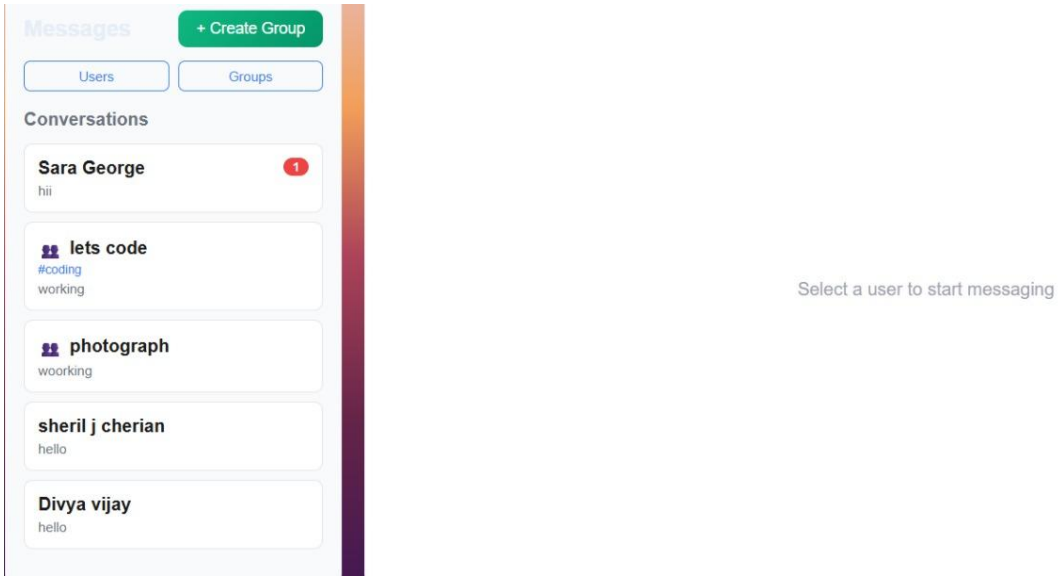


Figure - 10.5 – Message notification page

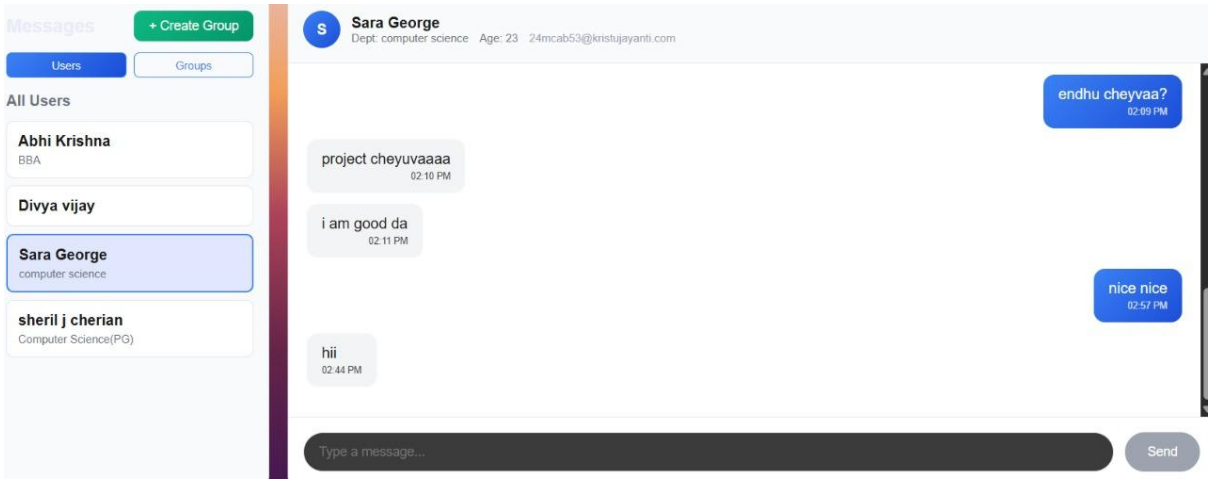


Figure - 10.6 - User messaging page

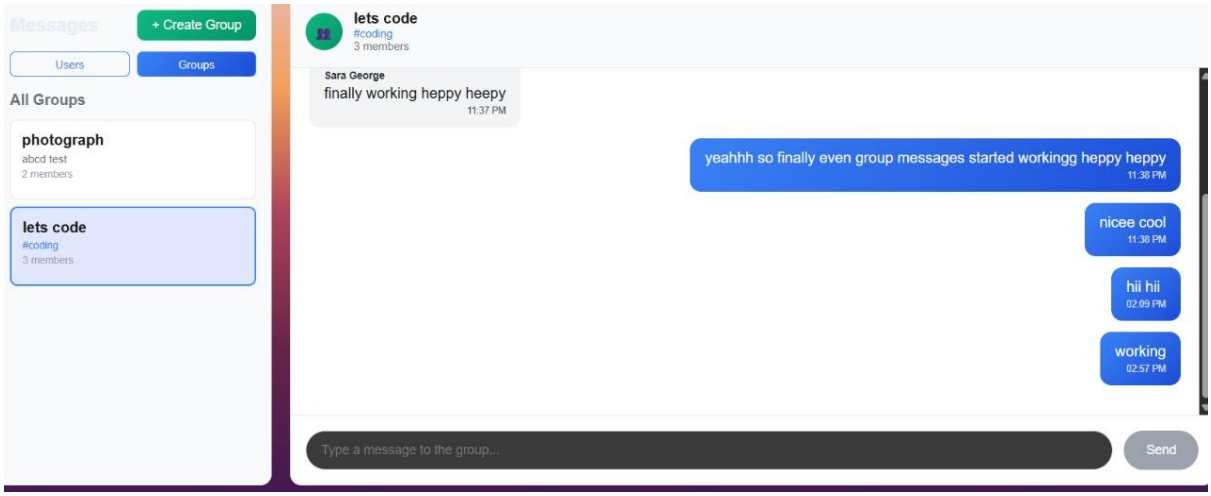


Figure - 10.7 - Group messages page

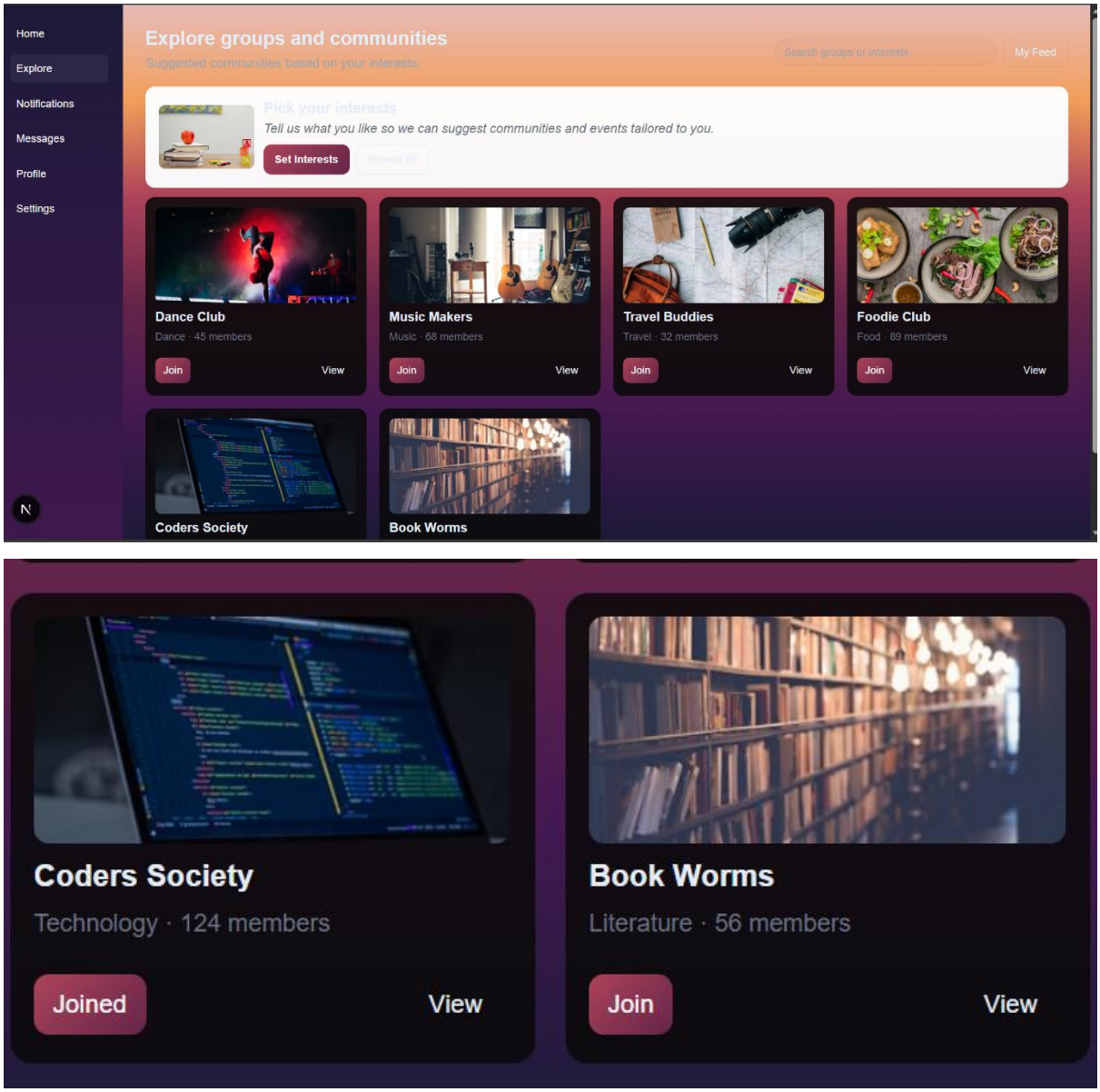


Figure - 10.8 - Explore communities page

Figure - 10.9 - Notifications page

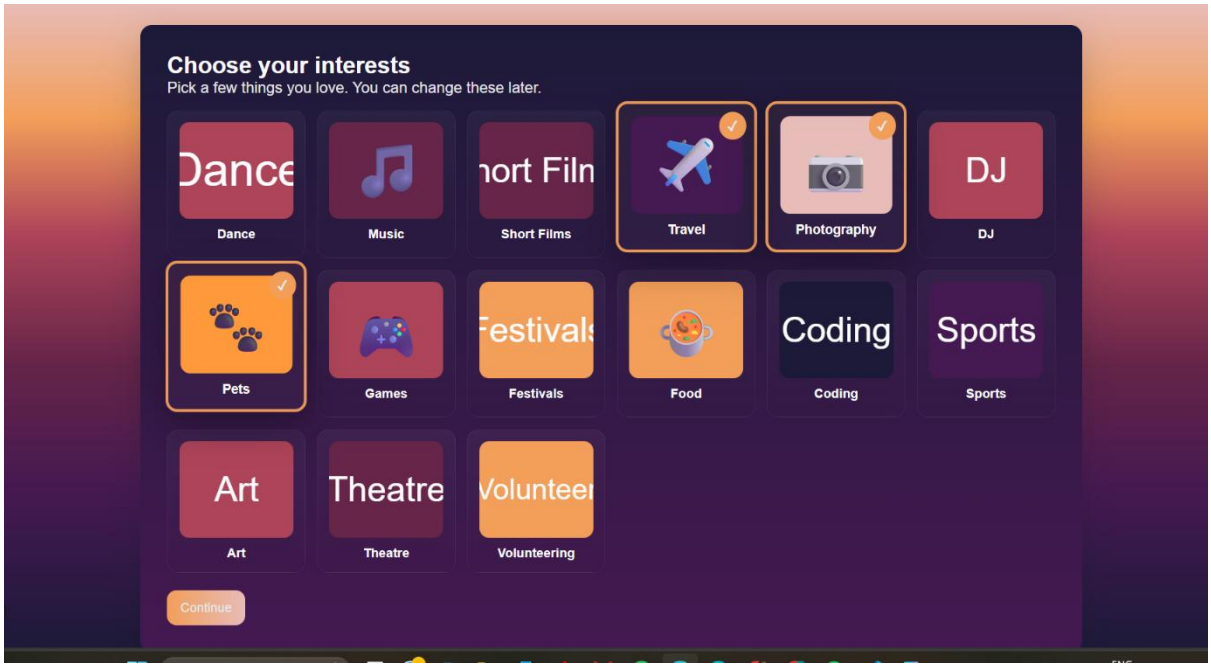


Figure - 10.10 - Interest creation page

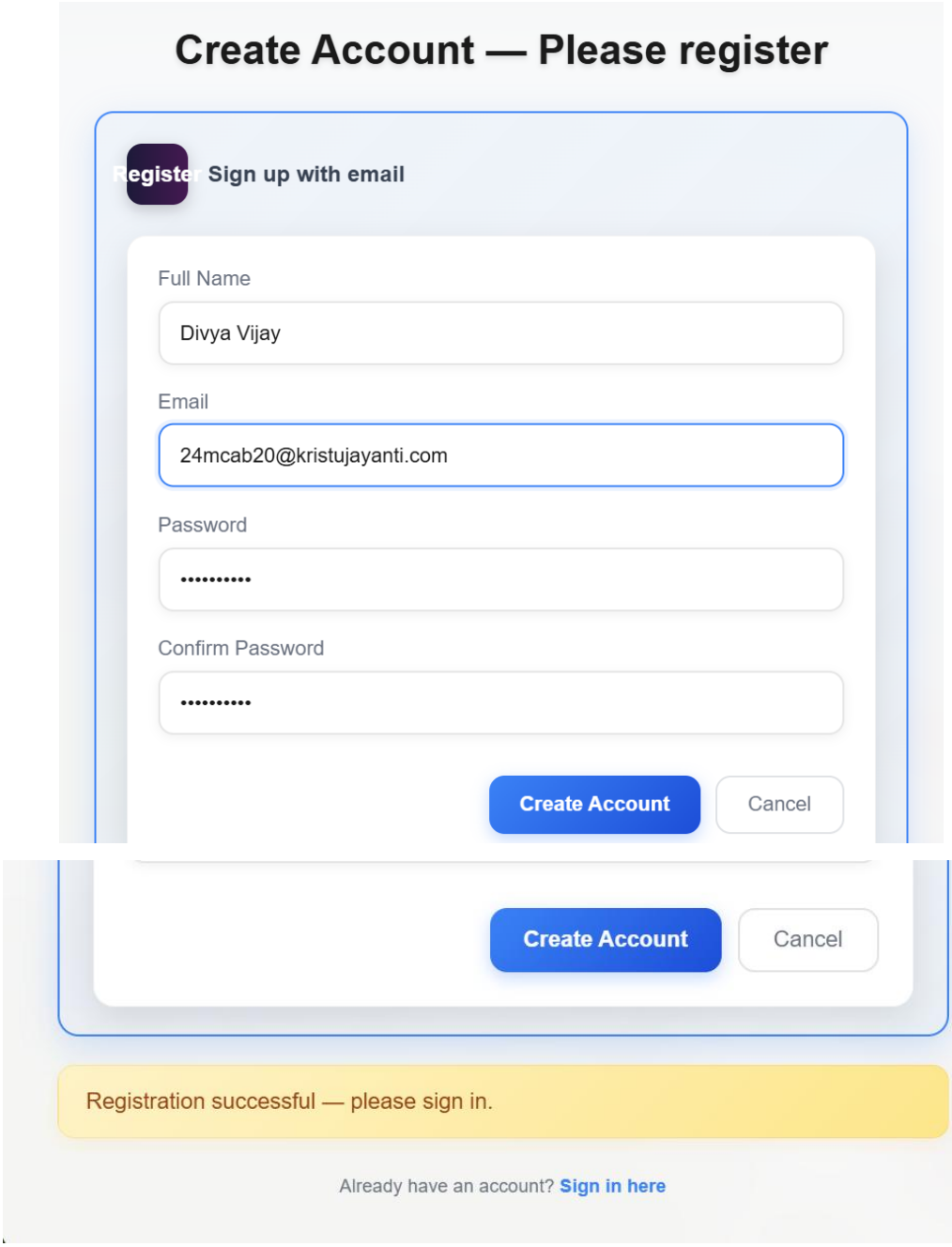


Figure - 10.11 - Create account page

11.REFERENCE

1. Next.js Documentation – The React Framework for Production.
   Retrieved from https://nextjs.org/docs

2. React Official Documentation – Declarative UI for Web Applications.
   Retrieved from https://react.dev/

3. TypeScript Documentation – TypeScript: JavaScript with Syntax for Types.
   Retrieved from https://www.typescriptlang.org/docs/

4. Node.js Documentation – Server-side JavaScript Runtime Environment.
   Retrieved from https://nodejs.org/en/docs

5. MySQL Documentation – Open-Source Relational Database Management System.
   Retrieved from https://dev.mysql.com/doc/

6. OpenAI API Documentation – AI and Natural Language Processing SDK.
   Retrieved from https://platform.openai.com/docs

7. bcryptjs Package – Password Hashing Library for Node.js.
   Retrieved from https://www.npmjs.com/package/bcryptjs

8. W3Schools – HTML, CSS, JavaScript, and Web Development Tutorials.
   Retrieved from https://www.w3schools.com/

9. MDN Web Docs – Comprehensive Web Technology Reference by Mozilla.
   Retrieved from https://developer.mozilla.org/

10. Kristu Jayanti College Official Website – Institutional Information and Updates.
    Retrieved from https://kristujayanti.edu.in/